1. Časť

Softvérový systém

Obsah

Úvod

Účel dokumentu

Predkladaný dokument obsahuje projektovú dokumentáciu k softvérovému systému, určenému pre realizovanie transakcií prostredníctvom mobilných zariadení. Dokument je výsledkom študentského tímového projektu v predmete Tímový projekt, realizovaný na Fakulte informatiky a informačných technológií STU v Bratislave a je určený pre zadávateľa projektu, ktorým je Ing. Michal Čerňanský, PhD.

Metodika vývoja

Vývoj sa riadi agilnou metodikou SCRUM, z čoho vyplýva viacero dôsledkov:

- vývoj prebieha v dvojtýždňových intervaloch – šprintoch
- šprinty sú zložené z príbehov (User Story)
- príbehy sú rozbité na úlohy (task)
- zložitosť príbehov je ohodnotená bodmi
- koniec každého šprintu je sprevádzaný odovzdaním riešení jednotlivých príbehov, naplánovaných pre daný šprint
- riešenie príbehu musí byť implementované, zdokumentované a otestované
- stretnutia tímu sú pravidelne raz za týždeň

Forma dokumentácie

Forma dokumentácie sa pridŕža pravidiel, určených v spomenutom predmete a obsah pozostáva z opisu jednotlivých šprintov a k nim prislúchajúcich príbehov (angl. User story). Opis každého príbehu obsahuje analýzu, návrh, opis implementácie a spôsob testovania jednotlivých príbehov. Taktiež je každý príbeh navyše rozbitý na úlohy.

Výber implementačného jazyka, prostredia a technológií

Implementácia je rozdelená do 2 častí:

- serverová časť
- klientska časť

Serverová časť

Cloudové služby – *Google App Engine* – výber tejto technológie bol sčasti výberom vedúceho tímu, ktorý nás k tejto technológii priamo nenútil, ale odporučil nám ju. Po dôkladnom štúdiu jednotlivých možností, ktoré sme mohli použiť (*Google App Engine, Amazon EC2, Microsoft Windows Azure, Eucalyptus*), sme sa rozhodli použiť Google App Engine. Medzi dôvody môžeme zaradiť možnosť programovať v jazyku Java, automatickú rozšíriteľnosť, pre naše potreby je to bezplatné, je možné využívať výhody infraštruktúry Google (spoľahlivosť, výkon, bezpečnosť).
Programovací jazyk – *Java* – pre programovací jazyk na serverovej časti sme sa rozhodovali medzi jazykom *Java* a *Python,* ktoré sú na Google App Engine podporované. Keďže členovia nášho tímu sú prevažne programátori, orientovaní na jazyk Java, v tomto smere to pre nás bola jasná voľba a vybrali sme si jazyk Java.
Komunikácia s klientskou časťou – *architektúra REST, prenosový protokol HTTP, výmena dát vo formáte JSON, implementované pomocou frameworku Restlet.* Pri komunikácii medzi serverom a klientskou časťou sme sa rozhodovali medzi použitím Rest architektúry a architektúrou SOA. Pre náš systém úplne postačuje jednoduchosť architektúry Rest, preto sme sa rozhodli použiť práve túto architektúru.

Vývojové prostredie – *Eclipse* – pre toto vývojové prostredie sme sa rozhodli preto, lebo pre Google App Engine existuje Plugin, ktorý možno jednoducho naimportovať do prostredia Eclipse.

Klientska časť - iPhone

Programovací jazyk – *Objective C, Cocoa Touch*
Vývojové prostredie – *Xcode*
V tejto časti sme nemali na výber, nakoľko programovanie pre iPhone je podporované len v tomto jazyku a prostredí.

Klientska časť – webová stránka

Programovací jazyk – Java pomocou frameworku Google Web Toolkit – táto voľba bola čiastočne prispôsobením sa technológii, použitej pri implementovaní serverovej časti. Google web toolkit je navyše možné nainštalovať priamo pri inštalácii Google App Engine a je jednoducho umožnená ich komunikácia.
Vývojové prostredie – Eclipse – podobne ako pri výbere prostredia pre serverovú časť, aj tu bola voľba prostredia založená na použití pluginu pre Google Web Toolkit, ktorým je umožnené jednoduché používanie Google Web Toolkitu v prostredí Eclipse.

Opis náplne projektu

Motivácia

V dnešnej dobe, plnej technológií, je platba pomocou kreditnej karty, či internetového bankovníctva každodennou samozrejmosťou pre čoraz väčšiu masu ľudí. Je tým uľahčená platba za rôzny tovar a služby, nakoľko nie je potrebné platiť žiadnymi peniazmi. Platenie je realizované vložením karty do terminálu a vložením PIN kódu, ktorým sa zákazník autentifikuje.
Čo tak ešte aj tento proces zjednodušiť? Neexistuje niečo, čo máme pravidelne pri sebe a čo je možné použiť pre realizáciu platieb? Takýmto zariadením by mohol byť mobilný telefón, ktorý v dnešnej dobe vlastní skoro každý. Do dnešnej doby bolo vytvorených už mnoho mobilných aplikácií, ktoré zjednodušovali ľuďom viaceré služby. Takouto aplikáciou by mohla byť aj aplikácia, ktorá by umožňovala realizáciu mobilných transakcií pomocou mobilného zariadenia.
Dôvodov pre takéto riešenie je viacero. Mnoho ľudí vrátane nás by rado používalo jedno zariadenie, ktoré by zabezpečovalo čo najviac úloh. Ako vieme, do mobilného zariadenia bol zabudovaný fotoaparát, je možné pripájať sa na internet, či čítať maily. Práve takéto služby by mohli byť využité aj pri mobilnom bankovníctve pomocou mobilného zariadenia. Práve takáto aplikácia by mohla znamenať veľký prevrat nielen pri využívaní mobilných telefónov, ale znamenalo by to aj uľahčenie každodenného života pre väčšinu ľudí, ktorí často potrebujú platiť za rôzne tovary či služby.

Opis a náplň projektu

Cieľom projektu je vytvoriť transakčný systém, ktorý umožní realizáciu transakcií pomocou mobilného zariadenia, resp. mobilných zariadení. Prenos transakcií bude založený na Cloud službách, ktoré budú zabezpečené pomocou technológie Google App Engine. Prenos informácie o transakcii medzi dvoma zariadeniami bude zabezpečovaný pomocou generovania a rozpoznávania 2D kódu, ktorý bude zosnímaný a rozpoznaný.

Proces transakcie detailnejšie

V našom systéme plánujeme vytvárať 3 typy transakcií. Pôjde o nasledovné transakcie:

- používateľ si chce vyžiadať peniaze od iného používateľa.
- používateľ chce poslať peniaze inému používateľovi.
- používateľ chce zadať, akú maximálnu sumu je ochotný zaplatiť.

Používateľ si chce vyžiadať peniaze od iného používateľa
Pri tomto príbehu používateľ jedného mobilného zariadenia vyžiada od iného používateľa určitú čiastku. To bude umožnené zadaním požadovanej sumy do formulára. Informácie z tohto formulára sa odošlú na server, ktorý vráti identifikačné číslo transakcie. Toto ID sa zakóduje do 2D kódu, ktorý sa následne vygeneruje na používateľom mobilnom zariadení. Tento 2D kód potom bude možné zosnímať iným mobilným zariadením. Zosnímaný kód sa následne dekóduje a zobrazia sa informácie o transakcii. Používateľ teda vidí, komu má zaplatiť a akú sumu má zaplatiť. Následne sa rozhodne, či chce danú transakciu zrealizovať. Ak áno, potvrdí transakciu a pošle požiadavku na server, že chce zaplatiť danú čiastku. Ak je všetko v poriadku, transakcia prebehne a vykonajú sa požadované operácie.

Používateľ chce poslať peniaze inému používateľovi
Tento príbeh je opačným príbehom k predchádzajúcemu. Používateľ s mobilným zariadením chce poslať inému používateľovi peniaze. Zadá požadovanú čiastku a odošle požiadavku na server. Server opäť vráti identifikačné číslo transakcie, ktoré sa obalí v 2D kóde. Takýto kód

môže opäť zosnímať iné mobilné zariadenie a prezrieť si, o akú transakciu ide. Túto transakciu opäť potvrdí a vykonajú sa plánované operácie.

Používateľ chce zadať, akú maximálnu čiastku chce zaplatiť
Posledný príbeh je založený na predchádzajúcom príbehu, kedy chce používateľ poslať nejakú čiastku inému používateľovi. Keďže nechce zadávať presnú sumu, môže zadať maximálnu sumu, ktorú je ochotný zaplatiť a to pošle na server. Druhý používateľ môže zadať požadovanú sumu. Ak je to suma menšia alebo rovná zadanej sume, prebehne transakcia a vykonajú sa požadované operácie.

Opis priebehu transakcie

Jednotlivé transakcie sa formou procesu veľmi podobajú. Prenos medzi klientskou časťou aplikácie na mobilnom zariadení a serverom, bude zabezpečený pomocou Rest architektúry. Vykonané transakcie sa budú ukladať pomocou cloudových služieb Google App Engine na server, konkrétne na Google databázu *DataStore.* Prenos medzi mobilnými zariadeniami bude prebiehať pomocou zosnímania 2D kódu mobilným fotoaparátom, zabudovaným v zariadení.

Ďalšie časti systému

Pre beh systému a jeho bezpečnosť je potrebné, aby prenos transakcií prebiehal medzi prihlásenými používateľmi. Z toho vyplýva nutnosť, umožniť používateľom registrovať sa do systému. To chceme realizovať pomocou webovej stránky, kde bude možná registrácia. Táto stránka by mala po prihlásení sa do systému zobraziť informácie o aktuálne prihlásenom používateľovi, zobraziť transakcie, ktoré používateľ vykonal a tieto transakcie filtrovať podľa rôznych kritérií. Túto webovú stránku plánujeme implementovať pomocou Google Web Toolkit technológie. Stránka bude so serverom opäť komunikovať pomocou architektúry Rest.

Zhrnutie

Cieľom tohto tímového projektu je teda vytvoriť systém, ktorý umožní realizáciu transakcií pomocou mobilných zariadení. Plánujeme vytvoriť aplikáciu, ktorá bude dostupná zatiaľ len pre mobilné zariadenia iPhone. K transakčnému systému je potrebné vytvoriť aj webovú stránku, kde bude možné prehliadať všetky transakcie, ktoré vykonal daný používateľ. Do prvého semestra si kladieme za cieľ vytvoriť prototyp, pomocou ktorého bude možné realizovať jednu zo spomínaných transakcií. Taktiež chceme do konca prvého semestra vytvoriť webovú stránku, ktorá umožní zatiaľ len registráciu používateľa, prihlásenie používateľa a základné zobrazenie transakcií.
Do konca tímového projektu plánujeme vytvoriť systém, ktorý bude umožňovať realizovať všetky typy spomínaných transakcií. Na webovej stránke by sme chceli implementovať funkcionalitu, vďaka ktorej bude možné vykonané transakcie filtrovať na základe rôznych kritérií.
Ďalším našim cieľom je zúčastniť sa súťaže TP-cup, organizovanej fakultou. V rámci toho je potrebné vypracovať aj článok na konferenciu IIT SRC.

Šprint č. 1

Úlohou prvého šprintu bolo najmä oboznámiť sa s technológiami, ktoré budú použité pre implementáciu a vytvoriť jednoduchú aplikáciu, ktorá umožní pomocou mobilného zariadenia odoslať informáciu (reťazec) na server, ktorý túto informáciu uloží pod identifikačným číslom. Následne po zadaní tohto identifikačného čísla bude možné pomocou ďalšieho mobilného zariadenia získať túto informáciu. Bude to realizované zadaním identifikačného čísla, ktoré bolo vygenerované serverom.

V rámci tohto šprintu boli identifikované nasledujúce príbehy:

- Vloženie textovej informácie do transakčného systému
- Vybratie textovej informácie z transakčného systému

Vloženie textovej informácie do transakčného systému

Ako používateľ chcem uložiť informáciu do systému, aby ju iný používateľ mohol zo systému vybrať.

Analýza problému

Aby sa informácie mohli vkladať do transakčného systému, je potrebné riešiť nasledujúce problémy:

- Prenos informácie medzi klientom a serverom – je potrebné zvoliť prenosový protokol a navrhnúť formát pre výmenu správ medzi klientom a serverom. Navrhnutý protokol by mal mať čo najširšie využitie a mal by byť podporovaný na veľkom množstve zariadení.
- Poskytnutie údajov o vloženej informácii od servera pre klienta. To je potrebné, aby klient mohol túto informáciu ďalej poskytnúť iným klientom, ktorí si ju následne môžu vyzdvihnúť.
- Ukladanie prijatých informácií od klienta na strane servera – je potrebné navrhnúť, ako sa budú prijaté informácie ukladať.
- Odosielanie informácií na strane klienta – je potrebné vytvoriť používateľské rozhranie, ktoré umožní zadať informáciu a následne ju odoslať na server.

Návrh riešenia

Prenos informácie (HTTP protokol)
Ako prenosový protokol sme zvolili HTTP. Dôvodom je jeho veľké rozšírenie. Protokol funguje na základe požiadaviek (Request) a odpovedí (Response). Každá požiadavka i odpoveď okrem samotných dát obsahuje aj hlavičky s metadátami.

Klient vždy iniciuje požiadavku na server. Požiadavky môžu prebiehať rôznymi metódami (napr. GET, POST, DELETE). Metóda indikuje, ako by sa mal server zachovať pri spracovávaní požiadavky (napr. POST sa zvyčajne používa pri požiadavkách, ktoré menia vnútorný stav aplikácie, naopak GET zväčša slúži na získanie dát z aplikácie, teda iba na zmenu zobrazenia).

Po prijatí požiadavky ju server spracuje a odošle odpoveď. Každá odpoveď obsahuje aj stavový kód, ktorý indikuje stav spracovania požiadavky. Existuje niekoľko druhov kódov (potvrdzovacie, presmerovacie, chybové atď.).

Ako prenosový formát (čiže formát tela správ) sme v tejto fáze zvolili iba čistý text (plain text). Dôvodom je, že zatiaľ ide iba o prenos neštruktúrovaných informácií (textové reťazce).

Poskytnutie údajov o vloženej informácii od servera pre klienta
Na poskytnutie týchto údajov môžeme použiť priamo HTTP odpoveď. Po spracovaní požiadavky na vloženie informácie do systému v odpovedi vrátime údaje o tom, ako sa dá uložená informácia vyzdvihnúť.

REST
Aby mohol klient so serverom komunikovať, je potrebné presne definovať rozhranie servera. Rozhrania, ktoré umožňujú klientom komunikovať so serverom prostredníctvom siete, sa nazývajú webové služby. Vo väčšine prípadov táto komunikácia prebieha prostredníctvom protokolu HTTP. Pomocou webových služieb pracujeme s tzv. zdrojmi (Resources). Zdrojmi sú akékoľvek objekty, ktoré v systéme reprezentujeme. Každý zdroj je jedinečne identifikovaný pomocou tzv. URI (Uniform Resource Identifier).

Existujú dva prístupy k návrhu webových služieb:

- Služba definuje operácie, ktoré klient môže vykonávať (príkladom je napr. XML-RPC, príp. SOAP).
- Operácie sú definované implicitne, služba definuje, ako pristupovať k dátam (na akých adresách, akou HTTP metódou, atď.). Takýto štýl architektúry označujeme aj ako REST.

Rozhodli sme sa použiť štýl architektúry REST. Dôvodom je, že iné prístupy (napr. SOAP) sú pre naše účely zbytočne zložité.

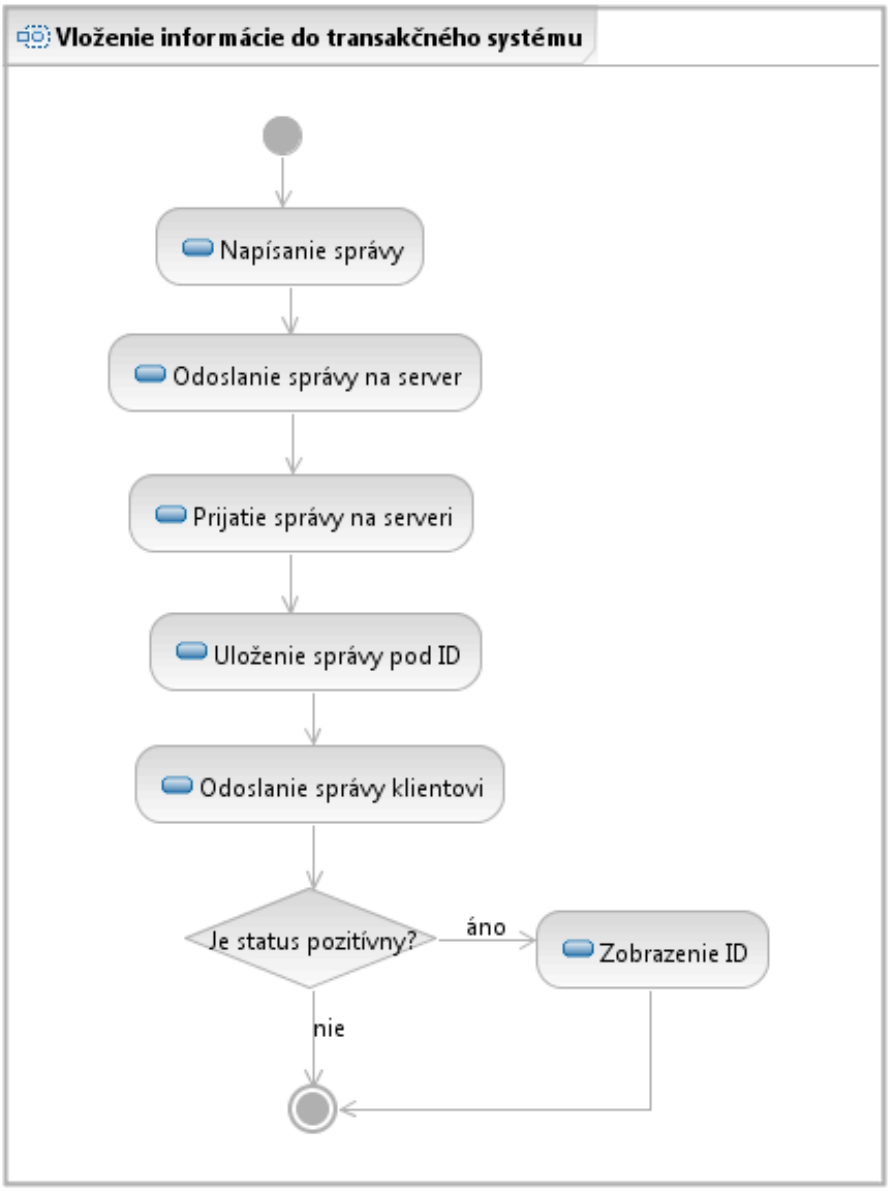
REST rozlišuje 4 základné metódy na prístup k dátam (CRUD – Create, Read, Update, Delete), ktoré zodpovedajú HTTP metódam (POST, GET, PUT, DELETE). V tomto príbehu použijeme iba metódu Create. Aby sme klientovi mohli potvrdiť úspešné uloženie informácie na serveri, v odpovedi použijeme HTTP kód určený pre tento prípad, 201 Created.

Podrobnejší opis komunikácie:
Požiadavka
HTTP Metóda: POST
URI: /strings/ (Pozn.: Toto URI je relatívne vzhľadom na server, kde bude REST rozhranie umiestnené.)
Telo: Reťazec, ktorý sa má uložiť

Odpoveď
HTTP kód pri úspechu: 201 Created
Hlavičky: Location: URI uloženého reťazca
Telo: Informácia o identifikačnom čísle, pod ktorým bol reťazec uložený

Ukladanie prijatých informácií od klienta na strane servera
Keďže cieľom prvého šprintu je oboznámenie sa s technológiou a vyskúšanie komunikácie medzi klientom a serverom, ukladanie informácií bude riešené iba pomocou dátovej štruktúry v pamäti servera. Problémom je, že údaje sa stratia po reštartovaní servera alebo aplikácie, teda nie sú perzistentné. Keďže tento jav nenastáva často, vzhľadom na naše ciele v prvom šprinte je to dostatočné riešenie.

Odosielanie informácií na strane klienta
Keď bude používateľ chcieť odoslať správu na server, napíše najskôr reťazec. Tento reťazec sa pomocou HTTP protokolu pošle na určenú URL adresu, na ktorej je server pripravený spracovať HTTP požiadavku. Po prijatí HTTP požiadavky vytiahne server správu z jej obsahu a uloží ju pod nejakým náhodne vygenerovaným číslom. Toto číslo sa potom odošle v odpovedi naspäť klientovi. Keď klient príjme odpoveď, tak skontroluje, či prišiel pozitívny potvrdzovací kód (či sa úspešne odoslala informácia). Ak bola transakcia v poriadku, tak zobrazí číslo, pod ktorým je na serveri uložená informácia. Graficky je tento postup zobrazený na obrázku Obrázok 1 **Proces odoslania informácie**Obrázok 1.



Obrázok 1 Proces odoslania informácie.

Opis implementácie

REST
Pre implementáciu webovej služby sme použili framework Restlet. Za zdroje sa považujú reťazce (String), sú definované rozhraním StringResource. O implementáciu sa stará trieda

InMemoryStringServerResource. Informácie sa ukladajú do objektu HashMap v pamäti, ktorá sa vytvorí pri štarte aplikácie. Operácia vkladania informácií je realizovaná metódou @Post store (String string). Výhodou frameworku Restlet je, že môžeme pomocou Java anotácii určiť, ktorá HTTP metóda je povolená pri volaní operácie. V tomto prípade sme určili, že je povolená iba metóda POST.

Aby sme vedeli definovať, ku ktorým zdrojom prislúchajú jednotlivé URI adresy, pri štarte aplikácie musíme nakonfigurovať tzv. smerovač (Router). Ten funguje na princípe regulárnych výrazov a umožní všetky požiadavky, ktoré zodpovedajú výrazu smerovať na určený zdroj. Pre splnenie tohto používateľského príbehu stačilo do smerovača pridať iba jednu cestu (route).

Klientska časť iPhone
Klientska časť bude bežať na mobilnom zariadení iPhone, preto je táto časť implementovaná v programovacom jazyku Objective C vo framework-u Cocoa Touch.

Informácia je z klientskej časti posielaná pomocou HTTP Request metódy POST. Na toto je použitá trieda z Objective-C `NSMutableURLRequest`, v ktorej sa metóda nastaví na POST, adresa kam sa posielajú tieto správy sa nastaví na "http://branovaprva.appspot.com/rest/strings/" a ako telo sa zadefinuje reťazec, ktorý je získaný z text boxu, ktorý je na obrazovke.

Po odoslaní požiadavky sa zo servera vráti odpoveď, z ktorej sa získajú údaje. Ak je status kód v rozmedzí od 200 do 300, znamená to, že správa bola doručená v poriadku. Preto sa pomocou triedy `UIAlertView` vygeneruje správa obsahujúca číslo ID, ktoré identifikuje miesto, kde je na serveri uložená správa, ktorá tam bola odoslaná.

Testovanie

Na otestovanie bolo potrebné implementovať aj druhý príbeh – Vybratie textovej informácie z transakčného systému. Pre opis testovanie pozri opis tohto príbehu.

Vybratie textovej informácie z transakčného systému

Ako používateľ chcem vybrať informáciu zo servera, aby som vedel, čo mi tam predchádzajúci používateľ zanechal.

Analýza problému

Aby sme mohli informácie z transakčného systému vyberať, potrebujeme riešiť niekoľko problémov:

- Prenos informácie medzi klientom a serverom – je bližšie opísaný v časti Vloženie informácie do transakčného systému.
- Vyzdvihnutie informácie zo servera prostredníctvom jej identifikátora. Môžu nastať aj chybové stavy, keď sa informácia so zadaným identifikátorom na serveri nenachádza.
- Zobrazenie získanej informácie na klientovi.

Návrh riešenia

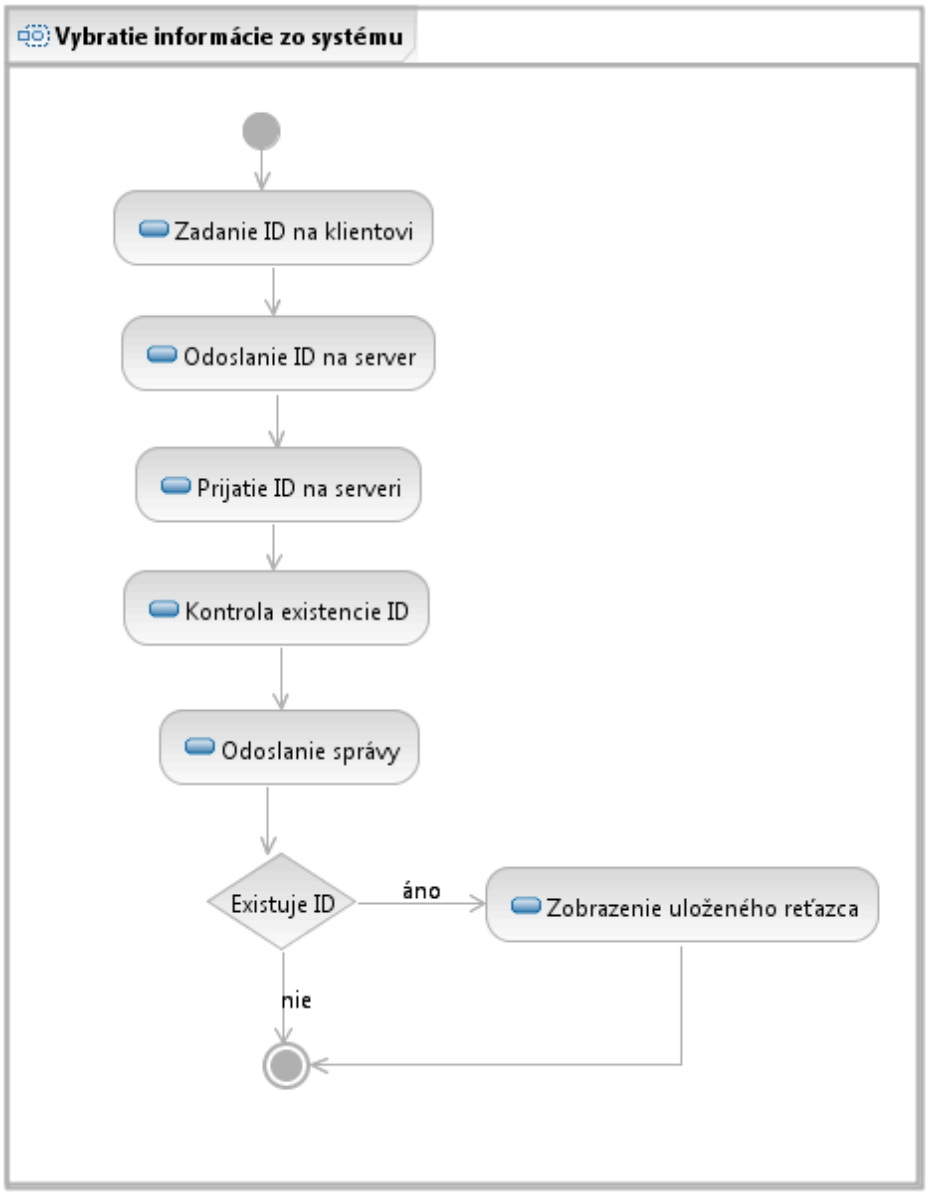
REST
Ako indikáciu neexistujúceho zdroja použijeme HTTP kód 404 Not Found. Podrobný opis komunikácie je nasledovný:

Požiadavka
HTTP Metóda: GET
URI: /strings/<identifikačné_číslo>
Telo: prázdne

Odpoveď
Úspešné vykonanie požiadavky
HTTP kód: 200 OK
Telo: Uložený reťazec

Neexistujúci reťazec
HTTP kód: 404 Not Found
Telo: prázdne

Priebeh vybratia informácie je na obrázku (Obrázok 2).



Obrázok 2 Vybratie informácie z transakčného systému.

Opis implementácie

REST
Pri implementácii sme rozšírili triedu InMemoryStringServerResource o ďalšiu metódu @Get retrieve(). Okrem toho bolo potrebné rozšíriť smerovač o ďalšiu cestu, s jedným variabilným parametrom, ktorým je identifikátor reťazca.

Klientska časť iPhone
Na klientovi bola rozšírená implementácia o rozhranie, pomocou ktorého bude možné zadať ID reťazca a získaný reťazec zo servera zobraziť.

Testovanie

Navrhli sme tento testovací scenár:

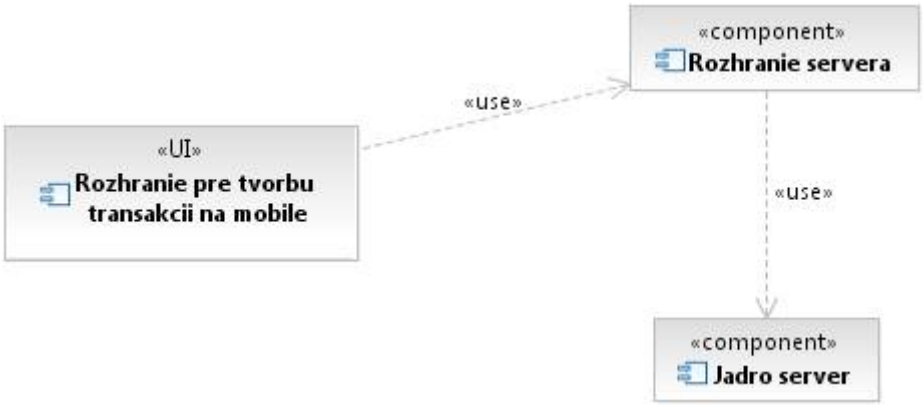
1. Na klientskom zariadení sa odošle požiadavka na vloženie vymysleného reťazca do systému. Očakávaným výstupom je odpoveď s kódom, reprezentujúcim úspešné vloženie reťazca a identifikátor reťazca v tele odpovede. Zároveň je potrebné zapamätať si tento identifikátor.
2. Na klientskom zariadení sa odošle požiadavka na vybratie reťazca so zapamätaným identifikátorom zo systému. Očakávaným výstupom je odpoveď, ktorá v tele obsahuje reťazec, zhodný s vymysleným reťazcom z kroku 1.

Výstup v oboch krokoch bol podľa očakávania.

Zhrnutie šprintu č. 1

Cieľom tohto šprintu bolo vytvoriť takzvanú „HelloWorld“ transakčnú aplikáciu, kde bude možné odoslať správu na server a potom ju získať naspäť. Na základe vyššie spomínaných informácií sa podarilo túto úlohu tímu splniť. Bola vytvorená aplikácia, ktorá na klientovi – mobilnom zariadení umožní zadať informáciu vo formáte reťazca. Túto informáciu pošle na server, kde sa uloží a je ju možné pomocou identifikačného čísla získať ďalším mobilným zariadením iPhone.

Po 1. šprinte sa nám podarilo identifikovať nasledovnú architektúru systému (Obrázok 3).



Obrázok 3 Architektúra systému po 1. šprinte.

V rámci architektúry systému boli identifikované nasledujúce komponenty:

- ***Rozhranie pre tvorbu transakcií na mobile*** – ide o komponent, predstavujúci používateľské GUI rozhranie pre vytvorenie reťazca, ktorý budeme posielať a taktiež

pre získanie tohto reťazca zo servera. Tento komponent používa pre svoje úlohy komponent *Rozhranie servera.*

- ***Rozhranie servera*** **–** tento komponent predstavuje REST API, vďaka ktorému je umožnená komunikácia medzi klientom a serverom. Tento komponent získava Http požiadavku z klientskej časti systému a posiela Http odpoveď späť klientskej časti. Pre ukladanie dát používa komponent *Jadro server.*
- ***Jadro server*** **–** tento komponent umožňuje ukladanie získaných dát na serveri a ich spätné sprístupňovanie.

Šprint č. 2

Cieľom druhého šprintu je posunúť prvotnú aplikáciu, vytvorenú v prvom šprinte ďalej. Potrebné je začať používať maticové kódy, v ktorých bude uložená informácia o uložení objektu na serveri. Úlohou teda bude vygenerovať maticový kód jedným zariadením a prijať druhým zariadením. Na strane servera je potrebné začať používať JSON objekty a umožniť registráciu používateľov do systému pomocou nich.
V rámci tohto šprintu boli identifikované nasledovné príbehy:

- Registrácia používateľov na webovej stránke
- Vloženie informácie do transakčného systému a zobrazenie 2D kódu
- Vloženie komplexnej informácie do transakčného systému
- Vybratie komplexnej informácie z transakčného systému

Registrácia používateľov

Ako používateľ sa chcem zaregistrovať, aby som mal prístup k funkcionalite pre registrovaných používateľov.

Analýza problému

Identifikovali sme nasledujúce problémy:

1. Je potrebné navrhnúť, ako sa dáta budú na serveri ukladať, s dôrazom na nasledujúci bod.
2. Keďže medzi registračné údaje patria aj heslá, je potrebné zabezpečiť, aby sa na serveri neukladali ako čistý text (zaistenie aspoň minimálnej bezpečnosti).
3. Pri registrácii môže nastať niekoľko chybových stavov – zadaný používateľ je už registrovaný, alebo niektoré údaje nespĺňajú požadovaný formát. Tieto situácie je potrebné ošetriť.
4. Je veľmi pravdepodobné, že spôsob registrácie a požadované údaje sa budú v budúcnosti meniť. Preto by bolo vhodné oddeliť registráciu od mobilného klienta a registráciu realizovať prostredníctvom webovej stránky (vytvoriť a upravovať ju sa nám v tejto fáze javí vhodnejšie ako neustále upravovať používateľské rozhranie na mobilnom klientovi).

Návrh riešenia

REST
REST rozhranie zohľadňuje fakt, že informácie o registrácii sú štruktúrované (pozri príbeh *Vkladanie štruktúrovanej informácie*).

Podrobný opis rozhrania:
Požiadavka (Request)
HTTP Metóda: POST
URI: /user
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “name”: “meno_pouzivatela”,
  “password”: “zvolene_heslo_pouzivatela”,
  “mail”: “user@server.tld”
}
```

Odpoveď (Response)
Úspešné vykonanie požiadavky
HTTP kód: 201 Created
Telo: prázdne

Používateľ už existuje
HTTP kód: 409 Conflict
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt (pozri vyššie) so správou o chybe

Niektorý z vstupných parametrov je chybný
HTTP kód: 400 Bad Request
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Web rozhranie pre registráciu
Pomocou tohto webového rozhrania je potrebné získať od používateľa všetky informácie, potrebné k registrácii. Na to budú slúžiť klasické formulárové textové polia, do ktorých sa budú jednotlivé údaje vpisovať. Tieto údaje je potom potrebné z formulára získať a vytvoriť JSON objekt, ktorý bude posielaný na server na REST-ové rozhranie, kde bude ihneď vyhodnotený. To znamená, že sa zistia všetky potrebné detaily, prípadne sa odhalia konflikty, pre ktoré nemôže byť takýto používateľ registrovaný.
Zo servera sa vráti opäť JSON objekt, ktorý bude niesť informácie o možnosti registrovať používateľa. V časti REST je možné vidieť, aké kódy môžu byť vrátené. V závislosti od toho bude používateľ vytvorený alebo bude známa chyba, ktorá nastala.

Jadro servera
Na serverovej časti je potrebné navrhnúť ako vhodne ukladať používateľov. Pre ukladanie bude použitá databáza, ktorá beží na google serveri – Google DataStore. Tým bude tiež zabezpečená perzistencia dát. S databázou sa nepracuje priamo relačným spôsobom, ale pomocou technológie „Java Data Objects“ objektovým prístupom. Entita používateľa, ktorá sa bude ukladať, bude mať atribúty, potrebné k úspešnému neskoršiemu prihláseniu sa do systému. Minimálne sú potrebné login, heslo a email používateľa.
V rámci registrácie je potrebné vytvoriť aj entitu email, ktorá bude obsahovať validátor emailu. Inštancie tejto triedy bude ukladaná pri každom registrovanom používateľovi.
Jadro servera spracováva používateľov, ukladá ich do perzistentnej pamäte, načítava ich a dovoľuje ich odstránenie a zmenu. V rámci jadra sa implementovala aj funkcionalita spoločná pre klienta a server.

Opis implementácie

REST
Implementácia je veľmi podobná prvému šprintu. Zdroj je definovaný pomocou rozhrania UserResource, ktoré implementuje trieda ServerUserResource. Metóda registrácia používateľa je implementovaná funkciou @Post create(RegistrationInfoDTO info). Objekt triedy RegistrationInfoDTO je automaticky deserializovaný zo vstupného JSON formátu.

Web rozhranie pre registráciu
Implementácia tejto časti bude realizovaná pomocou Google Web Toolkit. Pomocou neho bude vytvorený jednoduchý formulár, do ktorého bude môcť používateľ zadať všetky potrebné informácie. Následne bude implementované rozhranie ClientProxy, vďaka ktorému

bude pomocou funkcie @Post create(RegistrationInfoDTO info) odoslaný JSON objekt na server. Po získaní odozvy zo servera bude zobrazená informácia o úspechu či neúspechu celého procesu.

Jadro servera
Objekty v balíku **sk.fiit.mpayserver.shared** sú použiteľné v časti klienta ako aj v časti servera. Nachádza sa tam rozhranie, určujúce čo všetko musí zadať používateľ pri registrácii (heslo je oddelené kvôli bezpečnosti). Objekty v balíku **sk.fiit.mpayserver.core** je možné použiť iba v časti servera. Nachádza sa tam trieda **Core** ktorá poskytuje základné služby pre prácu jadra a to generovanie bezpečných náhodných čísel a sprístupňovanie inštancie potrebnej pre perzistenciu objektov.
V balíku sa ďalej nachádza trieda **UserManager**, ktorá vykonáva samotné požiadavky na prácu s používateľmi. Používatelia sa ukladajú ako inštancie objektu **PersistentUser**. Následne je možné získať uloženého používateľa vo forme inštancie triedy **EditableUser**, ktorá sa odkazuje na perzistentnú verziu používateľa. V úložisku sa nenachádza uložené heslo ale iba jeho „MessageDigest“ podľa ktorého sa overuje správnosť hesla, čo zvyšuje bezpečnosť systému. Pre získanie používateľa je potrebné poznať jeho heslo a mail. Z toho vyplýva, že získať inštanciu používateľa je možné iba s jeho pričinením. Triedy na získanie používateľa bez jeho hesla zatiaľ neboli implementované, a budú implementované neskôr pri vytváraní administrátorskej funkcionality jadra. Pomocou **UserManager** objektu je možné zistiť či daný používateľ už existuje (na základe jeho mailovej adresy). Pri registrácii je potrebné overiť mail používateľa, táto funkcionalita zatiaľ nebola implementovaná.
Server aj klient využívajú triedu **Mail** ktorá obaľuje triedu **String** a pridáva k nej podmienku validity mailovej adresy. Takto bola zaručená validita mailu v rámci systému. Taktiež je možné pomocou tejto triedy overovať validitu textového reťazca v prípade, že nechceme vytvoriť inštanciu objektu **Mail**.

Testovanie

Na otestovanie tejto funkcionality sme navrhli tieto testovacie scenáre:

- Požiadavka na registráciu používateľa s vymyslenými údajmi, pričom mail je formálne správny a meno aj heslo sú vyplnené. Očakávaným výstupom je odpoveď s kódom reprezentujúcim úspešnú registráciu.
- Požiadavka na registráciu rovnakého používateľa ako v prípade 1. Očakávaným výstupom je zamietnutie registrácie z dôvodu existencie používateľa, odpoveď s chybovým kódom a správa o chybe v tele odpovede.
- Požiadavka na registráciu, kde mail nie je formálne správny. Očakávaným výstupom je odpoveď s chybovým kódom a správa o chybe v tele správy.
- Požiadavka na registráciu, kde chýba niektorý z parametrov {meno, heslo, mail}. Očakávaným výstupom je odpoveď s chybovým kódom a správa o chybe v tele správy.

Výstupy vo všetkých krokoch zodpovedali očakávaniam.

Vloženie informácie do transakčného systému a zobrazenie 2D kódu

Ako používateľ chcem do systému vložiť informáciu. iPhone mi po vložení informácie zobrazí 2D kód, v ktorom je uložené ID, pod ktorým je informácia uložená na serveri.

Analýza problému

Tento príbeh pridáva funkcionalitu vygenerovania 2D maticového kódu ku príbehu "Vloženie informácie do transakčného systému" spracovávaného v šprinte 1. Pre generovanie 2D kódu sa bude využívať knižnica s touto funkcionalitou, ktorá bude získaná.

Návrh riešenia

Návrh sa oproti predchádzajúcemu príbehu zmení iba tak, že keď používateľ pošle správu na server a príde mu odpoveď, tak okrem čísla, pod ktorým sa uchovala informácia na serveri sa zobrazí aj 2D kód, v ktorom bude toto číslo zakódované.

Opis implementácie

Na implementáciu sme použili knižnicu *Softmatic Barcode Generator*. Použitím funkcií z tejto knižnice sme zakódovali informáciu, ktorá nám prišla zo servera a nastavili sme ju na objekt UIImage, ktorý predstavuje obrázok, a ktorý sa zobrazí na displeji používateľovi.

Testovanie

Testovanie sa z časových dôvodov nestihlo vykonať špecializovanými prostriedkami. V budúcnosti sa plánuje nájsť voľne prístupný dekodér 2D kódov, uložiť vygenerovaný 2D kód do súboru a otestovať správnosť zakódovania.

Vloženie komplexnej informácie do transakčného systému

Vybratie komplexnej informácie z transakčného systému

Cieľom v týchto príbehoch je rozšíriť príbeh z prvého šprintu (*Vybratie/Vloženie informácie z/do transakčného systému)* o možnosť vkladať a vyberať štruktúrované informácie. Je to najmä príprava pre ďalší šprint, aby si vývojári serverovej i klientskej časti vyskúšali prácu so štruktúrovanými dátami.

Analýza problému

- V prvom šprinte sa do systému ukladali iba jednoduché texty, preto treba systém rozšíriť.
- Je potrebné určiť formát, v ktorom sa budú dáta prenášať medzi klientom a serverom.
- Je potrebné navrhnúť, ako sa budú informácie na serveri ukladať.
- Je potrebné navrhnúť systém na serializáciu štruktúrovaných dát.

Návrh riešenia

Formát štruktúrovaných informácií
Keďže v systéme sa budú ďalej prenášať štruktúrované informácie, je potrebné navrhnúť, v akom formáte sa budú prenášať. Zvolili sme formát JSON, keďže je jednoduchý, kompaktný a má dobrú podporu na strane klientov (je podmnožinou jazyka Javascript, ktorý je podporovaný v každom webovom prehliadači). Druhou alternatívou bol formát XML. Jeho výhody ale neboli tak významné, aby sme ho uprednostnili pred formátom JSON.

S touto zmenou súvisí aj formát chybových správ. Ten by mal obsahovať indikáciu, že došlo k chybe a tiež chybovú správu. Chybový objekt teda bude nasledovný:

```
{
  “error”: true,
  “errorMessage”: “Správa o chybe.”
}
```

V budúcnosti bude pravdepodobne potrebné pridať aj chybový kód, čo s navrhnutým formátom bude pomerne jednoduché.

REST
Rozhodli sme sa, že v tomto príbehu pri vkladaní informácie do systému zatiaľ nepotrebujeme iné informácie, ako je vkladaný text. Preto sme nepridali ďalšie polia, iba sme pole s textom „obalili“ do JSON formátu. Pri vyberaní informácie zo systému sme pridali pole, ktoré nesie údaj o čase, kedy bola informácia vložená do systému.

Podrobný opis rozhrania:
Požiadavka
Vloženie štruktúrovanej informácie do systému
HTTP Metóda: POST
URI: /structured
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “text”: “textová informácia”
}
```

Odpoveď
Úspešné vykonanie požiadavky
HTTP kód: 201 Created
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI uloženej informácie
Telo: JSON objekt vo formáte:

```
{
  “id”: “identifikačné číslo, pod ktorým bola informácia uložená”,
  “uri”: “URI uloženej informácie”
}
```

Požiadavka
Vybratie štruktúrovanej informácie zo systému
HTTP Metóda: GET
URI: /structured/<identifikačné_číslo>
Telo: prázdne

Odpoveď
Úspešné vykonanie požiadavky
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “text”: “uložená textová informácia”,
  “time”: “čas vloženia informácie do systému”
}
```

Informácia so zadaným identifikačným číslom na serveri neexistuje
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt (pozri vyššie) so správou o chybe

Spôsob ukladania informácií na serveri
Keďže cieľom týchto príbehov je vyskúšať spracovanie štruktúrovaných informácií na strane servera i klienta, spôsob ukladania bude rovnaký ako v prvom šprinte – do dátovej štruktúry v pamäti servera. Problémy tohto riešenia sú rovnaké ako v prvom šprinte, výhodou je veľmi rýchla implementácia.

Klientska časť iPhone
Keď sa bude posielať informácia na server, z klientskej časti sa odošle JSON objekt, ktorý bude obsahovať jedno pole, tak ako je to uvedené v rozhraní. Toto pole zadá používateľ a bude predstavovať správu, ktorá sa odošle na server. Naspäť príde odpoveď, ktorá bude tiež vo forme JSON objektu. Odpoveď bude obsahovať viacero polí (podľa rozhrania), ale aplikácia z toho použije iba pole "id", podľa ktorého bude možné nájsť uloženú informáciu na serveri.
Keď bude chcieť používateľ získať správu aj s časom vloženia tejto správy na server, zadá ID, to sa použije v adrese URL, na ktorú pôjde požiadavka na vybratie správy. Naspäť na klientsku aplikáciu znovu príde komplexná informácia s dvoma poľami (čas a správa), vytiahne sa z nej iba správa a tá sa zobrazí používateľovi.

Opis implementácie

REST
Keďže dáta sa prenášajú vo formáte JSON a v kóde sa pracuje priamo s Java objektmi (kvôli elegancii, čistote kódu, typovosti atď. nepracujeme priamo s JSON objektmi), je potrebné zabezpečiť deserializáciu z formátu JSON. Rozhodli sme sa použiť knižnicu Jackson, ktorá má podporu priamo vo frameworku Restlet a tým umožňuje deserializovať transparentne. Konfigurácia spočíva iba v pridaní knižnice do classpath.
Okrem toho bolo potrebné zaistiť aj to, aby boli chybové stavy tiež reprezentované vo formáte JSON. Momentálne sa do JSON formátu prevádzajú chyby, ktorých výskyt môže používateľ ovplyvniť (tzv. client errors, teda HTTP kódy 4xx). Chyby z kategórie 5xx (vnútorné chyby servera) sa zatiaľ takto neprevádzajú. K tomuto rozhodnutiu sa dospelo počas implementácie a bude ho potrebné ešte prehodnotiť. Toto sa vo frameworku Restlet realizuje pomocou tzv. StatusService. Ich úlohou je vrátiť pre daný chybový stav aplikácie vhodnú reprezentáciu. Ako reprezentácia chybového stavu sa používa objekt triedy JsonErrorRepresentation.

Konkrétne je na serveri štruktúrovaná informácia implementovaná pomocou triedy StructuredInfoServerResource. Dôležitými metódami sú @Get StructuredInfo retrieve() a @Post void store(StructuredInfo info).

Klientska časť iPhone
Na prácu s JSON objektmi sme použili triedu NSDictionary. Premennej tejto triedy sme priradili objekt označený ako "text" a k nemu sme priradili hodnotu z textového poľa, ktoré vyplnil používateľ. Toto sme odoslali podobne ako v predchádzajúcej časti, kedy sme posielali iba obyčajný reťazec s tým, že boli zmenené atribúty HTTP požiadavky (pozri časť návrh REST, podrobný opis rozhrania) a JSON objekt sme pomocou triedy JSONRepresentation pretransformovali do reťazca, ktorý sa mohol nastaviť ako telo požiadavky.

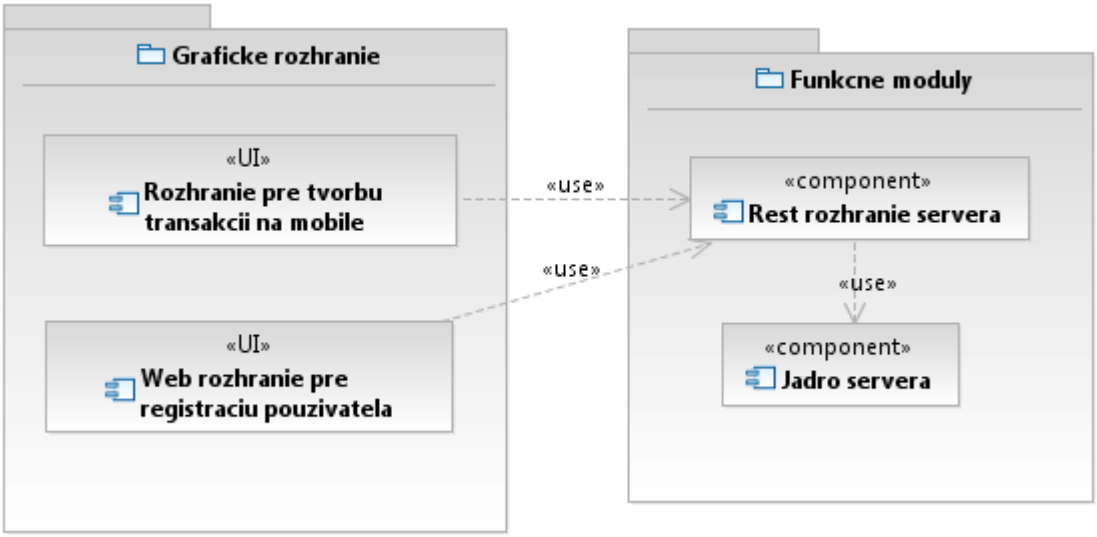
Podobne sa vykonala implementácia aj v opačnom smere, kedy sme oproti predchádzajúcemu príbehu (vybratie textovej informácie) nenastavili ako správu na zobrazenie jednoducho obsah odpovede, ale museli sme to pretransformovať pomocou JSONValue na JSON objekt, z ktorého sme pomocou NSDictionary vytiahli správu na zobrazenie používateľovi.

Testovanie

Testovacie scenáre sú rovnaké ako v prvom šprinte, preto ich na tomto mieste neuvádzame.

Zhrnutie šprintu č. 2

Cieľom tohto šprintu bolo obohatiť doteraz vytvorenú aplikáciu o používanie JSON objektov, generovania 2D kódu a odoslanie komplexnej informácie na server. Taktiež bolo potrebné vytvoriť web stránku na registráciu používateľov. Úloha, ktorej cieľom bolo vytvoriť web stránku, sa nepodarila v tomto šprinte dokončiť. Problémom bolo neodhadnutie náročnosti komunikácie medzi webovou stránkou, vytvorenou pomocou GWT
Po tomto šprinte architektúra systému vyzerá nasledovne (Obrázok 4).



Obrázok 4 Architektúra systému po 2. šprinte.

V rámci architektúry systému boli identifikované nasledujúce komponenty:

- ***Rozhranie pre tvorbu transakcií na mobile*** – ide o komponent, predstavujúci používateľské GUI rozhranie pre vytvorenie reťazca, ktorý budeme posielať a taktiež pre získanie tohto reťazca zo servera. Tento komponent používa pre svoje úlohy komponent *Rozhranie servera.* V tomto šprinte bol modul obohatený o generovanie 2D kódu a posielanie štruktúrovaných dát v podobe JSON objektov medzi klientom a serverom.
- ***Web rozhranie pre registráciu používateľa*** – tento komponent vznikol v tomto šprinte. Pomocou neho je možné do systému zaregistrovať nového používateľa. Komponent je predstavovaný web stránkou, ktorá túto funkcionalitu umožňuje. Tento komponent ale po tomto šprinte nebol doimplementovaný. Dôvody boli uvedené v zhodnotení šprintu.
- ***Rest rozhranie servera*** – tento komponent predstavuje REST API, vďaka ktorému je umožnená komunikácia medzi klientom a serverom. Tento komponent získava Http Request z klientskej časti systému a posiela Http Response späť klientskej časti. Pre ukladanie dát používa komponent *Jadro server.* Podobne ako v komponente *Rozhranie pre tvorbu transakcií na mobile,* aj tu je zmena v používaní JSON objektov. Taktiež bol tento komponent obohatený o REST rozhranie pre komunikáciu s webovou stránkou, na ktorej sa bude registrovať používateľ.

- ***Jadro server*** **–** tento komponent umožňuje ukladanie získaných dát na serveri a ich spätné sprístupňovanie. Komponent bol v tomto šprinte rozšírený o pridávanie používateľov do databázy.

Šprint č. 3

Úlohou 3. šprintu je nadviazať na predchádzajúce šprinty a ďalej zlepšovať funkcionalitu jednotlivých modulov systému. V rámci iPhone klienta je úlohou tohto šprintu vybrať informáciu zo servera na základe zosnímaného 2D kódu, čo zahŕňa zosnímanie 2D kódu a jeho dekódovanie. Taktiež je potrebné zahrnúť pre iPhone klienta možnosť autentifikácie používateľa do systému. Pre serverovú časť sú úlohy určené nasledovne: je potrebné dokončiť úlohu z predchádzajúceho šprintu, v rámci ktorého sa nepodarilo dokončiť implementáciu web stránky pre registráciu používateľa. Po registrácii používateľa je potrebné zahrnúť aj možnosť prihlásiť sa do systému z webovej aplikácie. Úlohou je implementovať toto prihlasovanie pomocou aspoň minimálnych bezpečnostných opatrení. Toto prihlasovanie bude možné vykonať aj na spomínanom iPhone.
V rámci tohto šprintu boli identifikované nasledovné príbehy:

1. Registrácia používateľa
2. Vybratie informácie z transakčného systému na základe 2D kódu
3. Prihlásenie sa do transakčného systému

Registrácia používateľa

Web stránka na registráciu používateľa

Táto úloha nebola včas dokončená pre 2. šprint. Jej cieľ je opísaný v druhom šprinte. V tomto šprinte bola úspešne ukončená. Problémy boli najmä s komunikáciou medzi Google Web Toolkit a frameworkom Restlet. Po odstránení všetkých problémov sa podarilo implementáciu tejto časti úspešne dokončiť.
Celý proces tvorby web stránky pre registráciu (analýza, návrh, implementácia, testovanie) je uvedený v 2. šprinte.

Vybratie informácie z transakčného systému na základe 2D kódu

Používateľ chce na základe získaného 2D kódu získať informáciu, ktorá tam bola uložená.

Analýza problému

Analýza snímania kamerou na iPhone
Na iPhone klientskej aplikácii je potrebné vybrať informáciu zo serverového úložiska na základe 2D kódu. Ako vstup pre 2D kód môže slúžiť kamera telefónu alebo už uložené fotky v galérii telefónu. DM kód obsahuje zakódované ID, pod ktorým je informácia na serveri uložená.

Analýza dekódovania 2D kódu
Kód, ktorý sa zosnímal na kamere (prípadne bol vytiahnutý z galérie ako obrázok) je potrebné dekódovať - v tomto prípade vytiahnuť z neho informáciu, ktorá nám určuje pod akým ID je uložená správa na serveri. Opäť ako aj pri generovaní 2D kódu je potrebné použiť knižnicu, ktorá poskytuje danú funkcionalitu.

Návrh

Návrh snímania kamerou na iPhone
Pre zosnímanie 2D kódu kamerou použijeme rozhranie, ktoré poskytuje iPhone framework. Týmto zabezpečíme, že sa nedostaneme do konfliktu s jeho licenčnými obmedzeniami, ktoré by mohli nastať, keby sme ku kamere pristupovali priamo.

Po zosnímaní kódu, dekódujeme jeho obsah a na základe ID, ktoré bolo v ňom zakódované vyberieme informáciu uloženú na serveri.

Návrh dekódovania 2D kódu
Na dekódovanie 2D kódu sa použije knižnica PtBarcodeDec[1], ktorá je bezplatná, voľne stiahnuteľná a poskytuje dekódovanie nami zvoleného "DataMatrix" kódu. Okrem nej bola ešte možnosť inej knižnice - RedLaser[2]. Táto knižnica by takisto spĺňala všetky naše požiadavky, dokonca by poskytovala aj rozpoznanie 2D kódu počas snímania na kameru, ale nie je bezplatná.

V aplikácii bude pridaná funkcia, ktorá dostane na vstupe obrázok, použitím knižnice PtBarcodeDec ho dekóduje a vráti číslo ID, ktoré bolo zakódované v obrázku.

Implementácia

Implementácia snímania kamerou na iPhone
Snímanie bolo vykonané navrhnutou triedou `UIImagePickerController.`

```
59 - (void)viewDidLoad {
60     [super viewDidLoad];
61         UIImagePickerController *picker = [[UIImagePickerController alloc] init];
62         picker.sourceType = UIImagePickerControllerSourceTypeCamera;
63         picker.delegate = self;
64         picker.allowsEditing = false;
65         [self presentModalViewController:picker animated:YES];
66 }
```

Rozhranie nám formou delegáta vráti zosnímaný obraz. Ten následne spracujeme prostredníctvom rozhrania pre prácu s DM kódom.

```
69 - (void) imagePickerController:(UIImagePickerController *)picker
              didFinishPickingMediaWithInfo:(NSDictionary *)info
70 {
71         UIImage *image = [info objectForKey:@"UIImagePickerControllerOriginalImage"];
72
73         UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Decoded"
               message:[DMFactory decodeDM:image.CGImage] delegate:self
               cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
74         [alert show];
75         [alert release];
76         [picker release];
77 }
```

Informáciu uloženú na serveri vyberieme zavolaním rozhrania `MPTransport`.

```
56         NSString *data = [MPTransport getDataByID:getIdField.text andAuth:authHeader];
```

[1] http://www.partitek.com/PtDMDecode.htm
[2] http://www.redlaser.com/

Implementácia dekódovania 2D kódu

Pre dekódovanie 2D kódu bola implementovaná funkcia decodeDM, ktorá je definovaná nasledovne:

```
+ (NSString*) decodeDM : (CGImageRef) brushImage
```

Ako vstupný parameter má bitmapový obrázok reprezentovaný objektom triedy `CGImageRef`. Na výstupe vracia objekt typu `NSString`, ktorý obsahuje ID transakcie, ktoré bolo zakódované v 2D kóde. Na dekódovanie bola použitá najmä funkcia `PtDMDecode` z knižnice PtBarcodeDec, ktorá je definovaná nasledovne:

```
int PtDMDecode(PTIMAGE* pImage, PTDECODEPARA* Para,
PTTOTALBARCODEINFO* pInfo)
```

Pred jej použitím boli inicializované všetky tri premenné daných štruktúr. Prvé dve museli byť inicializované ručne, na poslednú bolo možné použiť funkciu `PtDMDecodeInit` z knižnice. Zavolaním funkcie `PtDMDecode` sa nastaví premenná `pInfo`, ktorá obsahuje informácie obsiahnuté v 2D kóde. Ak táto premenná (objekt štruktúry `PTTOTALBARCODEINFO`) má nastavený počet identifikovaných 2D kódov väčší ako 0, nastaví sa do premennej `gCodeInfo` typu `NSString` obsah dátovej položky z `pInfo` vo formáte `UTF8String`. Funkcia na konci vráti práve premennú `gCodeInfo`. Ak neboli identifikované žiadne 2D kódy, funkcia vráti `nil`.

Testovanie

Dekódovanie 2D kódu bolo otestované zároveň aj s jeho generovaním. Najskôr sa odoslala na server správa, vrátilo sa ID, z ktorého bol vygenerovaný 2D kód. Toto bolo vykonané na notebooku. Potom na mobilnom zariadení iPhone bol zoskenovaný 2D kód z obrazovky notebooku a dekódovaný. Kód sa podarilo úspešne identifikovať. Nastal iba jeden problém v tom, že knižnica pri dekódovaní zamení ľubovoľný znak za "$". Je to zrejme spôsobené tým, že používame demo verziu knižnice. Podarilo sa nám to ale ošetriť tak, že sme do 2D kódu zakódovali dva krát ten istý reťazec, oddelený oddeľovacím znakom. Knižnica zamieňala iba jeden znak v reťazci, takže sme tým docielili, že jeden reťazec ID bude kompletný. Tento nedostatok sa odstráni, keď prejdeme na plnú verziu knižnice.

Prihlásenie sa do transakčného systému

Používateľ sa chce prihlásiť do transakčného systému na klientskej časti - iPhone ale aj na webovej stránke. Na strane iPhone klienta chce realizovať transakcie, na strane webovej stránky chce prezerať vykonané transakcie.

Analýza

Na implementáciu prihlasovania je možné použiť niekoľko prístupov. Ich výhody a nevýhody sú opísané nižšie.

HTTP autentifikácia

Tento spôsob je súčasťou protokolu HTTP. Princípom týchto metód je, že v každej požiadavke sa v hlavičke odosielajú aj prihlasovacie údaje.

Výhody:

- pomerne jednoduchá implementácia

Nevýhody (najmä, ak je klientskou aplikáciou webový prehliadač):

- nie je možné zmeniť vzhľad prihlasovacieho okna
- nie je možné implementovať odhlasovanie (alebo iba s ťažkosťami)
- ťažko sa implementuje vypršanie (timeout) prihlásenia
- nemusí byť veľmi bezpečné – závisí od konkrétnej metódy, pozri nižšie

Rozoznávame 2 typy HTTP autentifikácie, tzv. HTTP Basic a HTTP Digest autentifikáciu.

HTTP Basic autentifikácia
Pri tomto spôsobe sa meno a heslo prenáša v hlavičke každej požiadavky na server a je zakódované iba pomocou base64 algoritmu (prakticky je to teda iba čistý text). Problémom tohto prístupu je nízka bezpečnosť. Je preto nutné, aby všetka komunikácie prebiehala iba šifrovane prostredníctvom HTTPS protokolu, aby sa prihlasovacie údaje nedali jednoducho odchytiť.

HTTP Digest autentifikácia
Pri tomto spôsobe sa prihlasovacie údaje taktiež prenášajú v hlavičke každej požiadavky na server. Nie sú však jednoducho dešifrovateľné – prenášajú sa vo forme hash-u. Pri prvej požiadavke neautentifikovaného klienta na server mu server v odpovedi okrem informácie o zamietnutí pošle aj informáciu o tom, akým spôsobom má prihlasovacie údaje hash-ovať. V požiadavkách sa následne odosiela iba tento hash.

Aj v tomto prípade je vhodné, aby spojenie bolo šifrované, keďže sa zvyčajne používa hash algoritmus MD5, ktorý je náchylný na útok kolíziou (útočník odchytí náš hash a následne nájde kombináciu mena a hesla, ktorá generuje rovnaký hash ako naše prihlasovacie údaje).

Ak budeme považovať technológiu HTTPS za spoľahlivú a použijeme ju, pravdepodobne by stačilo použiť iba HTTP Basic autentifikáciu.

Metódy s použitím tzv. „tokenu“
Princípom týchto metód je, že odoslanie prihlasovacích údajov prebehne iba raz a používateľ následne obdrží vygenerovaný tzv. „token“ – identifikačný reťazec, pod ktorým ho server rozpozná. Tento reťazec sa následne posiela v hlavičke každej požiadavky na server. Takýto reťazec má časovo obmedzenú platnosť.

Výhody:

- ak útočník zachytí komunikáciu, nedokáže to jednoducho využiť v budúcnosti, nezíska naše prihlasovacie údaje
- možnosť implementovať odhlasovanie
- časovo obmedzené prihlásenie
- možnosť autentifikovať používateľa prostredníctvom externej služby

Nevýhody:

- požiadavka, v ktorej používateľ odosiela svoje prihlasovacie údaje, môže byť odchytená a zneužitá, pokiaľ neprebieha šifrovane
- ak útočník zachytí náš token, môže ho zneužiť po dobu jeho platnosti

Z nevýhod vyplýva, že je vhodné, aby sa aj pri použití tejto metódy použil šifrovaný prenos údajov. Pri použití HTTPS ale metóda neprináša veľa výhod v porovnaní s Basic HTTP

autentifikáciou (za predpokladu, že HTTPS považujeme za bezpečnú technológiu). Ak nemôžeme použiť šifrovaný prenos, je táto metóda bezpečnejšia a vhodnejšia.

Ďalšou veľkou výhodou je, že týmto spôsobom dokážeme autentifikovať používateľa prostredníctvom externej služby. Na serveri teda netreba implementovať registráciu a správu používateľov, ale tieto úlohy sa môžu presunúť na externú službu. Rozlišujeme 2 typy autentifikácie prostredníctvom externej služby: OpenID a OAuth.

OAuth
Táto metóda sa dá použiť na autentifikáciu používateľa voči niektorej konkrétnej službe (napr. Facebook, Twitter a pod.). Týmto spôsobom môžeme overiť, že používateľ je skutočne registrovaný v nejakej službe a vyžiadať si o ňom niektoré informácie (tie, ktoré on sám povolí).

Nevýhodou je, že táto metóda je viazaná na konkrétnu službu, je teda menej univerzálna. Je preto treba zvoliť službu, ktorá je používaná čo najväčším počtom používateľov. Výhodou je jednoduchšia implementácia ako v prípade OpenID.

OpenID
OpenID je technológia, ktorej cieľom je vytvoriť univerzálny otvorený decentralizovaný systém overovania identity. Jednou z hlavných myšlienok je, že používateľ si sám môže zvoliť poskytovateľa OpenID podľa svojho želania. V aplikácii teda stačí implementovať overovanie pomocou OpenID a neobmedzíme tak prihlasovanie iba na používateľov jednej služby. Nevýhodou je zložitejšia implementácia ako v prípade metódy OAuth.

Klientska časť webová stránka
Na klientskej časti pri webovej stránke je potrebné navrhnúť grafické rozhranie, pomocou ktorého bude možné zadať prihlasovacie údaje do systému, pomocou ktorých sa používateľ prihlási do systému. Ďalej je potrebné navrhnúť, ako poslať požiadavku s potrebnými údajmi na server a údaje z odpovede, poslanej zo servera získať a poskytnúť používateľovi. Tu by bolo vhodné získané údaje zobraziť ako informácie o prihlásenom používateľovi. To je cieľom ďalšej úlohy – *Zobrazenie informácií o prihlásenom používateľovi.*

Návrh riešenia

Vzhľadom na to, že cieľom tohto príbehu je vyskúšať si prihlasovanie z klientskej časti aplikácie a vyskúšať, ako sa vymedzujú chránené zdroje na serveri, zvolili sme zatiaľ iba najjednoduchšiu HTTP Basic autentifikáciu.

Rest
Požiadavky majú rovnaký formát, ako v prípade vkladania a vyberania štruktúrovaných informácií (pozri šprint 2), s týmito rozdielmi:

URI: reťazec /secured namiesto /structured
Hlavičky: Content-type: application/json; charset=UTF-8
Authorization: base64(meno:heslo)

Odpoveď
Nesprávne prihlasovacie údaje, nepodarilo sa autentifikovať používateľa
HTTP kód: 401 Unauthorized
Telo: JSON chybový objekt so správou o chybe

V prípade úspešnej autentifikácie sú odpovede rovnaké ako v rozhraní pre štruktúrované informácie.

Klientska časť webová stránka
Pre zadávanie prihlasovacích údajov je potrebné poskytnúť používateľovi grafické rozhranie, do ktorého bude možné tieto informácie zadať. Na to poslúžia grafické komponenty z knižnice *SmartGWT,* ktorá poskytuje dostatočné možnosti pre tvorbu grafických komponentov. Pre komunikáciu so serverom bude použitá http metóda *@Get,* pomocou ktorej získame existujúceho používateľa.

Implementácia

Rest
Bolo potrebné vytvoriť nový objekt triedy Authenticator. Následne ešte bolo potrebné vytvoriť ako jeden z jeho parametrov objekt Verifier, ktorý má za úlohu overiť, či zadané meno a heslo sú platné údaje. Použili sme najjednoduchšiu implementáciu MapVerifier, ktorý umožňuje zadať vopred pripravené páry meno:heslo do dátovej štruktúry Map. Vzhľadom na ciele tohto príbehu postačovalo zadať iba 2 rôzne páry „napevno“:
"xliptakm@gmail.com" a "heslomatej"
"lukaslipka@gmail.com" a "heslolukas"

Po vytvorení objektu Authenticator ešte bolo potrebné zvoliť, na ktorých URI adresách bude aktívny. To sa v Restlet frameworku realizuje pomerne jednoducho: pri konfigurácii jednotlivých adries v smerovači (objekt Router) sa ako ďalší parameter nastaví objekt Authenticator a všetky požiadavky na tieto adresy sú od tejto chvíle zabezpečené menom a heslom.

Klientska časť iPhone
Do hlavičky HTTP Requestu bolo priradené pole `Authorization`, ktoré obsahuje meno a heslo vo formáte `username:password` zašifrované v base64. Meno aj heslo bolo zatiaľ iba natvrdo v kóde zapísané, nie je žiadne okno, kde bude môcť používateľ toto meno a heslo zadať. Na zakódovanie reťazca bola vytvorená trieda `Base64`, ktorá obsahuje jednu metódu `encode(NSData*)plainText`.

URL adresa, na ktorú sa posielajú požiadavky bola zmenená z "http://branovaprva.appspot.com/rest/structured" na "http://branovaprva.appspot.com/rest/secured".

Klientska časť webová stránka
Pre implementáciu grafického rozhranie bola použitá knižnica SmartGWT. Komunikáciu so serverom zabezpečovalo Rest API rozhranie na serveri. Z klientskej časti je pripojenie umožnené pomocou proxy rozhrania *UserResourceProxy,* ktoré pomocou metódy retrieve, ktorá predstavuje http metódu *@Get,* získava používateľa v závislosti od zadaných prihlasovacích údajov. Detailnejší popis získavania informácií je popísaný v časti Návrhu prihlasovania. Po úspešnom prihlásení je používateľovi zobrazená stránka s informáciami o prihlásenom používateľovi. Neúspešné prihlásenie je indikované chybovou správou o neúspechu prihlásenia.

Testovanie

Najskôr bolo posielanie dát vyskúšané bez nastavenia autentifikácie, teda úplne bez mena a hesla, ako to bolo doteraz. Žiadna potvrdzovacia správa, ktorá by nám povedala pod akým ID sa uložila správa sa nezobrazila (a samozrejme ani 2D kód). Správa sa na serveri ani

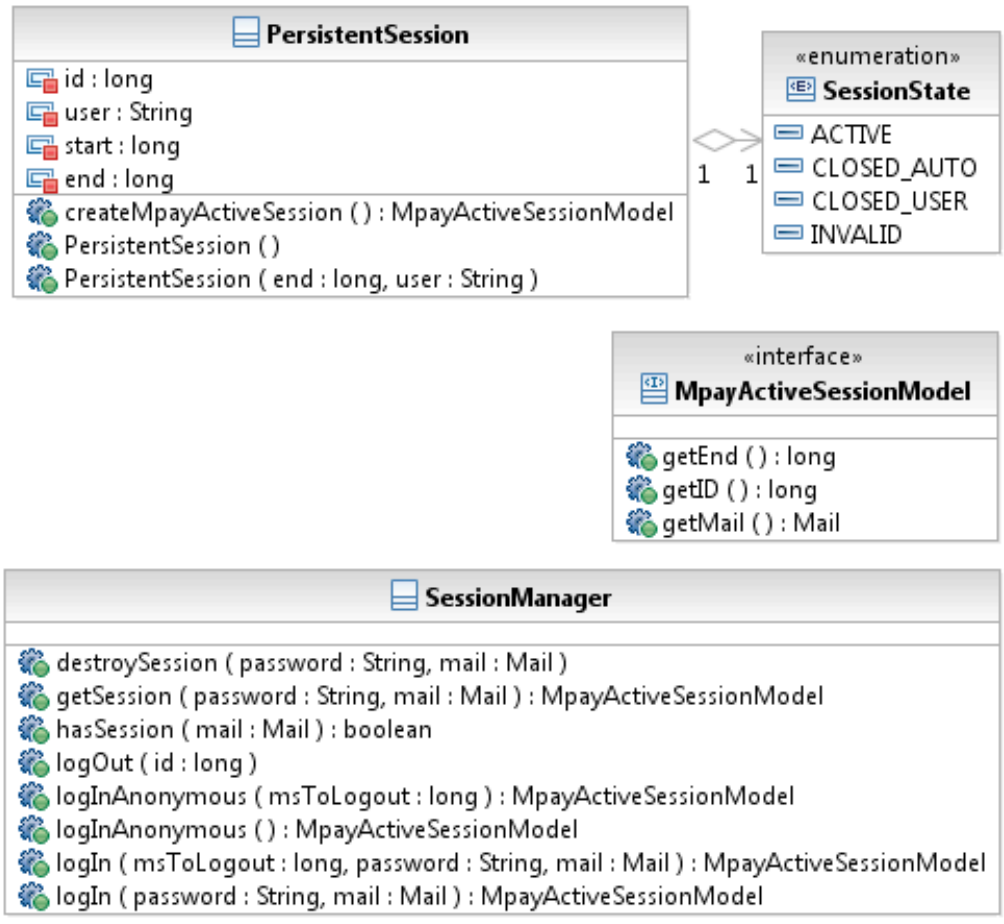
neuložila, pretože ako HTTP Response kód sa nevrátilo 201 (čo znamená vytvorenie objektu na serveri).

Keď sa použilo správne testovacie meno a heslo ("xliptakm@gmail.com" a "heslomatej"), správa sa úspešne uložila na serveri a na klienta sa vrátila správa s ID, a zobrazil sa 2D kód, v ktorom bola správa uložená. Pri pokuse odoslať správu s nesprávnym menom, alebo heslom, žiadna potvrdzovacia správa nepríde, aplikácia sa správa, ako keby sa meno a heslo vôbec nezadalo. To isté platí aj pri získavaní správy zo systému. Bez použitia autentifikácie, alebo použitím nesprávneho mena a hesla nepríde zo servera správa obsahujúca uložený reťazec. Správa príde iba so správnym menom a heslom.

Ak sa pri odoslaní použijú jedni testovacie prihlasovacie údaje ("xliptakm@gmail.com" a "heslomatej"), pri získavaní správy sa môžu použiť iné prihlasovacie údaje ("lukaslipka@gmail.com" a "heslolukas"). Jeden používateľ má teda prístup k tomu, čo do systému uložil druhý používateľ.

Jadro pre správu prihlásených používateľov a ich Session

Manažment sedení sleduje prihlásenie používateľa a vracia používateľovi identifikačné číslo, pomocou ktorého komunikuje so serverom. Model tejto časti systému je zobrazený na obrázku (Obrázok 5). Významnou vlastnosťou manažmentu sedení je možnosť používať anonymné prihlásenie, kedy sa používateľ neautorizuje pomocou mailu a hesla, ale vyžiada si identifikačné číslo sedenia zo servera a komunikuje iba pomocou daného čísla. V 3. šprinte bol upravený manažment používateľov tak, aby každý používateľ mohol získať svoje aktuálne sedenie. Sedenie je časovo obmedzené a po uplynutí preddefinovanej časovej doby je používateľ odhlásený.



Obrázok 5 Diagram tried jadra systému pre sedenie používateľov.

V budúcnosti sa očakáva rozšírenie možností pre nastavenie trvania sedenia používateľom. Manažment sedení je taktiež nástrojom jednoduchej bezpečnosti, avšak nie je na to primárne určený.

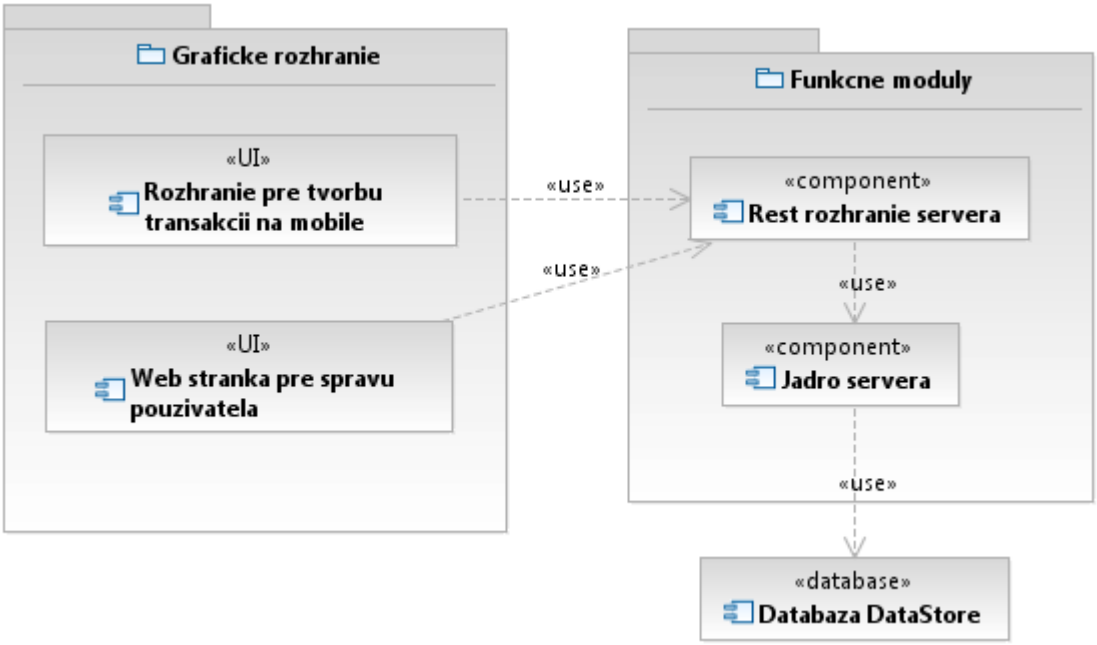
Testovanie jadra servera

Pre overenie funkčnosti servera sa používajú jednotkové testy. Cieľom týchto testov je hlavne testovanie perzistencie dát. Pre tento účel bola vytvorená testovacia trieda *JDOTestCase, ktorá* v metóde *setUp() pripraví prostredie pre testovanie (spustí službu pre perzistenciu). Po vykonaní testov sa v metóde tearDown()* prostredie ukončí. Trieda tiež obsahuje základné objekty pre testovanie, ktoré by bolo potrebné vytvárať pre každý JDO test samostatne. V 3. šprinte boli vytvorené jednotkové testy pre manažment sedení, manažment používateľov, všeobecný mechanizmus perzistencie. Tieto testy je možné opakovať po pridávaní funkcionality, čím zabezpečíme funkčnosť pôvodnej implementácie po implementovaní novej funkčnosti.
Výsledky testov po 3 šprinte sú nasledujúce:

- UserManagerTest úspešných 7 testov zo 7
- MpayPersistenceMangerTest úspešné 2 testy z 2
- MailTest úspešné 4 testy zo 4
- SessionManagerTest úspešných 8 testov z 8

Zhrnutie šprintu č. 3

Cieľom 3. šprintu bolo dokončiť web stránku pre registráciu používateľa, ktorá pre problémy implementácie nebola ukončená v 2. šprinte. Táto úloha bola úspešne ukončená. V ďalšej časti bolo potrebné nadviazať na registráciu používateľa a umožniť jeho prihlásenie ako na webovej stránke, tak aj na klientovi na iPhone. Tento príbeh bol ale splnený opäť len z časti, keďže sa čakalo za implementáciou registrácie používateľa. V rámci tohto príbehu sa podarilo úspešne analyzovať a navrhnúť spôsob, akým sa bude v prvotnej aplikácii prihlasovať. Taktiež bolo úspešne zapojené prihlasovanie do klientskej časti na strane iPhone klienta, kde sa využíva pri posielaní správe na server. Kvôli neskorej implementácii registrácie je prihlasovanie realizované iba pomocou natvrdo napísaných prihlasovacích údajov priamo v zdrojovom kóde. Webová stránka, pomocou ktorej by bolo možné prihlásiť sa do transakčného systému ale nebola včas dokončená a bola presunutá do nasledujúceho šprintu.
Architektúra systému vyzerá po tomto šprinte nasledovne (Obrázok 6):
Rozhranie pre tvorbu transakcií na mobile – tento komponent bol obohatený o prihlasovanie pri tvorbe transakcií a taktiež o vybratie informácie z transakčného systému na základe 2D kódu.
Web stránka pre správu používateľa – v tomto komponente bola dokončená registrácia používateľa. Prihlasovanie používateľa ale nebolo kvôli dostatku času a zdrojov doimplementované.
Rest rozhranie servera – v tomto komponente bola pridaná funkcionalita pre umožnenie prihlasovania sa z klientskych častí (iPhone i webová stránka).
Jadro servera – v tomto komponente bola doplnená funkcionalita pre manažment sedení používateľa.
Databáza Datastore – tento komponent slúži na ukladanie dát, ktoré je potrebné v systéme uchovávať. V tejto časti je zatiaľ ukladanie spravené pre používateľov systému.



Obrázok 6 Architektúra systému po 3. šprinte.

Revízia 3. šprintu

Táto revízia zachytáva zmeny, ktoré sa vykonali v REST rozhraní pre registráciu používateľa po automatických testoch.

Testy sú automatizované a sú implementované v triede *RestRegistrationTest*. Testovacie metódy pokrývajú nasledujúce správanie:

- aplikácia odpovedá HTTP kódom 409 pri pokuse o viacnásobnú registráciu rovnakého používateľa
- aplikácia odpovedá HTTP kódom 201, ak je registrácia úspešná
- aplikácia odpovedá HTTP kódom 400, ak je meno používateľa vynechané
- aplikácia odpovedá HTTP kódom 400, ak je meno používateľa null
- aplikácia odpovedá HTTP kódom 400, ak je heslo používateľa vynechané
- aplikácia odpovedá HTTP kódom 400, ak je heslo používateľa null
- aplikácia odpovedá HTTP kódom 400, ak je mail používateľa vynechaný
- aplikácia odpovedá HTTP kódom 400, ak je mail používateľa null
- aplikácia odpovedá HTTP kódom 400, ak je mail používateľa neplatný

Šprint č. 4

Úlohou 4. šprintu je začať s implementáciou transakcií, ktoré budú prebiehať v rámci transakčného systému. Podarilo sa nám identifikovať viacero typov transakcií. Do konca semestra si ale kladieme za úlohu implementovať len jeden z identifikovaných typov. V rámci toho je potrebné analyzovať transakcie a navrhnúť ich vhodné spracovanie na strane servera. Taktiež je potrebné navrhnúť, ako sa budú jednotlivé transakcie zobrazovať na webovej stránke.
Pre tento šprint boli identifikované tieto príbehy:

1. Analýza a návrh transakcií
2. Spracovanie transakcií na strane servera
3. Prihlásenie sa do transakčného systému
4. Zobrazenie transakcií

Analýza a návrh transakcií

Cieľom tohto príbehu bolo stretnúť sa v rámci tímu a prekonzultovať možné transakcie, ktoré bude možné vykonať v našom systéme.

Analýza typov transakcií

Pod transakciou sa myslí sled správ, ktoré sú vymieňané medzi serverom a klientskymi zariadeniami. Identifikovali sme tri možné scenáre transakcií, ktoré by mohli prebiehať pri simulovaní mobilného bankovníctva:

- Jeden klient sa rozhodne, že chce poslať presnú sumu peňazí.
- Jeden klient žiada od druhého presnú sumu peňazí, druhý klient zosníma 2D kód, potvrdí sumu a odošle ju.
- Prvý klient zadá maximálnu sumu, ktorú je ochotný zaplatiť a vygeneruje 2D kód. Druhý klient požaduje určitú sumu, zosníma 2D kód a suma sa okamžite zaplatí, ak neprevyšuje zadanú maximálnu sumu.

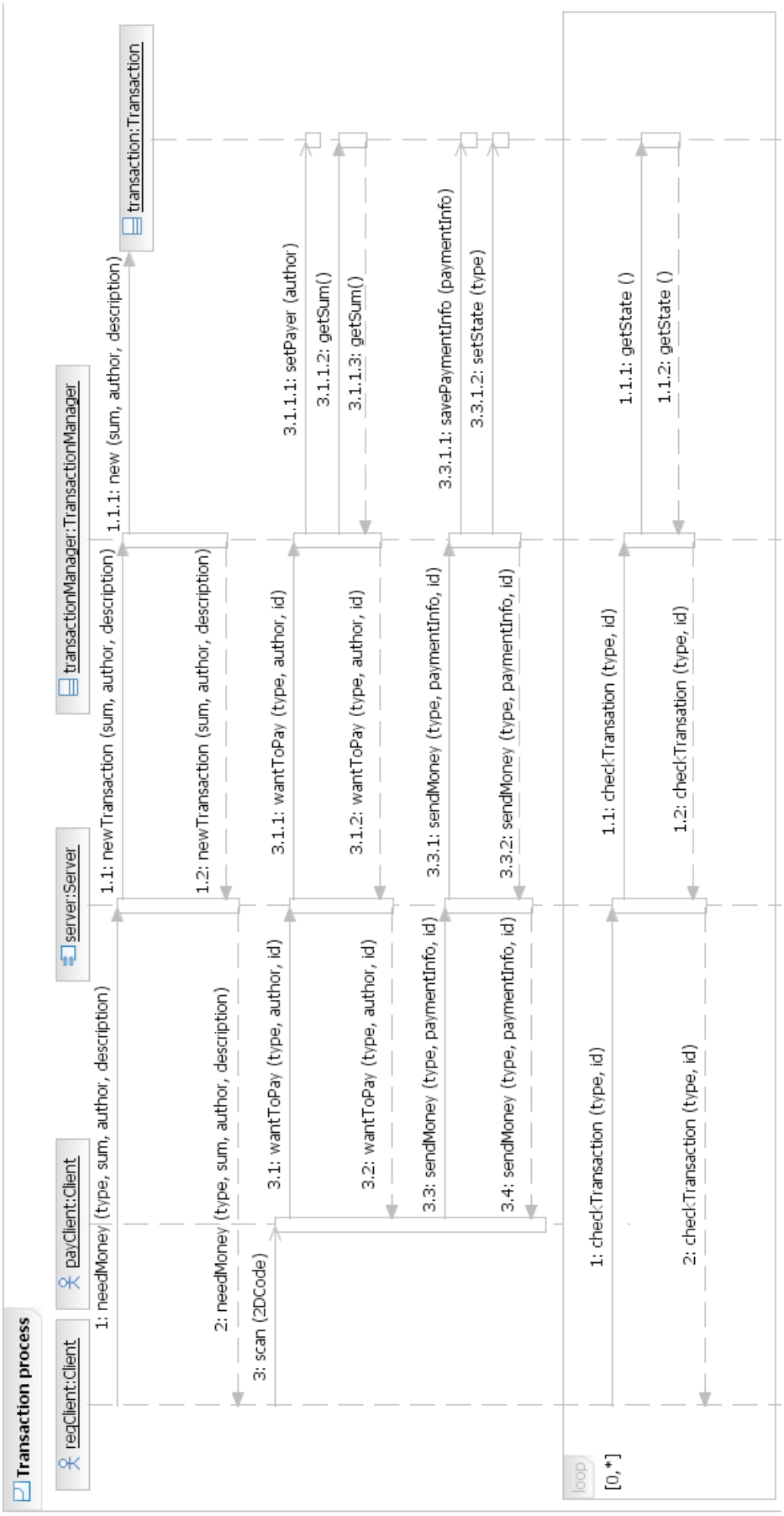
V mobilnom bankovníctve sa môžu vyskytnúť minimálne tieto scenáre. Pre naše potreby sa budeme v najbližšom období zaoberať hlavne druhým scenárom. V budúcnosti je pravdepodobné, že spracujeme aj ostatné scenáre, prípadne identifikujeme iné.

Návrh priebehu transakcie

Pre každý scenár transakcie je potrebné presne premyslieť, ako bude prebiehať, aby sa predišlo nedostatkom, a aby sa presne vedelo, čo bude neskôr potrebné implementovať v rozhraní, ktoré správy budú posielané a čo sa bude ukladať.

Ako bolo spomenuté v analýze, momentálne sa zaoberáme 2. scenárom, teda ide o prípad, kedy jeden klient žiada od druhého presnú sumu peňazí, druhý klient zosníma 2D kód, potvrdí sumu a odošle ju. Bližšie rozpracovaný priebeh tohto scenára je možné vidieť na obrázku (Obrázok 7). Celá transakcia sa bude skladať z viacerých správ, ktoré budú vymieňané medzi klientom a serverom, pričom počet klientov bude aspoň 2.

V prvom kroku pošle klient z mobilného zariadenia správu na server, že žiada zaplatiť. V tejto správe zahrnie sumu, ktorú žiada a typ správy (v tomto prípade "NEED_MONEY"). Z tejto správy bude potrebné zistiť, komu je potrebné zaplatiť, preto sa z nej bude musieť zistiť aj jej autor.



Obrázok 7 Scenár transakcie.

Táto správa naštartuje na serveri transakciu - manažér transakcií vytvorí novú transakciu pod číslom ID so stavom "PENDING", do ktorej uloží autora správy, síce zakladateľa transakcie, ale de facto jej adresáta, sumu, ktorú žiada zaplatiť a popis správy, v ktorom môžu byť upresňujúce informácie o platbe (napríklad začo žiada zaplatiť). Ako odpoveď klientovi na túto správu pošle server ID transakcie, pod ktorým je uložená. Typ tejto správy bude "NEW_TRANSACTION". Keď príde klientovi toto ID, vygeneruje sa mu z neho 2D kód a zobrazí sa na displeji.

Druhý klient, ktorý má zaplatiť, zosníma 2D kód z displeja prvého klienta a dekóduje ho, čím získa ID transakcie, pod ktorým je uložená v transakčnom systéme. Odošle na server správu, ktorou oznámi, že chce platiť. V tejto správe bude opäť zahrnutý jej autor, tentoraz platca. Okrem neho bude ešte uvedené ID transakcie, ktorú ide zaplatiť a ako typ správy bude uvedený "WANT_PAY". Manažér transakcií uloží autora správy ako odosielateľa peňazí. Po tomto server odošle klientovi sumu peňazí, ktorú má zaplatiť. To bude v prípade potvrdzujúcej správy, ktorá bude obsahovať typ "OK". Neskôr sa v tejto fáze transakcie možno pridá aj ďalší typ "ERROR", ktorý by mohol nastať v prípade, že by transakcia už nebola platná, alebo že by nebola v stave, že by ju bolo potrebné zaplatiť. Momentálne však pre náš systém neočakávame ani jeden z týchto prípadov, preto zatiaľ uvažujeme iba typ "OK".

Keď na klientske zariadenie príde suma peňazí, ktorú musí zaplatiť, bude to môcť potvrdiť, alebo jednoducho zrušiť. Ak to potvrdí, odošle na server potvrdzujúcu správu, ktorá bude obsahovať ID transakcie, typ správy (v tomto prípade "SEND_MONEY") a ďalšie možné údaje o platbe, ktoré budú potrebné. Keď príde táto správa na server, transakčný manažér uloží údaje o platbe do transakcie a zmení stav na "COMPLETED", aby sa vedelo, že platca už potvrdil platbu, a že sa očakáva, že peniaze sú odoslané. V tomto bode (ešte pred tým, než transakčný manažér nastaví nové údaje transakcii) sa neskôr uskutoční reálna banková transakcia z účtu na účet. Zatiaľ túto operáciu viac neriešime, keďže sme len vo fáze simulovania. Potom sa znovu vráti zo servera správa, ktorá potvrdí zaplatenie (s typom správy "OK").

Tým by sa mala transakcia ukončiť - peniaze by mali byť vyplatené. Ešte je ale potrebné, aby aj prvý klient mal odozvu od servera, že peniaze už sú zaplatené. Bude môcť odoslať na server správu, ktorou skontroluje stav transakcie. Na to bude slúžiť typ správy "CHECK". Spolu s typom odošle aj ID transakcie, ktorej stav chce skontrolovať. Ako odpoveď dostane od servera stav transakcie. V tejto správe bude ako typ nastavený "STATE". Tieto dve správy sa môžu zopakovať viac krát, pretože klient môže chcieť skontrolovať stav transakcie priskoro. Vtedy mu môže prísť odpoveď, že transakcia ešte nie je ukončená. V tom prípade môže znovu skontrolovať stav transakcie. Toto ale nie je nevyhnutné, klient vôbec nemusí skontrolovať stav transakcie, ak nechce. Ide len o to, aby mal možnosť hneď si pozrieť, či platca zaplatil.

Rest
Na každú z nižšie uvedených požiadaviek je v prípade, že autentifikácia je neúspešná, nasledujúca odpoveď (neuvádzame ju pri každej požiadavke, ale spoločne na začiatku, keďže sa nemení).

Odpoveď
Nesprávne prihlasovacie údaje
HTTP kód: 401 Not Authorized
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Nasleduje opis možných požiadaviek.

Požiadavka
Vytvorenie novej transakcie typu NEED_MONEY
HTTP metóda: POST
URI: /transaction
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “NEED_MONEY”,
  “sum”: suma,
  “description”: “popis, za čo je potrebné zaplatiť”
}
```

Odpoveď
Úspešné vykonanie požiadavky, vytvorenie novej transakcie
HTTP kód: 201 Created
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
{
  “id”: identifikačné číslo vytvorenej transakcie,
  “uri”: “URI vytvorenej transakcie”
}
```

Požiadavka
Zistenie podrobných informácií o transakcii
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácie o transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “typ transakcie”,
  “sum”: suma,
  “description”: “popis, za čo je potrebné zaplatiť”
  “state”: “stav transakcie”
}
```

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Odoslanie peňazí do transakcie
HTTP metóda: PUT
URI: /transaction/{identifikačné číslo transakcie}
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “SEND_MONEY”,
  “sum”: suma
}
```

Odpoveď
Platba úspešne vykonaná
HTTP kód: 200 OK
Telo: prázdne

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Platba zamietnutá – do transakcie sa nedajú odoslať peniaze
V prípade, že transakcia už bola uzavretá a pod.
HTTP kód: 400 Bad Request
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Platbu sa nepodarilo spracovať kvôli chybe
Napríklad sa nepodarilo komunikovať s bankou, na účte je malý zostatok a pod.
HTTP kód: 500 Internal Server Error
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Zistenie stavu transakcie
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}/state
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácia o stave transakcie
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “state”: “stav transakcie”
}
```

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Spracovanie transakcií na strane servera

Cieľom tohto príbehu je navrhnúť a implementovať, ako bude prebiehať spracovanie transakcií na strane servera.

Analýza a návrh spracovania transakcií v jadre servera

Táto analýza vychádza z ukončenej úlohy #84 *Analýza a návrh transakcií*. V úlohe sme vytvorili dokumentáciu k základnému typu transakcie „Užívateľ požaduje peniaze“. V budúcnosti sa počet a typ správ bude zväčšovať vzhľadom na ďalšie typy transakcií. Dokumentácia obsahuje potrebné typy správ a scenár transakcie. Pomocou analýzy tohto scenára sme definovali dve kategórie správ s rôznymi vlastnosťami:

- správy odoslané klientom
    - vždy obsahujú autora
    - môžu vytvárať a ukončovať transakcie
    - môžu kontrolovať priebeh transakcie
    - správy ktoré nevytvárajú transakciu obsahujú ID existujúcej transakcie
    - definujú metódu „onMessageReceived“ ktorá implementuje funkcionalitu správy a je invokovaná pri prijatí správy manažérom transakcií
- správy odoslané serverom
    - sú vytvorené na základe správy prijatej z klienta
    - vždy sa vzťahujú na transakciu

Typy správ odosielaných klientom sú:

- „CHECK"
- „SEND_MONEY“
- „NEED_MONEY“
- „WANT_MONEY“

Typy správ odosielaných serverom sú:

- „NEW_TRANSACTION“
- „STATE“
- „OK“
- „ERROR“ – zatiaľ sa neočakáva implementácia

Stav transakcie môže byť:

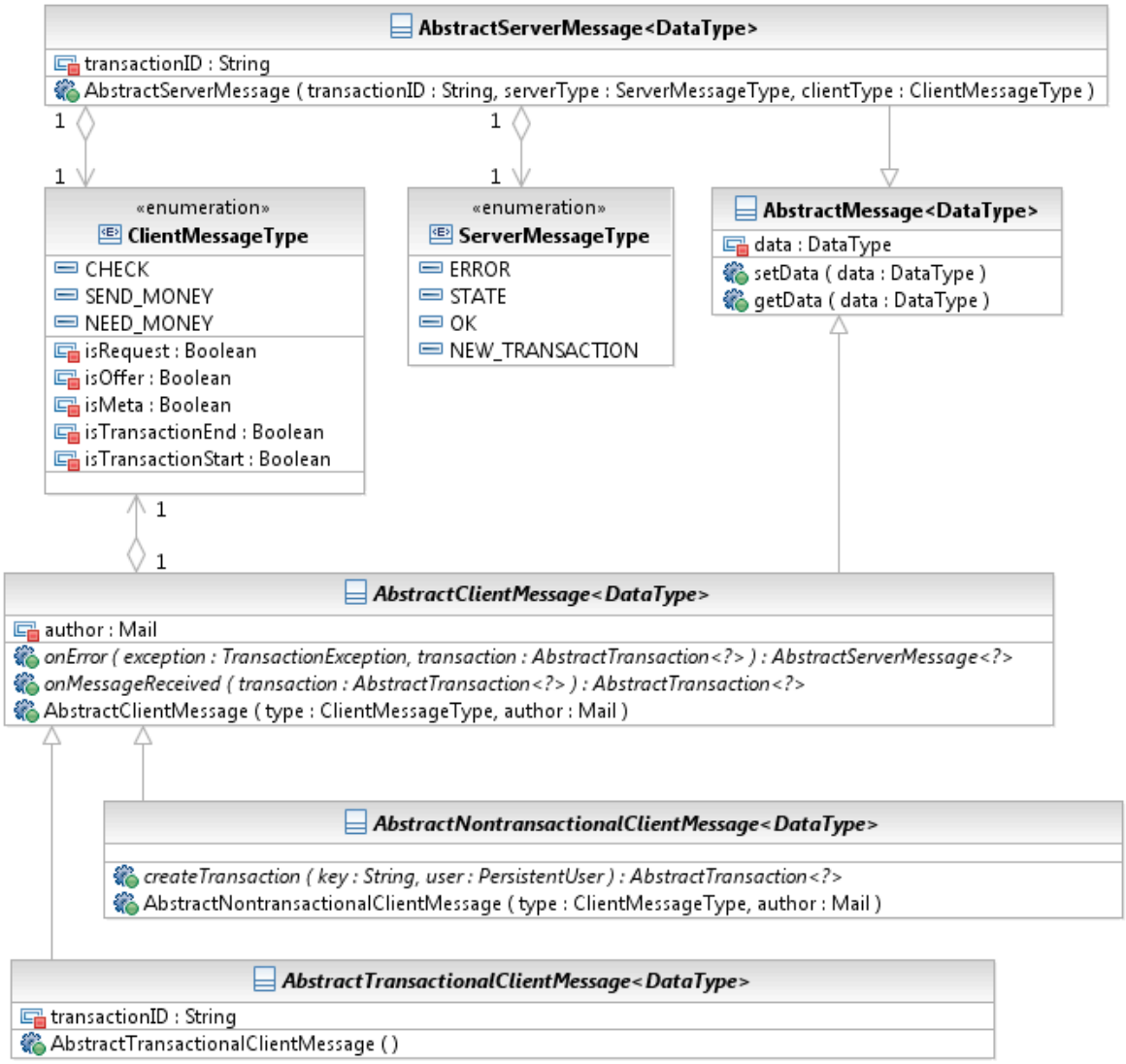
- PENDING
- COMPLETE
- CANCEL

Stav transakcie a typy správ sú navrhnuté tak aby ich bolo možné implementovať pomocou enumeračných typov v jazyku Java. Každá konkrétna správa obsahuje typ správy, ktorý hovorí o určitých vlastnostiach správy.
Vďaka rozdeleniu správ na klientske a serverové je možné lepšie rozdeliť rozhranie, kde na vstupe sa očakáva iný formát správ ako na výstupe. Taktiež to uľahčuje možnosť

implementácie správ pomocou polymorfizmu. Nakoľko implementácia transakcií a správ musí byť v tejto fáze vývoja všeobecná a flexibilná do budúcnosti je nutné definovať abstraktné správy ako generické objekty. Definovali sme jeden generický parameter pre abstraktnú správu:

- typ dát ktoré prenáša

Výsledný návrh abstraktného modelu správ je zobrazený na obrázku (Obrázok 8). Tento model obsahuje iba abstraktné triedy na základe ktorých vzniknú konkrétne implementácie správ. Model obsahuje iba najvýznamnejšie prístupové metódy („get“ a „set“) aby bol model kompaktný.



Obrázok 8 Model návrhu správ.

Implementácia spracovania transakcií na strane servera

Po implementovaní abstraktného modelu boli vytvorené základné správy pre vykonanie jednoduchej transakcie. Transakcia obsahuje údaje o platiteľovi a o sume ktorú treba zaplatiť. Implementácia správ je taktiež rozdelená na dve časti, a to na:

- všeobecné správy – tieto správy sa budú používať bez zmeny aj v iných transakciách, medzi ne patrí správa „CHECK“, „OK“, „ERROR“, „NEW_TRANSACTION“ a „STATE“
- správy pre konkrétnu transakciu – tieto správy sú platné len pre definovanú transakciu, v budúcnosti budú vytvorené ich komplexnejšie verzie a verzie pre nové druhy transakcií.

Model implementovaných správ možno vidieť na obrázku (**Obrázok 9**). Súčasťou spracovania transakcií sú aj triedy pracujúce so správami. Tieto triedy obsahujú základnú logiku prijatia správy na serveri, jej spracovania a vrátenia návratovej správy. Biznis logika je implementovaná v konkrétnych implementáciách klientských správ. Tento prístup má výhody:

- triedy na prácu s transakciami sú jednoduché, pri rozšírení funkcionality sa triedy transakcií menia minimálne
- každej správe je možné priradiť priamy dopad na transakciu
- implementácia logiky v správe je jednoduchá, ľahko doplniteľná
- logika predstavuje testovanie vstupných podmienok a vykonanie akcie na základe testovania - nie je potrebné uchovávať históriu správ alebo zložitý stavový diagram
- správu je možné nahradiť inou správou bez väčších problémov

Nevýhodami tohto prístupu je:

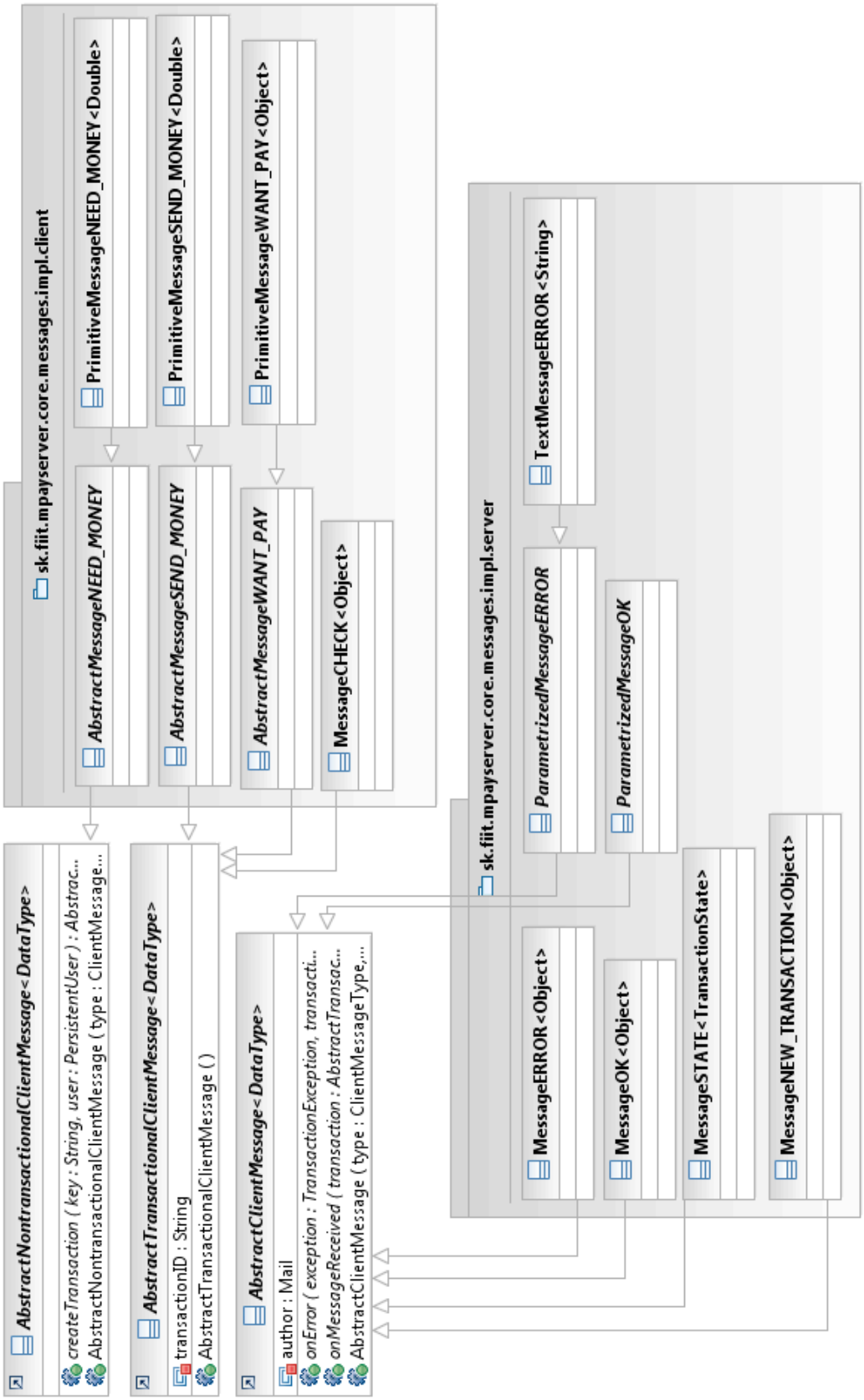
- roztrúsenie funkcionality do viacerých správ
- väčšia komplexnosť pri spracovaní transakcie

Testovanie spracovania transakcií

Pre overenie funkcionality jadra bol vytvorený jednotkový test „sk.fiit.mpayserver.core.transaction.TransactionTest“, ktorý testuje:

- vytvorenie transakcie pri prijatí správy NEED_MONEY
- vykonanie celej transakcie požiadania a poskytnutia peňazí
- chybové stavy ktoré môžu nastať pri posielaní správ
    - použitie nesprávnej hodnoty v správe
    - použitie null hodnoty v správe
    - poslanie nesprávnej správy
    - poslanie správy nesprávnym autorom
    - nutnosť prihlásenia pri vykonávaní transakcie

Pomocou tohto testu bola odhalená nasledujúca chyba technológie GWT:
Perzistentná verzia transakcie používa ako primárny kľúč hodnotu typu „String“. Pri spracovaní tejto hodnoty pomocou mechanizmu perzistencie sa nekontroluje či je textový reťazec rovnaký, ale kontroluje sa iba výskyt hľadaného reťazca v hodnote kľúča. Preto môže pre hľadaný kľúč „transakcia1“ byť vrátená transakcia s kľúčom „transakcia“. Kľúče sú generované automatickým mechanizmom, avšak klient môže nastaviť kľúč transakcie podľa svojich potrieb a preto bolo tento stav potrebné ošetriť. Problém bol ošetrený testovaním dĺžky reťazcov. Po otestovaní implementácie spracovania transakcií boli spustené všetky jednotkové testy pre jadro servera. Nimi bola overená plná funkcia predošlej funkcionality.



Obrázok 9. Model implementovaných správ.

Prihlásenie sa do transakčného systému

Analýza problému

Ide o pomerne jednoduchý problém, ktorý zahŕňa odoslanie dát vo vopred definovanom formáte zo servera klientovi. V rámci bezpečnosti treba zohľadniť nasledujúce aspekty.

- Povoliť zobraziť iba informácie o prihlásenom používateľovi – nie o iných používateľoch. Takáto funkcionalita by síce mohla byť potrebná neskôr (napr. s administrátorským účtom), ale keďže zatiaľ nie je určené ani to, či administrácia bude prebiehať cez REST rozhranie, v duchu pravidla YAGNI (You Ain't Gonna Need It – nebudete to potrebovať) takúto možnosť zatiaľ do rozhrania nepridávame.
- Na klienta by sa nemalo odosielať heslo zo servera – obrana pred tým, aby niekto nevytvoril aplikáciu, ktorá by umožnila používateľovi zobraziť jeho heslo v čitateľnom tvare (vážny bezpečnostný nedostatok). To je čiastočne vyriešené tým, že v aplikácii sa heslá ani nikde neukladajú ako čistý text, ukladá sa iba ich „message digest“ (známy aj ako „hash“) – táto vlastnosť vyplynula už z predchádzajúcich šprintov. Nie je však vhodné na klienta odosielať ani tento „hash“, keďže aj to predstavuje určité bezpečnostné riziko (môže zistiť detaily o našom algoritme a potenciálne nájsť aj kolízny reťazec k heslu).

Návrh riešenia

Pre problém 1 navrhujeme ako riešenie, že rozhranie webovej služby nebude podporovať možnosť vybrať si, o akom používateľovi sa majú informácie zobraziť. Preto sme navrhli, že k týmto informáciám sa bude pristupovať na jednej adrese a aplikácia z HTTP hlavičiek sama zistí, kto je momentálne prihlásený a zobrazí potrebné informácie. To je rozdiel napr. oproti prípadu výmeny reťazcov (pozri šprint 1), kde sa pre získanie reťazca musel zadať jeho identifikátor do adresy.

Ďalej si tento problém vyžaduje rozšíriť už implementované jednoduché prihlasovanie (pozri príbeh *Jednoduché prihlasovanie)* o možnosť prihlasovania registrovaných používateľov, keďže sa majú na klienta odoslať informácie, zadané pri registrácii, čo s aktuálnou implementáciou prihlasovanie nie je možné.

Problém 2 sa vyrieši jednoducho tým, že pole s hashom sa do odpovede nezahrnie.

REST

Požiadavka
Získanie informácie o prihlásenom používateľovi
HTTP Metóda: POST
URI: /user/me.json
Hlavičky: Authorization: base64(mail:heslo)
Telo: prázdne

Odpoveď
Úspešné vykonanie požiadavky – používateľ je prihlásený
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “name”: “Meno používateľa”,
  “mail”: “E-mail, s ktorým sa používateľ registroval”
}
```

Nesprávne prihlasovacie údaje
HTTP kód: 401 Not Authorized
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Implementácia

REST
Implementácia prihlasovania registrovaných používateľov spočívala vo vytvorení triedy *MPayHttpAuthenticator*. Ako jeden z parametrov konštruktora táto trieda dostáva objekt *UserManager*, ktorý predstavuje službu, ktorá umožňuje zistiť, či je používateľ v systéme zaregistrovaný s určitým heslom. Na základe volaní tejto služby potom *MPayAuthenticator* môže každý pokus o prihlásenia povoliť alebo zamietnuť podľa toho, či sú prihlasovacie údaje správne. Autentifikátor na konkrétnych URI potom aktivujeme rovnako ako v príbehu *Jednoduché prihlasovanie.*

Aby používateľ mohol získať požadované údaje o svojej registrácii, bolo treba rozšíriť triedu *ServerUserResource* o metódu *@Get retrieve*. Tá z HTTP hlavičiek zistí e-mail a heslo prihláseného používateľa pomocou funkcií frameworku Restlet *getChallengeResponse() .getIdentifier()* a *getChallengeResponse().getSecret()*. Následne použije službu *UserManager* z jadra servera a pomocou nej získa informácie o prihlásenom používateľovi. Zo získaných informácií následne vyskladá objekt s odpoveďou a ten vráti ako návratovú hodnotu. Restlet framework sa postará o skonvertovanie do formátu JSON a odoslanie odpovede používateľovi.

Testovanie

REST
Testovanie sa vykonáva automatizovane pomocou knižnice JUnit. Pre všeobecné informácie o automatizovanom testovaní REST rozhrania pozri kapitolu *Testovanie Rest rozhrania*. Testy sú implementované v triede *RestUserInfoTest*. Testovacie metódy pokrývajú nasledujúce správanie:

- aplikácia odpovedá HTTP kódom 401, ak používateľ nezadá nijaké prihlasovacie údaje
- aplikácia odpovedá HTTP kódom 401, ak používateľ zadá nesprávne prihlasovacie údaje
- aplikácia odpovedá HTTP kódom 200, ak používateľ zadá správne prihlasovacie údaje
- aplikácie vráti správne dáta pre používateľa, ktorý je prihlásený

Testovanie REST rozhrania

Keďže okrem jadra servera treba testovať aj správnu funkčnosť implementovanej webovej služby, bolo treba zanalyzovať rôzne možnosti testovania a navrhnúť vhodný spôsob.

Analýza problému

Rozlišujeme niekoľko typov testov, medzi nimi aj jednotkové a integračné testy. Jednotkové testy sa vyznačujú tým, že testujú jednotku (súčiastku, zvyčajne jednu triedu) v izolácií, pričom všetky jednotky, s ktorými jednotka spolupracuje, sa nahradia „falošnými“ implementáciami – anglicky tzv. test doubles. Pomocou týchto falošných implementácií môžeme ľubovoľne simulovať správanie spolupracujúcich jednotiek a otestovať tak všetky aspekty správania testovanej jednotky. Výhodou je, že takéto testy bežia rýchlo a nie sú

závislé na nastavení testovacieho prostredia. Nevýhodou je, že sa týmto spôsobom neotestuje spolupráca jednotiek.

Integračné testy testujú spoluprácu častí aplikácie. Ich výhodou je, že otestujú správanie aplikácie ako celku – teda funkčnosť jednotiek aj spoluprácu jednotiek. V takýchto testoch sa používajú skutočné implementácie, nie „falošné“ implementácie. Nevýhodou týchto testov je, že spravidla sa vykonávajú pomalšie (napr. kvôli prístupu do databázy) a treba zložitejšie konfigurovať testovacie prostredie – napr. vymazať databázu pred spustením každej testovacej metódy (aby sa metódy navzájom neovplyvňovali), spustiť server, na ktorý sa budú odosielať HTTP požiadavky a podobne.

Oba z týchto typov testov môžu pri vhodnom prístupe byť automatizované – je vhodné sa o to snažiť, keďže agilný vývoj prináša veľa zmien v aplikácii a často potrebujeme overiť, či tieto zmeny neželane neovplyvňujú funkčnosť iných častí aplikácie. Preto potrebujeme testy spúšťať často.

V ideálnom prípade je vhodné použiť oba typy testov, keďže jeden typ testov môže odhaliť chyby, ktoré v inom type zabudneme (alebo nedokážeme) pokryť. Takisto pri testovaní REST rozhrania by bolo možné použiť oba typy testov – jednotkové testy na konkrétne implementované triedy a integračné testy na otestovanie webovej služby ako celku.

Návrh riešenia

Z časových dôvodov zatiaľ implementujeme iba integračné testy. V porovnaní s jednotkovými testami poskytujú pre nás vhodnejší pomer prínosu k vynaloženému úsiliu, keďže pokryjú väčšiu časť funkcionality pri približne rovnakom úsilí.

Nevylučujeme ani možnosť, že v budúcnosti pridáme aj jednotkové testy, preto navrhujeme upraviť existujúci kód, aby bol ľahšie testovateľný. To zahŕňa odstránenie pevne definovaných závislostí medzi triedami z kódu a použitie injektovania závislostí (tzv. dependency injection). Aj keď sa môže zdať, že toto úsilie je zbytočné, zo skúseností môžem povedať, že vedie k lepšej architektúre aplikácie bez ohľadu na to, či budeme jednotkové testy implementovať. Viac o konkrétnej implementácii injektovania závislostí je uvedené v časti implementácia.

Ak by sme chceli striktne dodržovať pravidlá integračného testovania, museli by sme zakaždým pred spustením testov spustiť aj server s aplikáciou, vyprázdniť databázu a prípadne vykonať ďalšie podobné kroky. Tieto úkony sú ale pomerne obmedzujúce a časovo náročne a je navyše problematické ich zautomatizovať pomocou knižnice JUnit. Knižnicu JUnit preferujeme, keďže je podporovaná priamo vo vývojovom prostredí Eclipse a umožňuje testy spúšťať veľmi pohodlne. Preto sme navrhli tieto 2 výnimky:

- Odstránime nutnosť spúšťať server. Keďže úlohou servera je hlavne to, aby pomocou tzv. servletov danú HTTP požiadavku vo forme objektu odovzdal príslušnej triede na spracovanie, môžeme objekt s požiadavkou vytvoriť a odovzdať ho príslušnej triede aj programátorsky v testovacej metóde.
- Odstránime nutnosť mať databázu priamo na disku. Toto je možné vďaka flexibilnej architektúre cloud platformy Google App Engine, ktorá umožňuje nastaviť úložisko tak, že všetky dáta sa budú ukladať iba do operačnej pamäte. Pred spustením každej testovacej metódy teda môžeme mať prázdnu databázu, ktorá navyše bude bežať veľmi rýchlo (keďže ukladá iba do operačnej pamäte počítača).

Zavedením týchto výnimiek prídeme o možnosti

- Otestovať správne nastavenie servletov – táto vrstva je v prípade REST časti nášho

systému veľmi tenká a stará sa iba o presmerovanie všetkých požiadaviek na adresy začínajúce reťazcom *rest* na triedu s REST aplikáciou. Otestovať túto vrstvu automatizovane nepovažujeme za také dôležité, aby to vyvážilo spomínané nevýhody.

- Otestovať správnu funkčnosť produkčného úložiska v Google App Engine – v tomto prípade jednoducho považujeme implementáciu úložiska od Googlu za správnu a dôveryhodnú a predpokladáme, že aplikácia bude po výmene testovacieho úložiska za produkčné úložisko fungovať.

Oba z týchto aspektov ale otestujeme neautomatizovane vďaka iPhone klientom, ktoré REST rozhranie používajú, takže ani tieto časti systému neostanú neotestované.

Implementácia

Injektovanie závislostí
V Restlet frameworku je potrebné na injektovanie závislostí implementovať vlastnú triedu, ktorá rozširuje triedu *Finder*, nazvali sme ju *MPayInjector* (ďalej ako injektor). Túto triedu následne nastavíme pomocou príkazu *setFinderClass* na objekte triedy *Router*. Smerovač (*Router*) potom požiada injektor o vytvorenie objektu vždy, keď obdrží požiadavku na niektorý z REST zdrojov. Do objektov reprezentujúcich tieto zdroje (objekty tried dediacich od *ServerResource*) potom injektor môže ľubovoľne vkladať závislosti a celá logika týkajúca sa injektovania závislostí je tým vyčlenená do jednej triedy. Jednotlivé služby, ktoré má injektor vkladať sa nastavujú pomocou statických metód v tvare *set<NázovSlužby>*, napr. *setUserManager*. Injektor sa nachádza v balíku *sk.fiit.mpayserver.rest.di*.

Ak bude v budúcnosti vkladaných závislostí väčšie množstvo, zvážime aj použitie knižnice taoki (http://code.google.com/taoki), ktorá podporuje injektovanie závislostí do Restlet zdrojov pomocou široko používaného nástroja Google Guice. Zatiaľ ale takúto možnosť nepovažujeme za potrebnú.

Simulácia HTTP požiadaviek a odpovedí
Aby sme mohli simulovať HTTP požiadavky a odpovede bez nutnosti spusteného servera, postupujeme v našich testoch nasledovne. Vytvárame programátorsky nový objekt triedy *org.restlet.Request*, ktorej ako parameter odovzdávame požadovanú HTTP metódu a URI zdroja, na ktorý chceme pristupovať. Následne vytvárame objekt typu *org.restlet.Response*, ktorému ako parameter odovzdáme vytvorený *Request*. Následne vytvárame novú inštanciu triedy našej REST aplikácie – *MPayRestApplication*. Na tomto objekte potom zavoláme metódu *handle*, ktorej ako parametre odovzdáme vytvorený *Request* a *Response*. Potom už môžeme skúmať objekt *Response*, návratové HTTP kódy, obsah odpovede a iné vlastnosti, ktoré potrebujeme v konkrétnom teste overiť.

Použitie databázy v pamäti
Aby sa nepoužívala databáza na disku, ale iba jej implementácia v operačnej pamäti, riadili sme sa odporúčaným postupom konfigurácie od Google.

Do classpath projektu pridáme knižnice *appengine-testing.jar* a *appengine-api-stubs.jar*. Potom v testovacej triede vytvoríme objekt typu *LocalServiceTestHelper* nasledujúcim príkazom:

```
LocalServiceTestHelper testHelper
  = new LocalServiceTestHelper(new
LocalDatastoreServiceTestConfig());
```

Potom už pred každou testovacou metódou iba zavoláme príkaz *testHelper.setUp()* a po skončení každej testovacej metódy príkaz *testHelper.tearDown()*. V konkrétnom teste potom

môžeme ľubovoľne pristupovať k databáze a máme istotu, že neovplyvníme iné testy, keďže databáza sa po vykonaní každej testovacej metódy vytvorí v pamäti nanovo.

Zobrazenie transakcií

Používateľ si chce zobraziť transakcie na webovej stránke, ktoré vykonal, prípadne sa ich zúčastnil.

Analýza

V rámci tohto príbehu sme do tohto šprintu zaradili len jednu úlohu, a to vytvorenie používateľského rozhrania pre zobrazovanie transakcií na webovej stránke. Z toho plynie viacero dôsledkov:

- Keďže doteraz boli na webovej stránke implementované len registrácia a prihlasovanie používateľa, nevyžadovalo to žiadne zložité navrhovanie webovej stránky.
- Webová stránka nemala žiadnu štruktúru.
- Preto je potrebné navrhnúť, ako bude webová stránka vyzerať, identifikovať jednotlivé komponenty webovej stránky a implementovať grafické rozhranie pre zobrazovanie transakcií, ktoré ale zatiaľ nebude zobrazovať reálne transakcie.

Návrh

Návrh rozloženia a štruktúry stránky
Pri návrhu štruktúry stránky sme sa snažili dodržiavať základné princípy návrhu webových stránok. Identifikovali sme nasledovné prvky stránky (Obrázok 10):

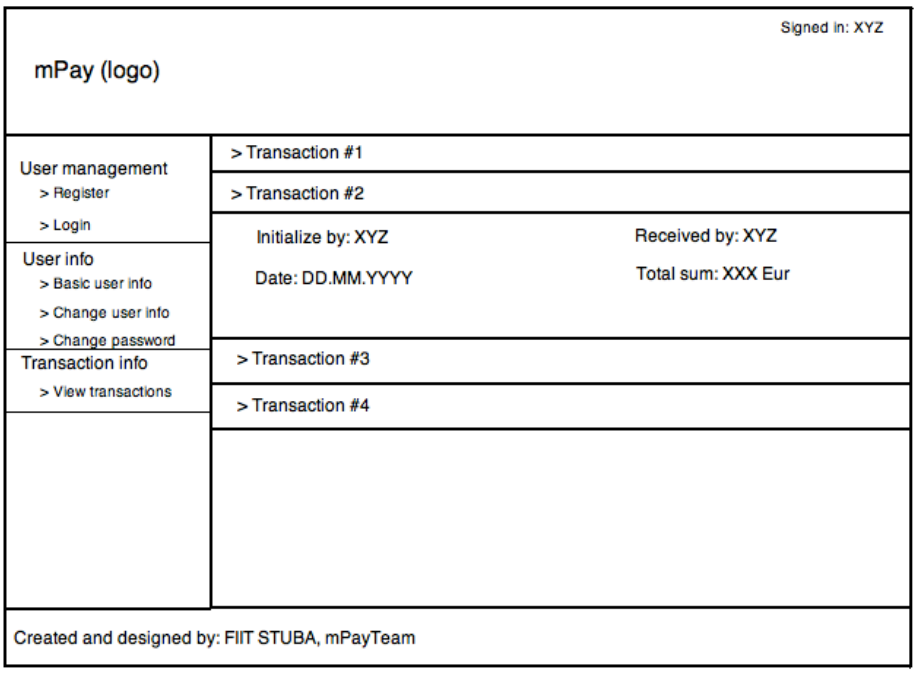
- *Header* – hlavička stránky, ktorá obsahuje logo stránky a informáciu o prihlásenom používateľovi.
- *Navigácia* – menu stránky, pomocou ktorého si používateľ môže zobrazovať jednotlivé stránky. V prvotných častiach by to malo obsahovať manažment používateľov (registrácia a prihlásenie), informácie o prihlásenom používateľovi a informácie o vykonaných transakciách pre prihláseného používateľa.
- *Obsahová časť stránky* – v tejto časti sa budú zobrazovať jednotlivé podstránky, ktoré bude používateľ vyžadovať.
- *Footer* – päta stránky, kde budú zobrazené autorské práva a informácie o autorovi.

Implementácia

Pre implementáciu webovej stránky sme použili knižnicu SmartGWT, ktorá poskytuje veľké množstvo grafických komponentov, ktoré sú vhodné pre tvorbu webovej stránky. Pomocou nich je tvorba štruktúry webovej stránky veľmi jednoduchá a jednotlivé komponenty sú po funkčnej i dizajnovej stránke veľmi vhodné.

Testovanie

Testovanie tejto časti zatiaľ nie je potrebné, nakoľko ide len o tvorbu štruktúry celej stránky, kde nie je implementovaná žiadna funkcionalita.

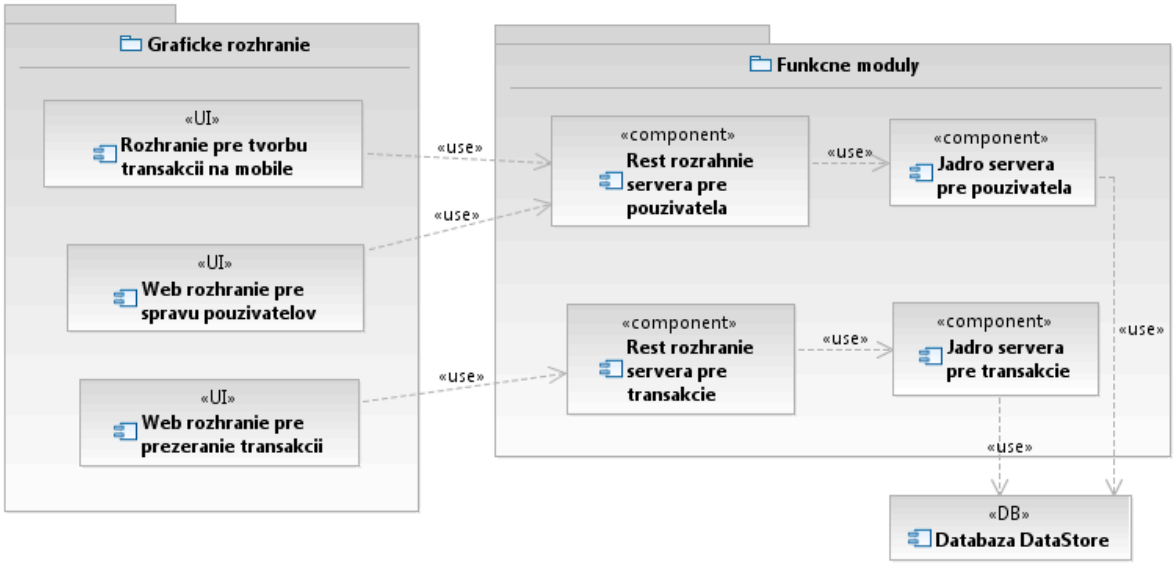


Obrázok 10 Štruktúra stránky.

Zhrnutie 4. šprintu

Cieľom 4. šprintu bolo začať používať transakcie, ktoré budú obsahovať informácie o rôznych platbách. V tomto šprinte sa podarilo implementovať na serveri funkcionalitu, ktorá umožňuje vykonávať transakcie medzi dvoma zariadeniami. Je umožnené tieto transakcie ukladať na serveri a vytvárať správy, pomocou ktorých bude komunikovať server so zariadením. Pri ukončení šprintu sme ale zistili, že implementácia správ nebola najlepším riešením. Po dohode tímu sa do ďalšieho šprintu zaradili úlohy, ktoré zmenia implementáciu, vytvorenú v tomto šprinte. V rámci klientskej časti sa na webovej stránke implementovalo prihlásenie sa do transakčného systému. Taktiež sa navrhla štruktúra webovej stránky, ktorá bude základom pre zobrazovanie vykonaných transakcií.

Architektúra systému vyzerá po tomto šprinte nasledovne (Obrázok 11):
V rámci architektúry sa zmenili alebo pridali tieto moduly:
Web stránka pre správu používateľa – bolo obohatené o možnosť prihlasovania a základného zobrazovania transakcií.
Rest rozhranie servera pre transakcie – bolo pridané rozhranie pre komunikáciu klienta so serverom pre vytváranie transakcií v module Rest rozhranie servera. Z toho sa vytvorili nové moduly.
Rest rozhranie servera pre používateľa – vzniklo po rozdelení modulu Rest rozhranie servera. Jeho funkcionalita ostala nezmenená.
Web rozhranie pre prezeranie transakcií – nový modul, ktorý bude predstavovať rozhranie pre zobrazovanie transakcií na webovej stránke. V tomto šprinte vznikla základná štruktúra, v ktorej sa budú transakcie zobrazovať.
Jadro servera pre používateľa – vznikol taktiež rozdelením komponentu jadro servera na časti. Tento komponent ostal nezmenený.
Jadro servera pre transakcie – modul vznikol taktiež rozdelením komponentu jadro servera. Tento komponent bol pridaný v tomto šprinte a obsahuje ukladanie a prácu s transakciami v jadre servera.



Obrázok 11 Architektúra systému po 4. šprinte.

Revízia 4. šprintu

Analýza typov transakcií

Oproti pôvodnému návrhu zo 4. šprintu sme pridali ešte jednu entitu *Database*, v ktorej sú uložené všetky transakcie. Nad objektom entity *Database* sme vytvorili volanie metódy *storeTransaction*, ktorá uloží danú transakciu do databázy - v tomto prípade inštanciu transakcie, ktorú vytvoril *TransactionManager* volaním metódy *new* nad triedou *Transaction*.

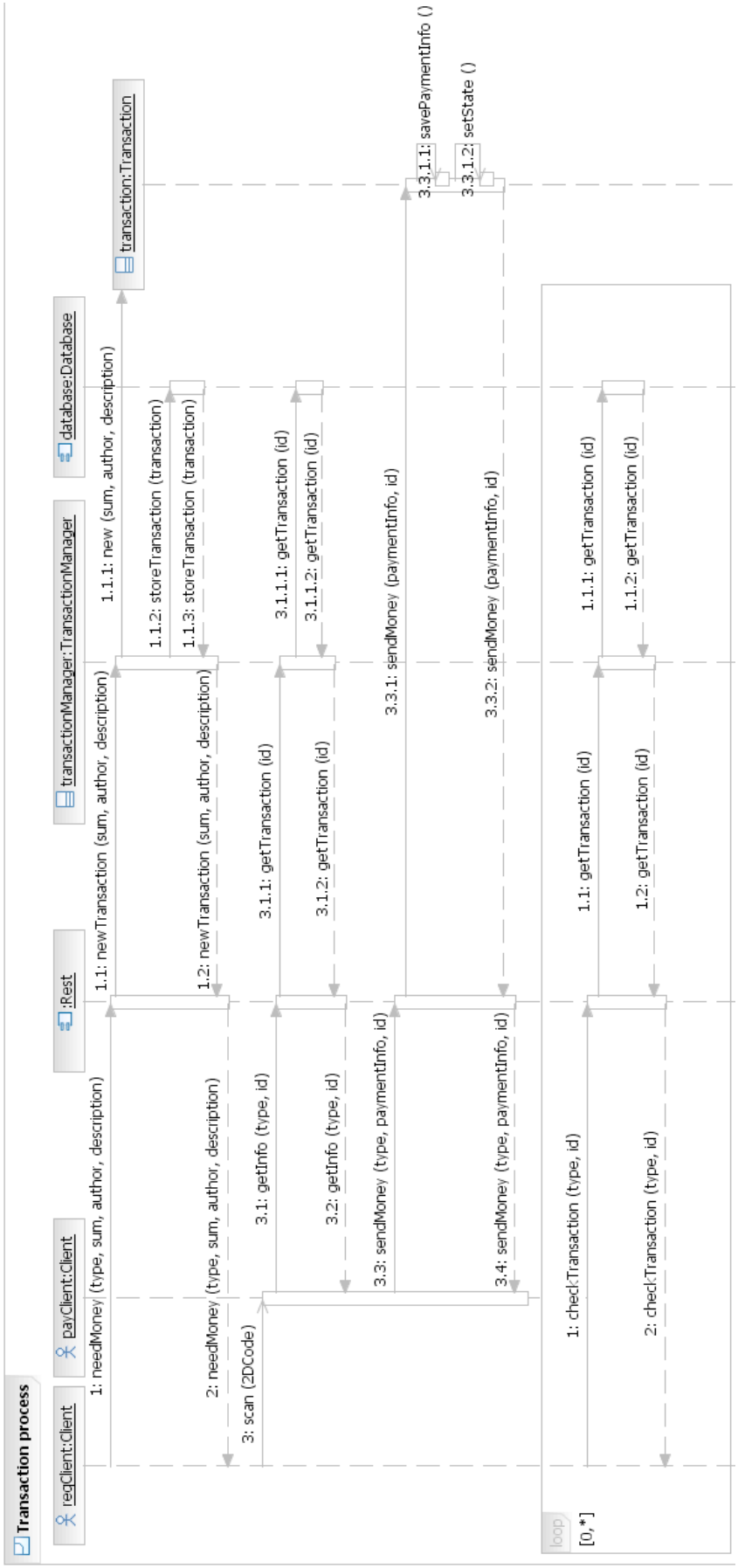
Entitu *Server* sme premenovali na *Rest*, keďže v našom prípade sa na danom mieste konkrétne používa REST architektúra.

Metódu *wantToPay* sme zrušili a namiesto nej sme navrhli vytvoriť metódu *getInfo*, ktorá vráti klientovi informácie o platbe. Túto zmenu sme urobili preto, lebo klient sa v podstate chce dozvedieť informácie o platbe. Okrem toho sa táto metóda bude dať využiť v budúcnosti aj v iných prípadoch. *Rest* získa informácie o transakcii volaním metódy *getTransaction* nad objektom *TransactionManager*-a, ktorý rovnomennou metódou získa z databázy objekt transakcie a vráti ho *Rest*-u.

Presunuli sme časť zodpovednosti z *TransactionManager*-a na *Transaction*. Dohodli sme sa, že ako trieda, ktorá obsahuje atribúty, by s nimi mala vedieť aj pracovať. Preto sa metóda *sendMoney* viac nebude nachádzať v triede *TransactionManager*, ale v triede *Transaction* a *Rest* ju odtiaľ bude volať. V tejto metóde si potom objekt triedy *Transaction* sám nastaví potrebné atribúty. Taktiež sme z volania metódy *sendMoney* odstránili argument type, pretože sme si uvedomili, že nie je potrebný, keďže sa volá konkrétna metóda triedy *Transaction*, ktorá vie, čo má robiť. Ani argument *author* nie je potrebné explicitne posielať ako argument, keďže bude obsiahnutý v argumente *paymentInfo*.

Ďalšia časť zodpovednosti, ktorú sme odstránili z triedy *TransactionManager* je tá, že pri zisťovaní stavu transakcie klientom, ktorý vyžiadal platbu, nemusí zisťovať stav transakcie, ale iba získa z databázy objekt transakcie, vráti ho *Rest*-u a on už z neho získa stav transakcie, ktorý pošle v odpovedi naspäť klientovi.

Zrevidovaný sekvenčný diagram, v ktorom sme uskutočnili zmeny je zobrazený na obrázku (Obrázok 12).



Obrázok 12 Zrevidovaný sekvenčný diagram.

REST rozhranie pre realizáciu transakcie typu NEED_MONEY

Návrh riešenia

REST
Na každú z nižšie uvedených požiadaviek je v prípade, že autentifikácia je neúspešná, nasledujúca odpoveď (neuvádzame ju pri každej požiadavke, ale spoločne na začiatku, keďže sa nemení).

Odpoveď
Nesprávne prihlasovacie údaje
HTTP kód: 401 Not Authorized
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Nasleduje opis možných požiadaviek.

Požiadavka
Vytvorenie novej transakcie typu NEED_MONEY
HTTP metóda: POST
URI: /transaction
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “NEED_MONEY”,
  “sum”: suma,
  “description”: “popis, za čo je potrebné zaplatiť”
}
```

Odpoveď
Úspešné vykonanie požiadavky, vytvorenie novej transakcie
HTTP kód: 201 Created
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
{
  “id”: identifikačné číslo vytvorenej transakcie,
  “uri”: “URI vytvorenej transakcie”
}
```

Požiadavka
Zistenie podrobných informácií o transakcii
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácie o transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “typ transakcie”,
  “sum”: suma,
  “description”: “popis, za čo je potrebné zaplatiť”
  “state”: “stav transakcie”,
  “payment”: {“payer”: “e-mail platcu”}
}
```

Poznámka: pole payment sa v odpovedi vyskytuje, iba ak je transakcia zaplatená.

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Odoslanie peňazí do transakcie
HTTP metóda: PUT
URI: /transaction/{identifikačné číslo transakcie}
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “payment”: {“sum”: suma}
}
```

Odpoveď
Platba úspešne vykonaná
HTTP kód: 200 OK
Telo: prázdne

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Platba zamietnutá – do transakcie sa nedajú odoslať peniaze.
Te transakcia už bola uzavretá, odoslaná suma sa nezhoduje s očakávanou a pod.
HTTP kód: 400 Bad Request
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Zistenie stavu transakcie
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}/state.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácia o stave transakcie
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “state”: “stav transakcie”
}
```

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Implementácia

REST
Implementácia bola opäť pomerne jednoduchá, pozostávala z nasledujúcih krokov:

- Pridanie nových ciest do smerovača.
- Vytvorenie rozhrania *NeedMoneyTransactionResource*, ktoré predstavuje zdroj reprezentujúci transakciu. Toto rozhranie implementuje trieda *ServerNeedMoneyTransactionResource*. Dôležité metódy tejto triedy sú *@Get NeedMoneyTransactionDTO retrieve()*, ktorá slúži na získanie informácií o transakcii z jadra, *@Post create(NeedMoneyTransactionDTO)*, ktorá slúži na vytvorenie novej transakcie a *@Put update(NeedMoneyTransactionDTO)*, ktorá slúži na editáciu transakcie, v našom prípade na pridanie platby do transakcie.
- Vytvorenie prenosových dátových objektov, ktoré reprezentujú informácie o transakcii a informáciu o platbe. Tieto prenosové objekty sú implementované v triedach *PaymentDTO a NeedMoneyTransactionDTO*.

Jediným zaujímavým problémom bolo, ako obmedziť vrátené informácie o transakcii iba na niektoré polia (pri získavaní stavu transakcie). Dá sa povedať, že bolo treba implementovať mechanizmus filtrovania vrátených polí. Nakoniec sme sa rozhodli použiť rovnakú metódu, ktorou sa získavajú informácie o transakciách a v kóde jednoduchou podmienkou podľa URI zistiť, či sa má vrátiť iba pole so stavom, alebo všetky polia o transakcii. Ak sa v budúcnosti opäť vyskytne potreba filtrovať vrátené výsledky, bolo by vhodné vyvinúť univerzálnejší mechanizmus.

Testovanie

REST
Testy sú automatizované a sú implementované v triede *RestNeedMoneyTransactionIntegrationTest* v balíku *sk.fiit.mpayserver.rest* v zložke *test*. Testovacie metódy pokrývajú nasledujúce správanie aplikácie:

- Aplikácia odpovedá HTTP chybovým kódom 401 pri pokuse o použitie rozhrania k transakciám neprihláseným používateľom.
- Aplikácia odpovedá HTTP chybovým kódom 400, ak je požadovaná suma v transakcii nižšia alebo rovná 0.
- Aplikácia odpovedá HTTP kódom 201, ak sa podarí úspešne vytvoriť transakciu.
- Aplikácia v odpovedi vráti identifikačné čislo transakcie a jej URI, ak sa podarí

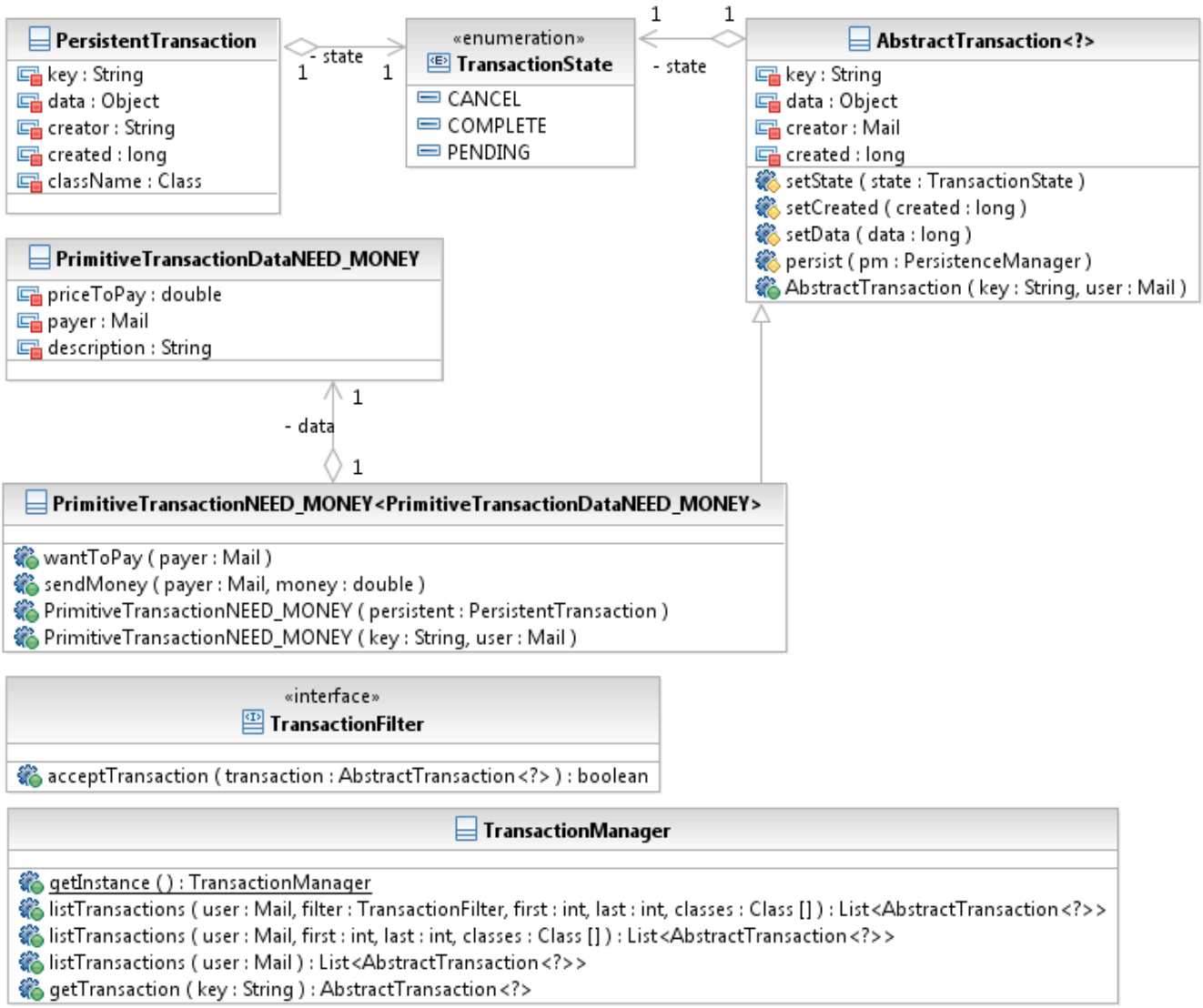
úspešne vytvoriť transakciu.

- Aplikácia vráti všetky povinné polia v transakcii pri vyžiadaní informácie o transakcii. Použije pritom HTTP kód 200.
- Aplikácia odpovedá HTTP chybovým kódom 400, ak je zadaná suma pri pokuse o platbu nižšia alebo rovná 0.
- Aplikácia odpovedá HTTP chybovým kódom 400, ak je zadaná suma pri pokuse o platbu iná ako požadovaná suma v transakcii.
- Aplikácia odpovedá HTTP kódom 200, ak sa podarí platbu úspešne spracovať.
- Aplikácia odpovedá HTTP chybovým kódom 404 vždy, keď sa klient pokúša pristupovať k transakcii s neexistujúcim identifikačným číslom.
- Aplikácia vráti stav transakcie s HTTP kódom 200.
- Aplikácia nevráti iné polia v transakcii, ak si klient vyžiada iba stav transakcie.

Analýza a návrh spracovania transakcií v jadre servera

Táto analýza vychádza z ukončenej úlohy #84 "Analýza a návrh transakcií". V úlohe sme vytvorili dokumentáciu k základnému typu transakcie „Užívateľ požaduje peniaze". V budúcnosti sa počet a typ správ bude zväčšovať vzhľadom na ďalšie typy transakcií. Správy prichádzajú na REST rozhranie kde sú volané jednotlivé funkcie nad transakciou. Všeobecné funkcie transakcie sú implementované v abstraktnej triede aby bolo možné vytvoriť nový druh transakcie len s malými úpravami. Konkrétna transakcia rozširuje abstraktnú tak, že definuje konkrétny typ dát s ktorými pracuje a tým že definuje metódy ktoré je možné volať z REST API. Tieto metódy sú paralelou k správam, preto pre SEND_MONEY existuje metóda *sendMoney* a podobne. Tieto rozširujúce metódy vykonávajú zmenu nad transakciou, preto sa v rámci ich volaní vykoná perzistencia transakcie s novým stavom. REST API môže pristupovať iba k týmto metódam a metódam ktoré nemenia stav transakcie. Preto sú všetky metódy setXXX prístupového charakteru *protected*.

Pre prácu nad transakciou je potrebné získať jej inštanciu pomocou metódy *getTransaction(String key)* triedy *TransactionManager*. Pre získanie transakcie musí byť užívateľ prihlásený v systéme. Táto trieda poskytuje aj možnosť získať zoznam transakcií daného užívateľa, a vykonávať stránkovanie nad týmito transakciami. V budúcnosti sa očakáva aj implementácia filtrovania transakcií. Pre perzistenciu transakcií sa využíva samostatná trieda *PersistentTransaction, ktorá obsahuje základné polia transakcie*. Pri načítavaní transakcie z úložiska sa využíva reflektívne API, a serializácia dát transakcie. Schému tried môžete vidieť na obrázku (Obrázok 13).



Obrázok 13 Diagram transakcií.

Šprint č. 5

Úlohou 5. šprintu je implementovať jednu z identifikovaných transakcií – používateľ si chce od iného používateľa vyžiadať peniaze. Z predchádzajúceho šprintu je vytvorený podklad pre túto transakciu v jadre servera, kde je možné umiestňovať transakcie a pracovať s nimi. V tomto šprinte je potrebné implementovať vytváranie transakcií na mobilnom zariadení a posielať informácie o nich na server, kde sa budú ukladať. Taktiež je potrebné implementovať zobrazovanie týchto transakcií na webovej stránke.

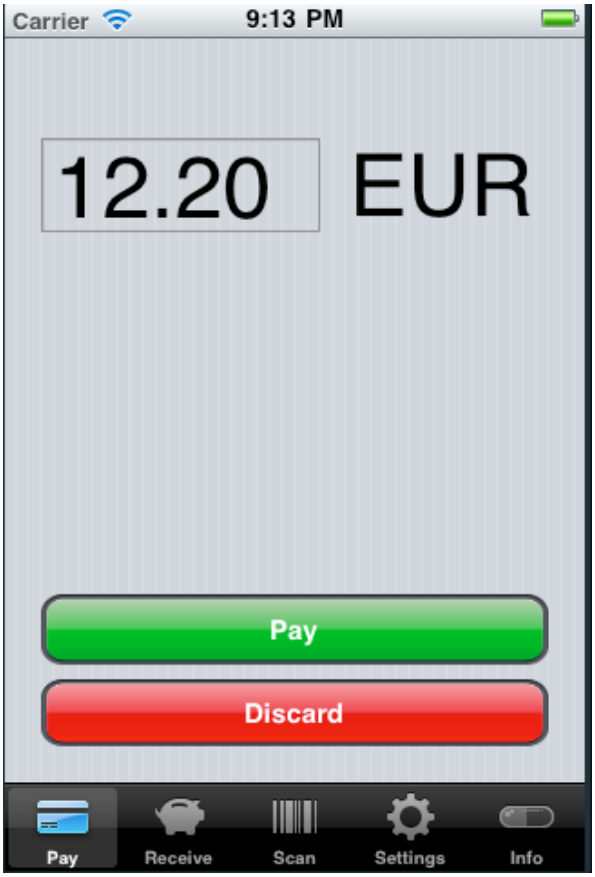
V rámci tohto šprintu sme identifikovali nasledovné príbehy:

1. Návrh grafického rozhrania pre obrazovky na iPhone
2. Realizácia platby iniciovanej príjemcom – žiadosť o platbu
3. Realizácia platby iniciovanej príjemcom – prijatie požiadavky
4. Realizácia platby iniciovanej príjemcom – zistenie stavu
5. Zobrazovanie zrealizovaných platieb na webovej stránke

Návrh grafického rozhrania pre obrazovky na iPhone

Obrazovka Pay

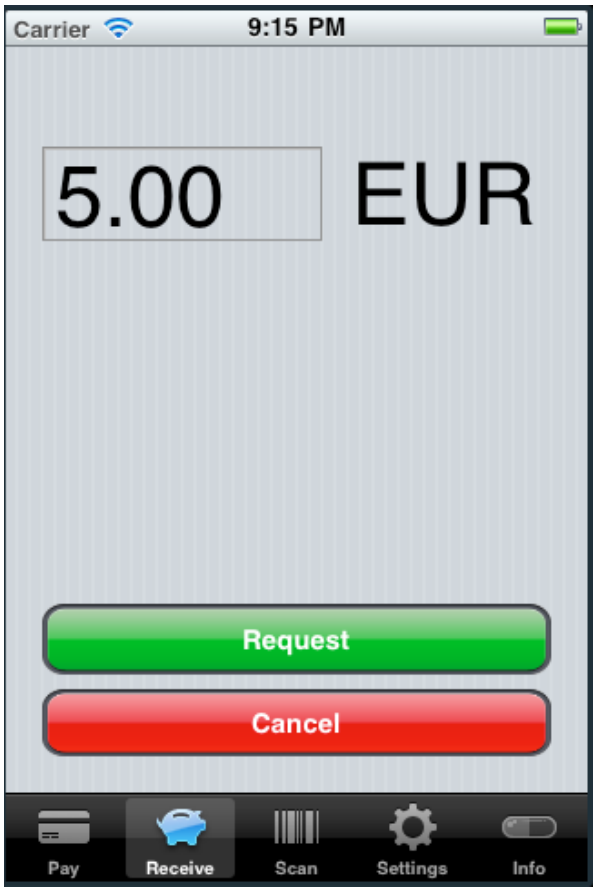
Obrazovka slúži na realizovanie požiadavky na platbu (Obrázok 14). Používateľ má možnosť zadať sumu, ktorú je ochotný zaplatiť. V prípade, že sa rozhodne platbu pred jej vykonaním zamietnuť, môže tak vykonať stlačením tlačidla „Discard“. Po potvrdení operácie platby tlačidlom „Pay“, dôjde k umiestneniu informácie o platbe do cloudu a je prezentovaná obrazovka s 2D kódom.



Obrázok 14 Obrazovka realizujúca platbu.

Obrazovka Receive

Obrazovka slúži na vytvorenie požiadavky na prijatie platby (Obrázok 15). Používateľ má možnosť zadať sumu, ktorú vyžaduje a chce prijať. Po zamietnutí sumy tlačidlom „Cancel“, dôjde k vynulovaniu zadanej čiastky. Po potvrdení zadanej sumy tlačidlom „Request“, dôjde k umiestneniu požiadavky do cloudu a je prezentovaná obrazovka s 2D kódom.



Obrázok 15 Obrazovka, realizujúca úrijatie špecifickej sumy.

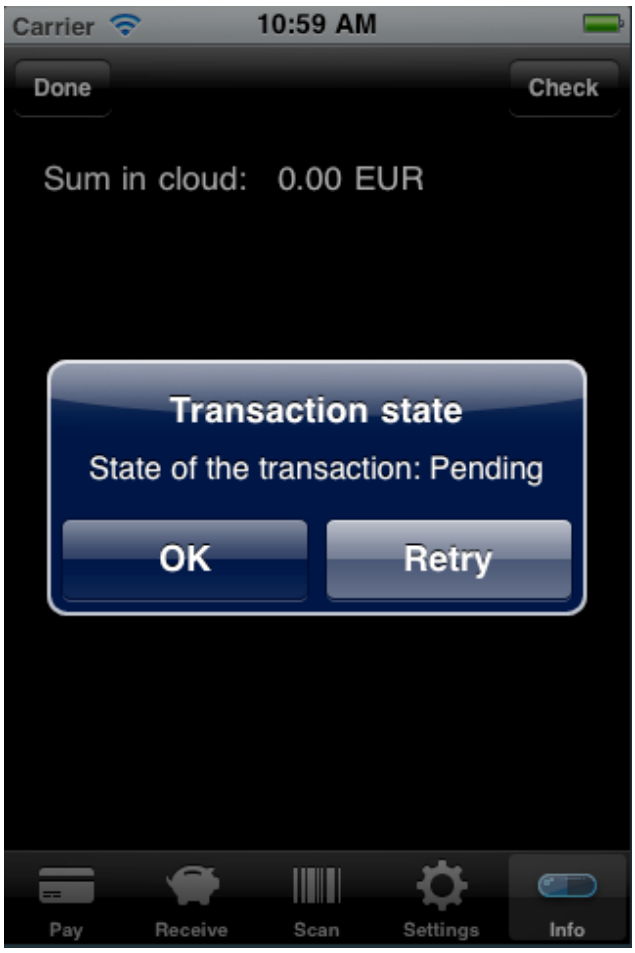
Obrazovka zobrazenia 2D kódu

Obrazovka slúži na zobrazenie 2D kódu so zakódovanou informáciou (Obrázok 16). Obrazovka sa prezentuje po vykonaní požiadavky na platbu alebo požiadavky na prijatie platby. Zobrazuje informáciu o sume, ktorá je v cloude uložená, prípadne sumu, ktorú používateľ vyžaduje.

Obrázok 16 Obrazovka s 2D kódom.

Správa o stave transakcie

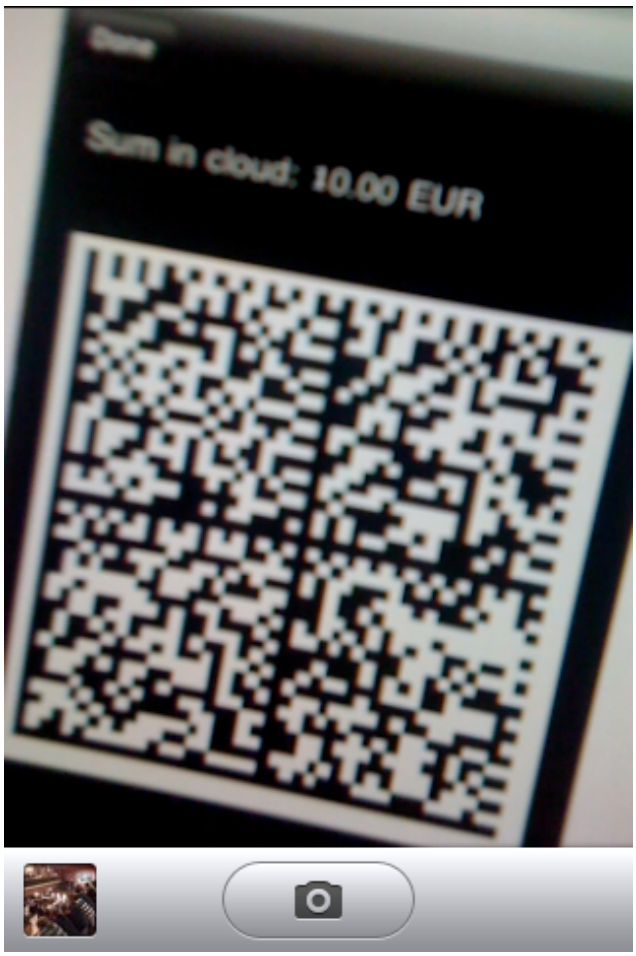
Po kliknutí na tlačidlo "Check", ktoré sa nachádza na obrazovke pre zobrazenie 2D kódu sa zobrazí informačné okno so správou o stave transakcie ako je znázornené na obrázku (Obrázok 17). Toto okno bude možné zavrieť stlačením tlačidla "OK". Tlačidlo "Retry" bude slúžiť na opätovné zistenie stavu.



Obrázok 17 Správa o stave transakcie.

Obrazovka Scan

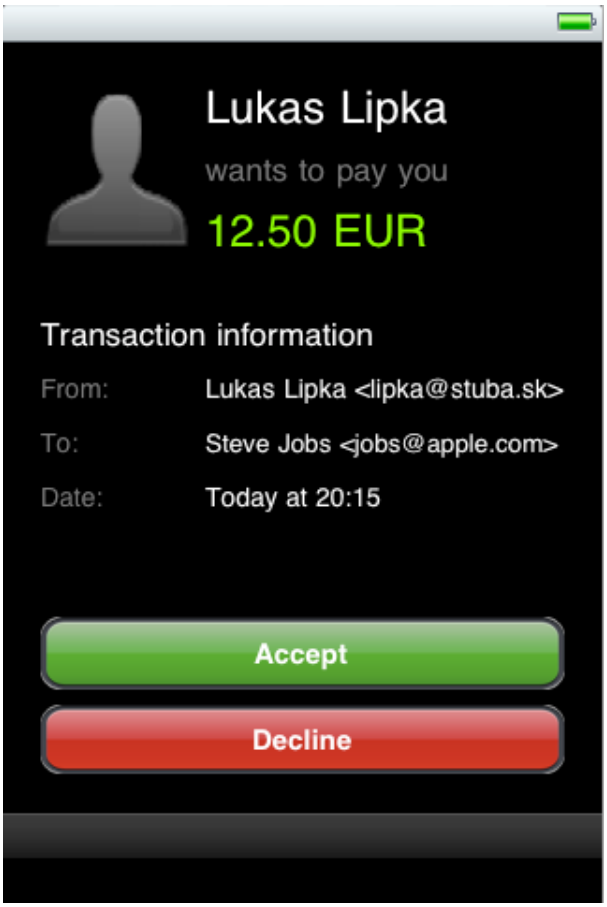
Obrazovka slúži na zosnímanie vygenerovaného kódu z iného zariadenia (Obrázok 18). Obsahuje obraz z hľadáčiku fotoaparáta a tlačidlo na zosnímanie daného obrazu. Po zacentrovaní 2D kódu a stlačení tlačidla zosnímania, dôjde k zachyteniu a dekódovaniu 2D kódu. Na základe informácie uloženej v 2D kóde – či sa jedná o prijatie alebo odoslanie platby – dôjde k prezentovaniu obrazovky na odpoveď k danej požiadavke.



Obrázok 18 Obrazovka na skenovanie 2D kódu.

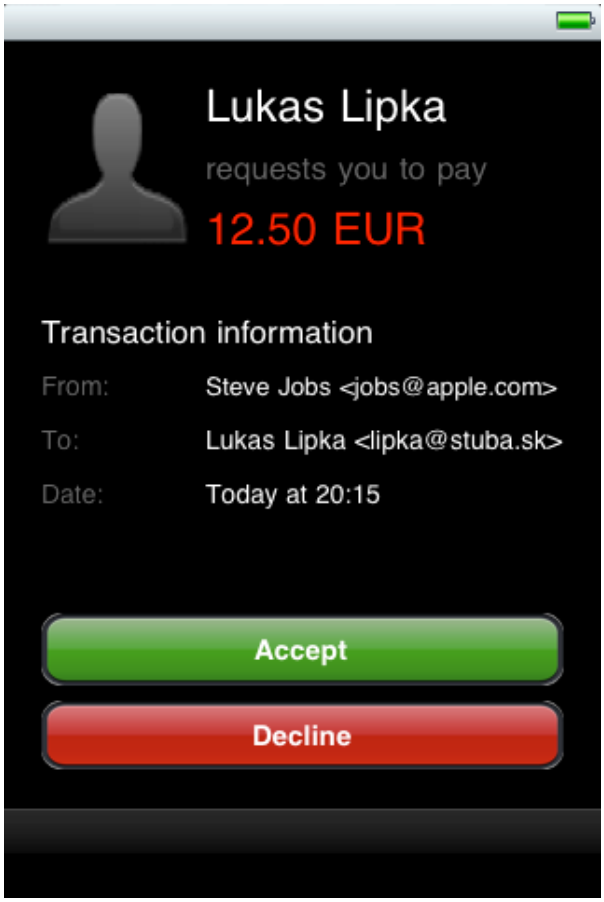
Obrazovka Response

Obrazovka je prezentovaná používateľovi po odskenovaní 2D kódu. Na základe typu požiadavky – prijatie (Obrázok 19) alebo odoslanie platby (Obrázok 20) – sa menia aj informácie zobrazené na obrazovke. O type operácie, ktorá sa ide vykonať, informuje používateľa text pod menom osoby, ktorá operáciu inicializovala. V prípade transakcie prijatia platby je suma zobrazená zelenou farbou, aby vizuálne indikovali používateľovi, že sa jedná o operáciu prijatia platby.



Obrázok 19. Obrazovka pre prijatie platby

V prípade operácie odoslania platby je suma zobrazená červenou farbou, ako vizuálny indikátor odoslania finančnej čiastky.

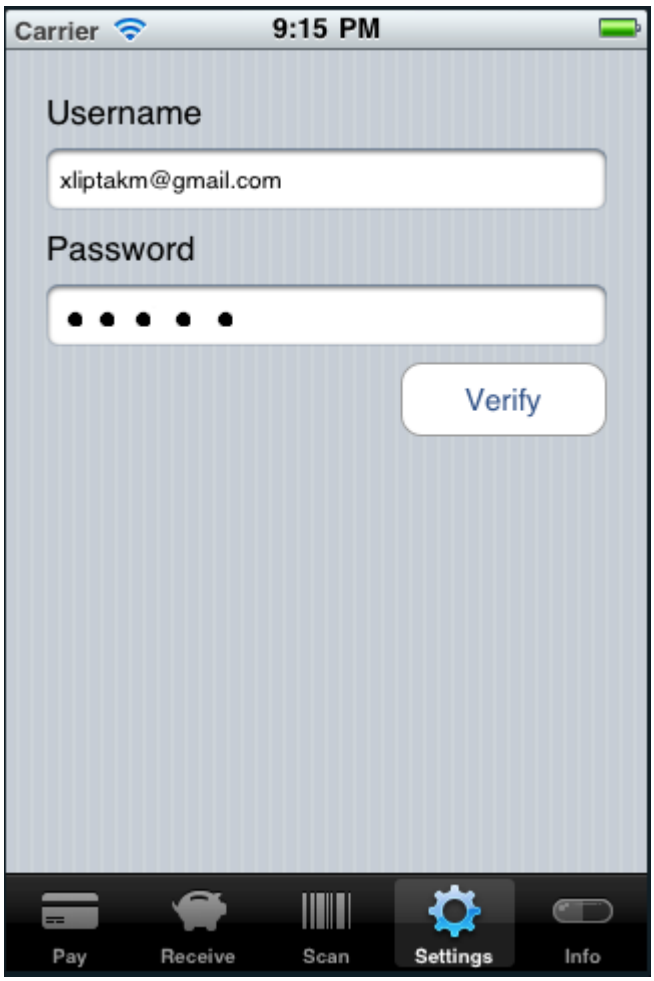


Obrázok 20 Obrazovka pre vykonanie platby.

Operácie sa potvrdzujú resp. rušia dvojicou tlačidiel, ktoré taktiež využívajú zelenú a červenú farbu ako indikátor danej akcie.

Obrazovka s nastaveniami

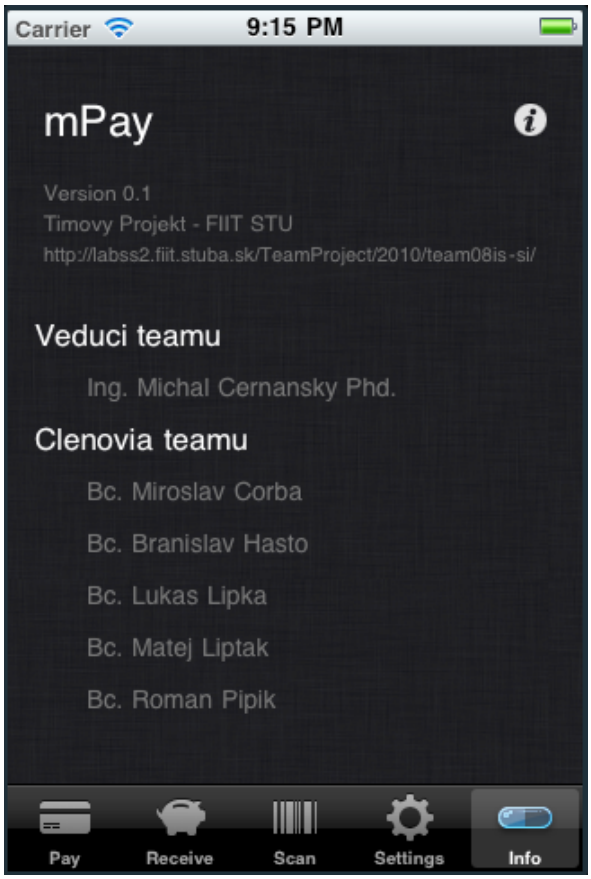
Obrazovka slúži na zmenu nastavení o účte používateľa (Obrázok 21). Zadávajú sa do nej prihlasovacie údaje – používateľské meno a heslo. Údaje sa ukladajú ihneď po zadaní. Tlačidlo „Verify“ slúži na overenie korektnosti zadaných údajov.



Obrázok 21 Obrazovka s nastaveniami účtu používateľa.

Obrazovka Info

Obrazovka obsahuje informácie o aplikácii, jej verzii a autoroch. (Obrázok 22)



Obrázok 22 Obrazovka, zobrazujúca info o aplikácii.

Realizácia platby iniciovanej príjemcom – žiadosť o platbu

Používateľ chce od iného používateľa získať peniaze. Vloží požadovnú sumu a vygeneruje sa mu 2D kód s ID transakcie.

Analýza

Je potrebné implementovať funkcionalitu na iniciovanie a vykonanie platby na základe požiadavky zo strany príjemcu. Po umiestnení informácie o platbe na server je potrebné prezentovať 2D kód, ktorý môže odsnímať iné zariadenie a dokončiť transakciu.

Návrh

Žiadosť o platbu je typ operácie, ktorá je iniciovaná príjemcom platby. Po vybratí tejto operácie, je používateľovi prezentovaná obrazovka, do ktorej zadá požadovanú sumu, ktorú chce prijať. Následne má používateľ možnosť túto sumu potvrdiť resp. zamietnuť.
Po potvrdení sumy, sa odošle požiadavka na server s informáciami o používateľovi, typom platby a sumou, ktorú je ochotný prijať. Server na túto požiadavku pošle odpoveď o úspešnosti operácie. V prípade, že bola operácia úspešná, je v správe zahrnutý aj

identifikátor platby, pod ktorým je na serveri uložená. Po prijatí tohto identifikátora, dochádza na klientskej strane k jeho zakódovaniu do 2D kódu a prezentovaniu na obrazovku zariadenia. Po zobrazení 2D kódu na zariadení, ho je možné zosnímať iným zariadením a pokračovať v transakcii zvoleného typu.

Implementácia

Odoslanie požiadavky na platbu bolo implementované vytvorením obrazovky pre túto špecifickú operáciu. Táto obrazovka sa nazýva *PaymentRequestView*. Implementuje funkcionalitu na zadanie sumy a tlačidlá na jej potvrdenie resp. zamietnutie. Pri zadávaní sumy je používateľovi prezentovaná na obrazovke zariadenia klávesnica výlučne s numerickými znakmi a interpunkciou.
Po zamietnutí zadanej sumy tlačidlom „Discard“, dochádza k jej vynulovaniu. Ďalej má používateľ možnosť zadať novú sumu, alebo prejsť na inú obrazovku.
Po potvrdení zadanej sumy tlačidlom „Pay“, sa vytvorí správa s požiadavkou typu „payment request“ a dochádza k jej odoslaniu na server prostredníctvom triedy *MPTransport*. Tá slúži ako rozhranie pre komunikáciu so serverom. Trieda taktiež zabezpečuje autentifikáciu používateľa voči serveru.
Server na danú požiadavku odpovie správou o úspešnosti operácie. V prípade úspešnosti spracovania požiadavky, bude v odpovedi zahrnuté ID transakcie. Toto ID transakcie sa ďalej použije na vygenerovanie 2D kódu. Ten sa prezentuje zobrazením obrazovky *DataMatrixView*.

Realizácia platby inicovanej príjemcom – prijatie požiadavky

Používateľ zosníma 2D kód a zobrazí sa mu informácia o transakcii, ktorú môže na základe kódu vykonať.

Analýza

Po zosnímaní 2D kódu je potrebné zobraziť informácie o transakcii, ktorá je na serveri uložená pod identifikátorom zakódovaným v 2D kóde.

Návrh

Používateľ zosníma 2D kód zobrazený na obrazovke iného zariadenia. Následne dochádza k jeho dekódovaniu. Na základe identifikátora uloženého v 2D kóde sa na server pošle požiadavka, na získanie informácie o transakcii. Podľa typu informácie v odpovedi zo servera sa používateľovi prezentuje obrazovka, podľa typu transakcie – prijatie platby alebo odoslanie platby. Obrazovka by mala používateľovi vizuálne indikovať, o aký typ transakcie sa jedná, aby nedošlo k ich zameneniu a tým aj k nechcenému odoslaniu finančnej čiastky. Používateľ ma následne možnosť, daný typ transakcie prijať resp. zamietnuť.

Implementácia

Získanie informácií bolo implementované prostredníctvom odoslania požiadavky na server. Požiadavka sa posiela prostredníctvom triedy *MPTransport*. Trieda taktiež zabezpečuje autentifikáciu používateľa voči serveru. Informácia o platbe obsahuje údaje o type transakcie, výške finančnej čiastky, informácie o používateľov, ktorý transakciu inicioval, dátum transakcie.
Po získaní informácie, sa na základe typu transakcie – prijatie alebo odoslanie - prezentuje príslušná obrazovka. Obrazovka bola implementovaná prostredníctvom zobrazenia *PaymentResponseView*. Obrazovka okrem typu operácie a výške finančnej čiastky, obsahuje fotografiu používateľa, stručné informácie o iniciátorovi platby a dátum jej vytvorenia. Obrazovka obsahuje vizuálne farebné indikátory, aby nedošlo k zameneniu spomínaných

dvoch typov transakcií. Používateľ môže na obrazovke vykonať dve operácie a to prijatie, alebo zamietnutie platby.

Časť potvrdenie sumy

Analýza

Keďže vykonávame operácie s platbami a finančnými čiastkami, je potrebné zabezpečiť aby používateľ nevykonal žiadnu z operácií nechcene resp. omylom.

Návrh

Na zamedzenie vykonávania chybných operácií, je potrebná vizuálna indikácia o type operácie, ktorá sa ide vykonať. Na túto indikáciu je vhodné použiť farebné rozlíšenie textu so sumou:

- Zelenou farbou sa zobrazí suma, ktorá bude prijatá po potvrdení operácie
- Červenou farbou sa zobrazí suma, ktorá bude používateľovi odrátaná po potvrdení operácie

Analogicky budú zafarbené aj tlačidlá na prijatie a odmietnutie požiadavky.
Po stlačení tlačidla potvrdenia sa môže prezentovať dialógové okno, ktoré vyzve používateľa potvrdiť, či chce skutočne vybratú operáciu vykonať.
Jedným z ďalších ochranných prvkov by mohla byť nutnosť podržať tlačidlo potvrdenia na určitý čas.

Implementácia

V obrazovke *PaymentResponseView* bolo implementované farebné rozlišovanie sumy pri prijímaní a odosielaní finančnej čiastky. Ďalším pridaným indikátorom boli nasledujúce variácie textov:

- *Wants to send you,* pre operáciu prijatie finančnej čiastky
- *Wants you to pay,* pre operáciu odoslania finančnej čiastky

Tlačidlá pre potvrdenie a zamietnutie transakcie takisto obsahujú farebný indikátor. Pre potvrdenie operácie sa využíva zelené tlačidlo „Accept“, pre zamietnutie červené tlačidlo s textom „Decline“.

Zobrazovanie zrealizovaných transakcií

Ako používateľ systému si chcem na webe zobraziť prehľad svojich transakcií, aby som vedel, aké transakcie som vykonal.

Analýza požiadaviek

Jadro servera
V jadre servera je potrebné vytvoriť mechanizmus, ktorý získa zoznam transakcií pre aktuálne prihláseného používateľa. Tento mechanizmus bol už implementovaný v predchádzajúcom šprinte.

REST
REST rozhranie v tomto prípade musí byť schopné:

- Poskytnúť zoznam transakcií pre prihláseného používateľa.
- Nesmie používateľovi poskytnúť údaje o transakciách iného používateľa.
- V odpovedi sa musia zahrnúť transakcie, ktoré používateľ inicioval, ako aj transakcie,

do ktorých sa používateľ iba zapojil.

Podobné bezpečnostné požiadavky sme už riešili v prípade používateľského príbehu Zobrazenie informácií o prihlásenom používateľovi. Preto by bolo vhodné aj bezpečnosť v tejto úlohe riešiť podobným spôsobom.

Webová stránka
Na webovej stránke je potrebné získať z Rest rozhrania zoznam transakcií a tie zobraziť v už pripravenom používateľskom rozhraní.

Návrh riešenia

REST
Bezpečnostné požiadavky budeme riešiť rovnako ako v prípade príbehu Zobrazenie informácií o prihlásenom používateľovi. REST rozhranie navrhneme tak, aby ani neposkytovalo možnosť vrátiť údaje o transakciách iného používateľa. O tom, pre ktorého používateľa sa zobrazí zoznam transakcií, rozhodneme z hlavičiek požiadavky na server.

Nasleduje opis navrhnutých požiadaviek.

Požiadavka
Získanie zoznamu transakcií
HTTP metóda: GET
URI: /transactions.json
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: prázdne

Odpoveď
Nesprávne prihlasovacie údaje
HTTP kód: 401 Not Authorized
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Odpoveď
Zoznam transakcií
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
[
  {
    “id”: identifikačné čislo transakcie,
    “author”: “e-mail iniciátora transakcie”,
    “type”: “typ transakcie”,
    “sum”: suma,
    “state”: “stav transakcie”,
    “payment”: {“payer”: “e-mail platcu”}
  },

  {
    “id”: identifikačné čislo transakcie,
    “author”: “e-mail iniciátora transakcie”,
    “type”: “typ transakcie”,
```

```
    "sum": suma,
    "state": "stav transakcie",
    "payment": {"payer": "e-mail platcu"}
  },

  ...
]
```

Webová stránka

Používateľské rozhranie pre webovú stránku bolo vytvorené už v predchádzajúcom šprinte. Tento šprint slúži na implementáciu jej funkcionality. Na získavanie transakcií zo servera použijeme http metódu *Get,* ktorá vráti transakcie, ktoré sú uložené na serveri pre prihláseného používateľa. Následne vytvorí nový záznam v používateľskom rozhraní, kde sa zobrazia informácie o získanej transakcii.

Implementácia

REST
Implementácia pozostávala z nasledujúcih krokov:

- Pridanie novej cesty *„/transactions"* do smerovača.
- Vytvorenie rozhrania *TransactionsListResource*, ktoré predstavuje zdroj reprezentujúci zoznam transakcií. Toto rozhranie implementuje trieda *ServerTransactionsListResource*. Jediná metóda tejto triedy je *@Get List<NeedMoneyTransactionDTO> retrieve()*, ktorá slúži na získanie zoznamu transakcií z jadra.
- Prenosové dátové objekty nebolo treba vytvárať a použili sme už vytvorené objekty *NeedMoneyTransactionDTO*.

Webová stránka

Implementácia spočíva v implementovaní rozhrania *TransactionsListResource,* najmä v implementovaní http metódy *@Get retrieve(),* ktorá vráti zoznam transakcií pre prihláseného používateľa. Postupným prechádzaním cez zoznam sa vložia jednotlivé informácie o transakciách do používateľského rozhrania. Bude to realizované napĺňaním *smartGWT* komponentu *Section,* v ktorom sa bude daná transakcia rozklikávať a zobrazovať deteilnejšie.

Testovanie

REST
Testy sú automatizované a sú implementované v triede *RestTransactionsListIntegrationTest* v balíku *sk.fiit.mpayserver.rest* v zložke *test*. Testovacie metódy pokrývajú nasledujúce správanie aplikácie:

- Aplikácia odpovedá HTTP chybovým kódom 401 pri pokuse o použitie rozhrania k zoznamu transakcií neprihláseným používateľom.
- Aplikácia pri zadaní správnych prihlasovacích údajov vráti zoznam s HTTP kódom 200.
- Aplikácia do zoznamu zahrnie transakcie, ktoré používateľ inicioval.
- Aplikácia do zoznamu zahrnie transakcie, v ktorých používateľ zaplatil.

- Aplikácia do zoznamu nezahrnie transakcie, kde sa používateľ nezúčastnil.
- Aplikácia vráti všetky polia v transakcií (autor, identifikátor, typ, suma, popis, stav, platba).

Zhodnotenie 5. šprintu

V tomto šprinte sa nám podarilo vytvoriť funkčný prototyp, ktorý umožňuje realizáciu transakcií. Do systému sme doimplementovali transakcie, ktoré možno vytvárať a pracovať s nimi na mobilnom zariadení. Taktiež bola implementovaná webová stránka, kde sa jednotlivé transakcie zobrazujú pre aktuálne prihláseného používateľa.

Architektúra systému sa oproti predchádzajúcemu šprintu nezmenila. Zmenili sa len jednotlivé komponenty.

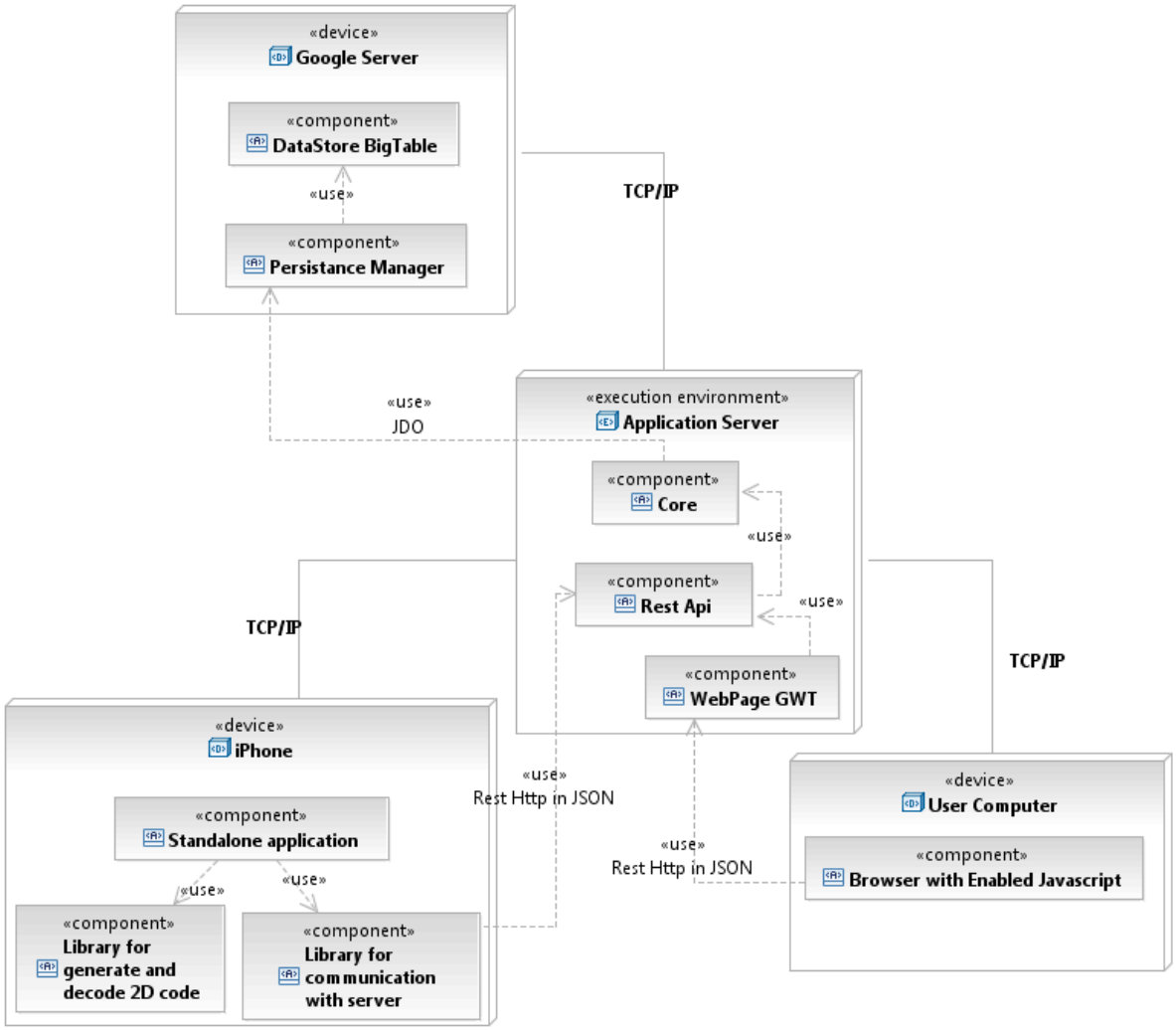
Rozhranie pre tvorbu transakcií – pre tento komponent vzniklo viacero nových obrazoviek, pomocou ktorých je možné vytvárať a pracovať s transakciami.

Web rozhranie pre prezeranie transakcií – bola vytvorená funkcionalita pre zobrazovanie transakcií na webovej stránke.

Rest rozhranie servera pre transakcie – bolo umožnené získavať zoznamy transakcií z databázy.

Zhodnotenie 1. semestra

Po 1. semestri sa nám podarilo vytvoriť funkčný prototyp, pomocou ktorého je možné realizovať jeden typ transakcií pomocou mobilných zariadení. Ide o typ transakcie, pri ktorom jeden z používateľov vytvorí požiadavku, že chce od iného používateľa získať peniaze. Na základe tejto požiadavky sa na serveri vytvorí transakcia, ktorá bude uložená pod určitým ID. Toto ID sa neskôr zakóduje do 2D kódu, ktorý zosníma iný používateľ. Po jeho dekódovaní druhý používateľ vidí, o akú transakciu ide. Na základe toho sa rozhodne, či transakciu uskutoční zaplatením požadovanej sumy, alebo sa rozhodne transakciu zrušiť. Tento prototyp sme nasadili aj na reálne zariadenia iPhone a skúšali jeho funkcionalitu. Výsledok nasadenia systému je možné vidieť na obrázku (Obrázok 23).



Obrázok 23 Diagram nasadenia pre vytvorený systém

Opis jednotlivých zariadení a komponentov:
Google server – toto zariadenie je využívané technológiou Google App Engine, ktorú používame. Komponenty, ktoré sú potrebné pre náš systém sú google databáza DataStore, ktorú využívame na ukladanie používateľov a transakcií. K databáze je možné pristupovať len cez *PersistanceManager*-a, ktorý pracuje priamo nad databázou a ukladá do nej perzistentné dáta.
Application server – ide vlastne o server, na ktorom sa vykonávajú všetky operácie serverovej časti. Sú v ňom komponenty Jadro servera (*Core*), ktoré pracuje nad databázou a umožňuje

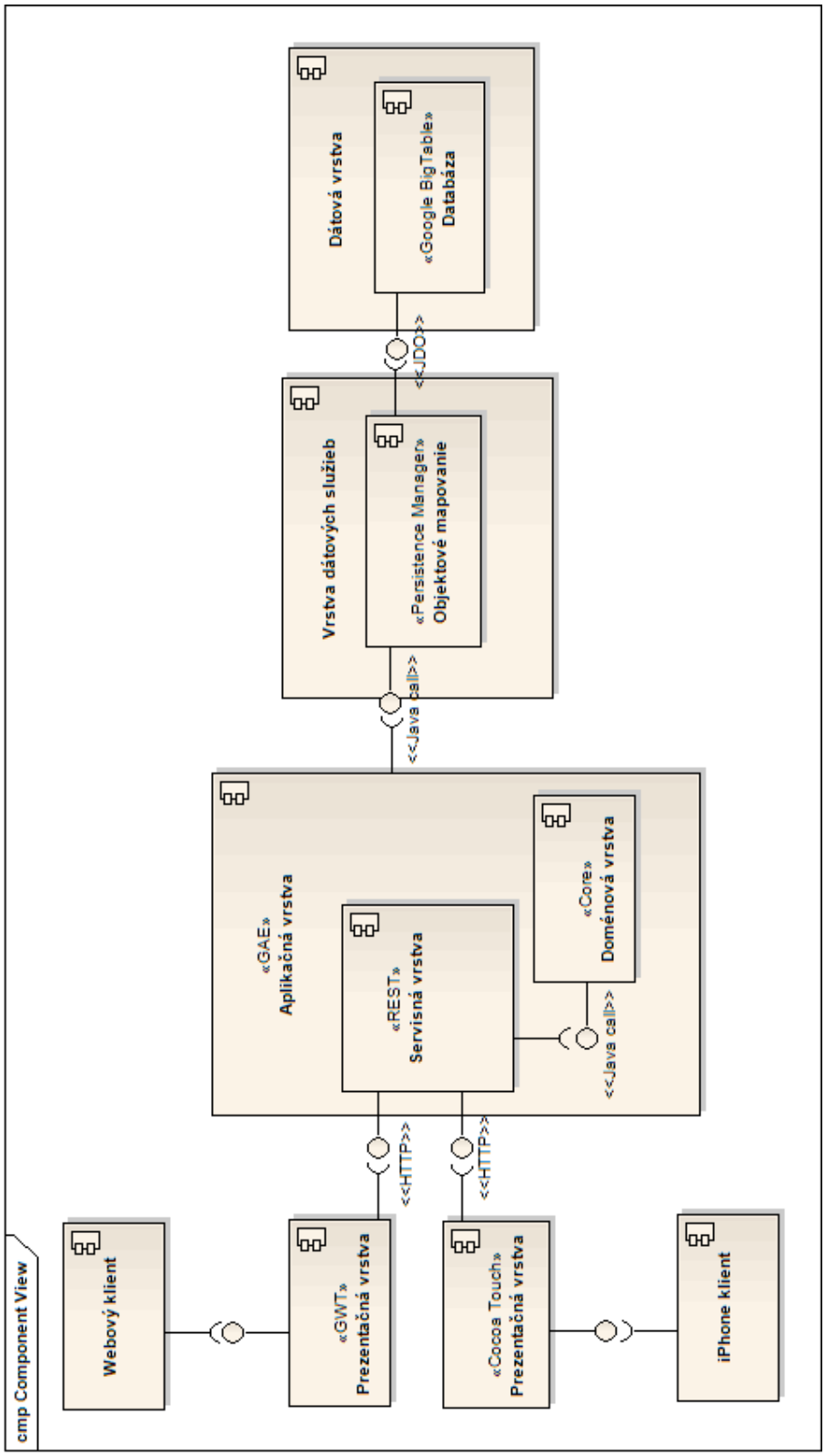
pracovať s dátami, uloženými v databáze, komponent Rest Api, ktoré umožňuje komunikáciu medzi serverom a klienstkymi aplikáciami a komponent webová stránka (*WebPage GWT*), ktorá predstavuje implementáciu klientskej webovej stránky. Aplikačný server využíva na komunikáciu s Google Serverom TCP/IP protokol. Komponent Jadro servera používa pre ukladanie do databázy JDO objekty, ktoré sú ukladané do databázy. Komponent Rest Api priamo využíva služby Jadra servera a komponet Webová stránka využíva Rest rozhranie komponentu Rest Api.

iPhone **–** mobilné zariadenie, na ktorom sa vykonáva realizácia transakcií. Využíva pri tom komponenty *Standalone application,* v ktorej je možné vytvárať a pracovať s transakciami, kompnent Knižnica pre generovanie a dekódovanie 2D kódu, pomocou ktorej sa vytvára 2D kód pri vytvorení transakcie a dekóduje po jeho zosnímaní a komponent Knižnica pre komunikáciu so serverom, kde sú využívané najmä JSON objekty. *Standalone application* využíva ako knižnicu pre generovanie a dekódovanie 2D kódu, tak aj knižnicu pre komunikáciu so serverom. Knižnica pre komunikáciu so serverom využíva pre komunikáciu so serverom Rest architektúru a komunikuje pomocou JSON objektov. iPhone vo všeobecnosti používa pre komunikáciu s aplikačným serverom TCP/IP spojenie.

User computer **–** ide o zariadenie, na ktorom beží webová stránka používateľa. Využíva jeden komponent – Prehliadač, v ktorom musí byť povolený *Javascript.* Ten komunikuje s komponentom Webová stránka pomocou Rest architektúry a JSON objektov. Zariadenie *User computer* komunikuje s aplikačným serverom pomocou TCP/IP komunikácie.

Revízia po 1. semestri

Po 1. semestri, v ktorom architektonický model nezodpovedal skutočnej archtektúre, sme sa ho rozhodli zmeniť nasledovne (Obrázok 24).



Obrázok 24 Architektúra systému po revízii

Opis architektúry systému

Ako architektonický štýl sme zvolili vrstvový štýl. Systém je zložený z viacerých vrstiev. Webový klient alebo iPhone klient komunikujú so systémom pomocou prezentačnej vrstvy. Na prezentačnej vrstve webového klienta beží JavaScript, generovaný pomocou GWT. Prezentačná vrstva iPhone klienta je naopak vytváraná pomocou frameworku Cocoa Touch.
Na serverovej strane je potom aplikačná vrstva, zabezpečovaná serverovým riešením Google App Engine. Táto vrstva je rozdelená na servisnú vrstvu, ktorá je zabezpečovaná REST-ovými webovými službami a na doménovú vrstvu, ktorá tvorí jadro systému. Prezentačné vrstvy jednotlivých klientov komunikujú so servisnou vrstvou pomocou protokolu HTTP výmieňaním HTTP request-ov a response-ov.

Servisná vrstva a doménová vrstva vzájomne komunikujú jednoduchými Java volaniami. Na ďalšej úrovni je vrstva dátových služieb, ktorá zabezpečuje objektové mapovanie na objekty v databáze. Ako objektový mapovač slúži Persistance Manager, poskytovaný GAE. S aplikačnou vrstvou komunikuje pomocou Java volaní. Poslednou vrstvou je dátová vrstva, na ktorej je databáza. Databáza je vytvorená v rámci cloud databázy Google Big Table. Do tejto objektovej databázy sú vkladené objedkty vo forme JDO.

Šprint č. 6

V tomto šprinte sme sa rozhodli začať s analýzou a implementáciou nového typu transakcie. Taktiež sme si stanovili cieľ dokončiť úprava grafického rozhrania stránky, kde chceme prejsť na používanie knižnice SmartGwt. V rámci šprintu je taktiež potrebné dotiahnuť už naimplementovany typ transakcie. Veľká časť šprintu bude venovaná potrebných dokumentov pre IIT.SRC.

Analýza druhého typu transakcie

Analýza

V rámci rozširovania aplikácie aj na ďalšie možnosti použitia pri simulovaní mobilného bankovníctva sme sa rozhodli naimplementovať ďalší scenár transakcie. Ide o 3. scenár transakcie, ktorý sme identifikovali v 4. šprinte, pri ktorom prvý klient zadá maximálnu sumu, ktorú je ochotný zaplatiť a vygeneruje 2D kód. Druhý klient požaduje určitú sumu, zosníma 2D kód a suma sa okamžite zaplatí, ak neprevyšuje zadanú maximálnu sumu.

Návrh

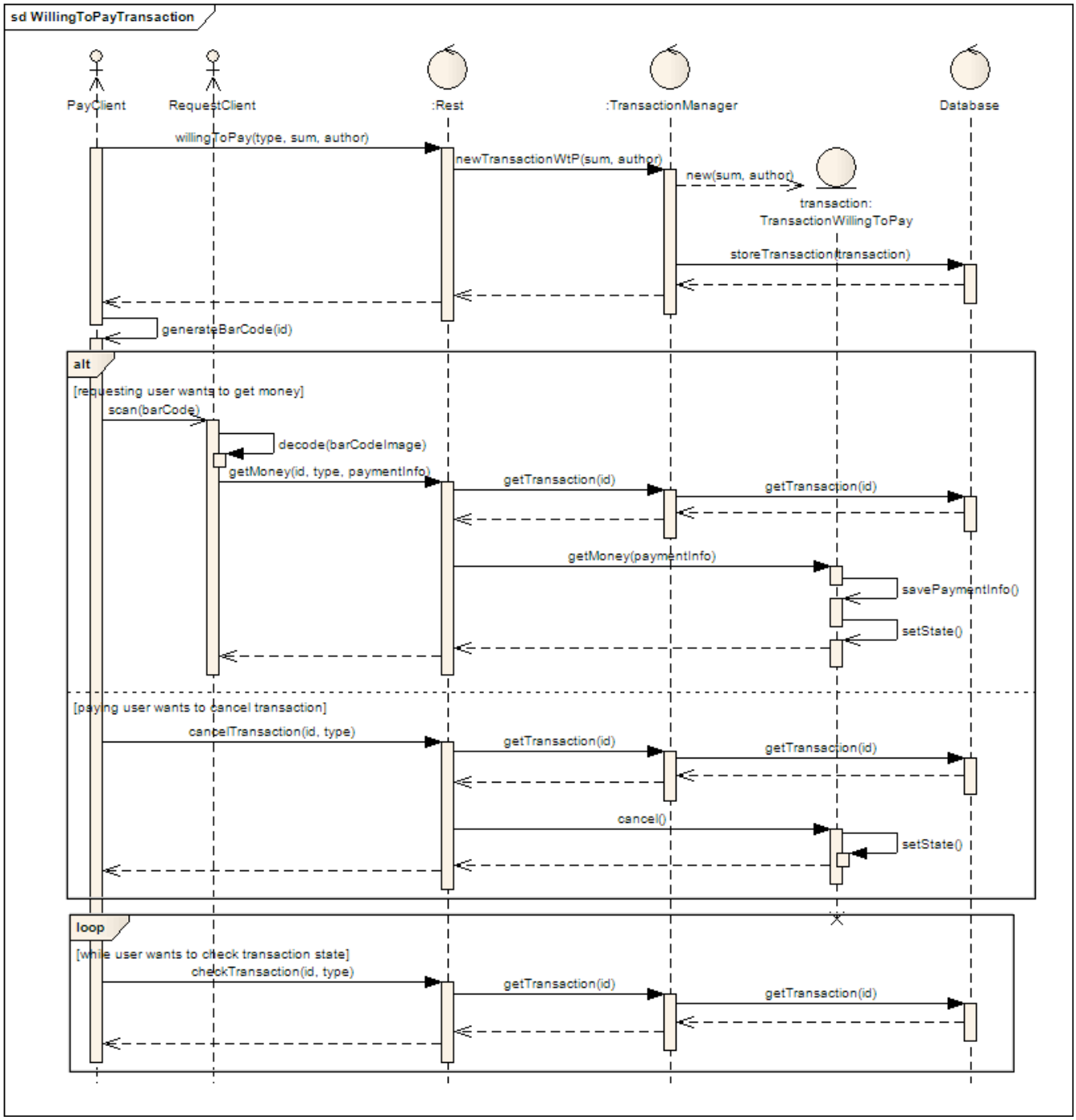
Túto transakciu sme nazvali Transakcia WillingToPay, keďže ide o transakciu, ktorú iniciuje platca, ktorý je ochotný zaplatiť určitú, maximálnu sumu. Priebeh tejto transakcie je zobrazený na obrázku (Obrázok 25).

Používateľ, ktorý sa chystá niekde platiť, si nastaví na svojom zariadení maximálnu sumu, ktorú je ochotný zaplatiť a pošle na Rest správu (požiadavku), že pri najbližšej platbe je ochotný zaplatiť zadanú sumu. V tejto správe bude obsiahnutá identifikácia autora správy, suma peňazí a typ správy, ktorý bude v tomto prípade "WILLING_TO_PAY". Potom Rest zavolá nad TransactionManager-om metódu newTransactionWtP s parametrami suma a autor, v ktorej TransactionManager vytvorí novú transakciu so sumou, autorom a stavom "PENDING" pod identifikátorom ID. Autor správy bude v tomto prípade platca, čiže aj autor platby, transakcie. Objekt tejto novovytvorenej transakcie uloží TransactionManager do databázy a Restu vráti ako návratovú hodnotu ID transakcie, pod ktorým bude transakcia jednoznačne identifikovaná. Toto ID vráti aj Rest v odpovedi používateľovi, ktorý na jeho základe vygeneruje 2D kód a zobrazí na svojom zariadení.

Keď bude mať prvý používateľ zobrazený 2D kód, druhý používateľ, ktorý chce, aby mu ten prvý zaplatil, ho môže zosnímať so svojím zariadením a dekódovať, aby získal ID transakcie. Potom tento druhý používateľ odošle požiadavku na server, ktorá bude znamenať, že požaduje peniaze. V tejto správe pošle Restu ID transakcie, na ktorej by mal mať "pripravenú" sumu peňazí, typ správy "GET_MONEY" a informácie o platbe ako je suma peňazí, ktoré požaduje a identifikácia používateľa, teda adresáta platby. Rest získa cez TransactionManagera objekt Transakcie z databázy a zavolá nad ním metódu getMoney s parametrom paymentInfo. V nej sa skontroluje, či suma, ktorú požaduje používateľ neprevyšuje sumu, ktorú povolil prvý používateľ. Ak ju neprevyšuje, tak sa na tomto mieste uskutoční samotný bankový prevod. Uložia sa údaje o platbe, nastaví sa stav transakcie na "COMPLETED". Ak by suma prevyšovala maximálnu povolenú sumu, stav transakcie by sa ihneď zmenil na "CANCELED". Rest bude mať informáciu o úspešnosti platby vo forme stavu transkakcie. Túto informáciu nebude potrebovať, iba ju pošle v odpovedi na zariadenie. Na tom sa zobrazí buď správa o úspešnom prebehnutí platby (ak bude stav "COMPLETED"), alebo sa zobrazí správa o neúspešnej platbe (ak bude stav "CANCELED").

Prvý používateľ bude môcť zrušiť transakciu počas celej doby, pokým bude mať zobrazený 2D kód. Toto zrušenie sa bude dať realizovať buď stlačením tlačidla na obrazovke s 2D kódom, alebo zavretím aplikácie. V prípade zatvárania aplikácie sa bude kontrolovať stav transakcie a v prípade, že transakcia ešte nebola ukončená sa zariadenie opýta používateľa, či si naozaj želá zatvoriť aplikáciu a zrušiť tým transakciu. V prípade, že potvrdí zatvorenie aplikácie, tak sa uskutoční proces rušenia transakcie. Na Rest sa odošle správa cancelTransaction s ID transakcie, ktorá sa má zrušiť a typom správy "CANCEL". Rest získa cez TransactionManagera objekt transakcie z databázy a zavolá nad týmto objektom metódu cancel, v ktorej si transakcia nastaví stav na "CANCELED" a ako návratovú hodnotu vráti pravdivostnú hodnotu, či sa podarilo "zrušiť" transakciu. Pretože v prípade, že by druhý používateľ stihol medzičasom dokončiť transakciu a tá by nadobudla stav "COMPLETED", nebude možné takú transakciu viac zrušiť. Rest pošle na klientské zariadenie správu o úspešnosti zrušenia transakcie, ktorá sa potom zobrazí v ľudsky čitateľnej podobe na zariadení.

Používateľ bude mať možnosť prezrieť si stav transakcie. Bude môcť odoslať na server správu, ktorou skontroluje stav transakcie. Na to bude slúžiť typ správy "CHECK". Spolu s typom odošle aj ID transakcie, ktorej stav chce skontrolovať. Ako odpoveď dostane od servera stav transakcie. V tejto správe bude ako typ nastavený "STATE". Tieto dve správy sa môžu zopakovať viac krát, pretože klient môže chcieť skontrolovať stav transakcie priskoro. Vtedy mu môže prísť odpoveď, že transakcia ešte nie je ukončená. V tom prípade môže znovu skontrolovať stav transakcie. Toto ale nie je nevyhnutné, klient vôbec nemusí skontrolovať stav transakcie, ak nechce. Ide len o to, aby mal možnosť hneď si pozrieť, či už prebehli peniaze druhému používateľovi.



Obrázok 25 Scenár transakcie

Návrh Rest rozhrania

Na každú z nižšie uvedených požiadaviek je v prípade, že autentifikácia je neúspešná, nasledujúca odpoveď (neuvádzame ju pri každej požiadavke, ale spoločne na začiatku, keďže sa nemení).

Odpoveď

Nesprávne prihlasovacie údaje
HTTP kód: 401 Not Authorized
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Nasleduje opis možných požiadaviek.

Požiadavka
Vytvorenie novej transakcie typu WILLING_TO_PAY
HTTP metóda: POST
URI: /transaction/wtp
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  "type": "WILLING_TO_PAY",
  "limit": maximálna suma, ktorú chce používateľ zaplatiť
}
```

Chybný vstup (záporný alebo nulový limit, neznámy typ transakcie a pod.)
HTTP kód: 400 Conflict
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Odpoveď
Úspešné vykonanie požiadavky, vytvorenie novej transakcie
HTTP kód: 201 Created
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
{
  "id": identifikačné číslo vytvorenej transakcie,
  "uri": "URI vytvorenej transakcie"
}
```

Požiadavka
Získanie peňazí z transakcie (zadanie sumy na zaplatenie)
HTTP metóda: PUT
URI: /transaction/wtp/{identifikačné číslo transakcie}
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  "sum": suma na zaplatenie,
  "description": "bližší popis platby"
}
```

Odpoveď
Platba úspešne vykonaná
HTTP kód: 200 OK
Telo: prázdne

Chybný vstup (nulová alebo chýbajúca suma a pod.)
HTTP kód: 400 Conflict
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Pokus o zmenu prebehnutej transakcie medzi dvomi inými používateľmi
HTTP kód: 403 Forbidden
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Transakcie je v nevhodnom stave (ukončená, zrušená a pod.), suma je nad limitom, príjemca je rovnaký používateľ ako platiteľ
HTTP kód: 409 Conflict
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Zrušenie platby
HTTP metóda: PUT
URI: /transaction/wtp/{identifikačné číslo transakcie}/state
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “state”: “CANCEL”
}
```

Požiadavka úspešná, transakcia bola zrušená
HTTP kód: 200 OK
Telo: prázdne

Používateľ nezačal transakciu, nemá právo zrušiť ju
HTTP kód: 403 Forbidden
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Transakcia sa už nedala zrušiť (peniaze už boli prevedené, transakcie už bola predtým raz zrušená a pod.)
HTTP kód: 409 Conflict
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Zistenie podrobných informácií o transakcii
HTTP metóda: GET
URI: /transaction/wtp/{identifikačné číslo transakcie}.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácie o transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “author”: “iniciátor transakcie”,
  “type”: “WILLING_TO_PAY”,
  “sum”: suma,
  “description”: “bližší popis platby”
  “receiver”: “príjemca platby”
  “state”: “stav transakcie”,
  “payment”: {“payer”: “e-mail platiteľa”}
}
```

Poznámka: v závislosti od stavu, v ktorom sa transakcia nachádza, sa niektoré polia nemusia v odpovedi objaviť vždy – v PENDING transakcii je napr. len typ a autor. Suma, popis, príjemca a pole payment sa objaví až po zaplatení.

Používateľ pristupuje k transakcii dvoch iných používateľov (ak je transakcia už kompletná)
HTTP kód: 403 Forbidden
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Požiadavka
Zistenie stavu transakcie
HTTP metóda: GET
URI: /transaction/wtp/{identifikačné číslo transakcie}/state.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácia o stave transakcie
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “state”: “stav transakcie”
}
```

Používateľ pristupuje k stavu transakcie dvoch iných používateľov (ak je transakcia už kompletná)
HTTP kód: 403 Forbidden
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Neznáme číslo transakcie
HTTP kód: 404 Not Found
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON chybový objekt so správou o chybe

Implementácia

Implementácia REST rozhrania
Implementácia pozostávala z nasledujúcich bodov:

- Pridanie ciest do smerovača.
- Vytvorenie rozhrania *WillingToPayTransactionResource*, ktoré predstavuje zdroj reprezentujúci transakciu. Toto rozhranie implementuje trieda *ServerWillingToPayTransactionResource*. Dôležité metódy tejto triedy sú *@Get WillingToPayTransactionDTO retrieve()*, ktorá slúži na získanie informácií o transakcii v systéme, *@Post create(WillingToPayTransactionDTO)*, ktorá vytvára novú transakciu a *@Put update(WillingToPayTransactionDTO)*, ktorá slúži na editáciu transakcie, v našom prípade na určenie výšky zaplatenej sumy, pridanie popisu pre platbu, zrušenie transakcie atď. Zvýšenú pozornosť bolo tentokrát treba venovať dôkladnej kontrole vstupných údajov do spomínaných metód a vracaniu správnych návratových kódov v každej situácii.
- Vytvorenie prenosového dátového objektu, ktorý reprezentujú informácie o transakcii. Okrem toho sme použili už existujúci prenosový dátový objekt pre platby, *PaymentDto*.

Aj v prípade tejto transakcie sme sa stretli s potrebou obmedziť vrátené informácie iba na pole o stave transakcie (tak, ako v prípade transakcie NEED_MONEY). Opäť sme sa kvôli jednoduchosti rozhodli iba podľa URI zisťovať, aké informácie sa majú vrátiť a priamo v metóde retrieve vrátiť výsledky

Testovanie

Testovanie REST rozhrania
Testy sú automatizované a sú implementované v triede *RestWillingToPayTransactionIntegrationTest* v balíku *sk.fiit.mpayserver.rest* v zložke *test*. Testovacie metódy pokrývajú nasledujúce správanie aplikácie:

- Aplikácia odpovedá HTTP chybovým kódom 401 pri pokuse o použitie rozhrania k transakciám neprihláseným používateľom.
- Aplikácia odpovedá HTTP chybovým kódom 400 pri zadaní limitu transakcie rovného alebo menšieho ako 0.
- Aplikácia odpovedá HTTP kódom 201 ak sa podarí úspešne vytvoriť novú transakciu.
- Aplikácia po vytvorení transakcie vráti v odpovedi jej kód a URI.
- Aplikácia odpovedá HTTP chybovým kódom 400 pri zadaní sumy na zaplatenie menšej alebo rovnej ako 0.
- Aplikácia odpovedá HTTP chybovým kódom 409 ak je suma na zaplatenie vyššia ako limit.
- Aplikácia odpovedá HTTP chybovým kódom 409 ak príjemca platby rovnaký ako platiteľ.
- Aplikácia odpovedá HTTP chybovým kódom 409 pri pokuse získať platbu zo zrušenej

transakcie.

- Aplikácia odpovedá HTTP chybovým kódom 409 pri pokuse druhýkrát získať platbu zo zaplatenej a ukončenej transakcie.
- Aplikácia odpovedá HTTP kódom 200 ak sa platba úspešne vykoná.
- Aplikácia odpovedá HTTP chybovým kódom 404 pri pokuse o prístup k transakcii s neznámym (neexistujúcim) identifikátorom.
- Aplikácia odpovedá HTTP kódom 200 ak sa podarí transakciu na požiadanie úspešne zrušiť.
- Aplikácia odpovedá HTTP chybovým kódom 403 pri pokuse zrušiť transakciu, ktorú používateľ nezačal.
- Aplikácia odpovedá HTTP chybovým kódom 409 pri pokuse o zrušenie dokončenej, alebo už zrušenej transakcie.
- Aplikácia odpovedá HTTP chybovým kódom 403 pri pokuse akokoľvek neoprávnene pristúpiť k niektorej transakcii (pristúpiť k dokončenej transakcii medzi dvomi inými používateľmi).
- Aplikácia odpovedá HTTP kódom 200 pri oprávnenom prístupe k informáciám o transakcii (oprávnený prístup je k čakajúcim a zrušeným transakciám a k dokončeným transakciám, ktorých sa používateľ zúčastnil).
- Aplikácia odpovedá HTTP kódom 200 pri oprávnenom prístupe k informáciám o stave transakcie (oprávnený prístup je k čakajúcim a zrušeným transakciám a k dokončeným transakciám, ktorých sa používateľ zúčastnil).
- Aplikácia pre čakajúce a zrušené transakcie vracia iba ich typ, stav a autora.
- Aplikácia pre dokončené transakcie, ktorých sa používateľ zúčastnil, vracia všetky dostupné a povolené informácie (nie je povolené vracať limit transakcie - nikdy).

Implementácia transakcie v jadre servera

Transakcia využíva podobné rozhranie ako Transakcia „NeedMoney“, nová inštancia sa získa zavolaním metódy TransactionManager#createPrimitiveTransactionWILLING_TO_PAY. Prístup k existujúcej transakcii sa získa volaním metódy TransactionManager.getTransaction(String), kde String je kľúč transakcie.
Transakciám pribudla možnosť zavolať funkciu cancel (Mail) kde Mail je užívateľ ktorý chce transakciu zrušiť. Iba užívateľ ktorý transakciu vytvoril ju môže zrušiť. Jadro sa snaží zabezpečiť čo najvyššiu bezpečnosť transakcií tým, že ak sa dostane transakcia do stavu ERROR, CANCEL alebo COMPLETE už nie je možné meniť jej stav (tieto stavy sú konečné, a pre stav COMPLETE je to aj samozrejmé).
Pri volaní transakcie sa kontrolujú všetky argumenty, a ak sa zistí chyba v argumentoch funkcie transakcia vytvorí výnimku IllegalArgumentException, pričom stav transakcie sa nemení. Ak niektorý argument nevyhovuje podmienkam transakcie považuje sa to za vážnu chybu (napríklad niekto chce viac peňazí ako je platiaci ochotný zaplatiť). V takom prípade vznikne výnimka TransactionError indikujúca vážnu chybu v transakcii. Pri vzniku výnimky počas stavu transakcie PENDING sa zmení stav transakcie na ERROR. Inak sa stav transakcie nemení. Pri vzniku výnimky nie sú vykonané žiadne činnosti nad transakciou. Po

úspešnom volaní transakcie je stav transakcie uložený a klientské vrstvy neriešia perzistenciu transakcií.

Transakcia obsahuje okrem všeobecných dát (autor transakcie, ID a podobne) transakcie nasledovné polia:

- Double payerLimit
- Double actualPayment
- Mail receiver
- String description

Transakcia má vytvorenú iba jednu špeciálnu metódu:

- getMoney(Mail receiver, Double payment);

Použitie knižnice SmartGwt na stránke

Analýza

Použitie knižnice SmartGwt umožnuje použitie veľkého množstva už vytvorených komponentov, ktoré je jednoduché použiť. Taktiež obsahuje množstvo pomocných prvkov, ktoré umožňujú jednoduchú správu dát, či funkcionalít.

Pre naše potreby potrebujeme komponenty, ktoré umožnia prihlásenie a registráciu používateľa, zobrazovanie základných informácií o používateľovi, zobrazovanie vykonaných transakcií spolu s ich detailným popisom a možnostž transakcie filtrovať. Pre všetky spomínané potreby má knižnica SmartGwt vhodné nástroje a komponenty.

Návrh

Pri návrhu web stránky sme identifikovali nasledovné pod stránky:

- Hlavná stránka – bude obsahovať základné informácie o systéme spolu s prihlásením a registrovaním nového používateľa.
- Zobrazovanie základných informácií o používateľovi – bude obsahovať základné informácie o prihlásenom používateľovi.
- Zobrazovanie vykonaných transakcií – bude obsahovať všetky vykonané transakcie prihláseného používateľa s možnosťou ich filtrovania a zobrazenia detailov danej transakcie.

Hlavná stránka

Na hlavnej stránke je potrebné mať základné informácie o systéme a taktiež informáciu o tom, ako systém použiť. Pre možnosť použitia systému je potrebné sa zaregistrovať pomocou registračného formulára. Pre zobrazenie vykonaných transakcií a ďalších údajov je potrebné sa pomocou prihlasovacieho formulára prihlásiť. Návrh obrazovky hlavnej stránky je na obrázku (Obrázok 26).

Obrázok 26 Návrh hlavnej stránky

Po prihlásení do systému je potrebné navrhnúť vhodný spôsob navigácie po stránke. Rozhodli sme sa pre navigáciu v hornej časti stránky pomocou TabBaru. Toto rozloženie je možné vidieť na ďalších obrazovkách.

Zobrazovanie základných informácií o používateľovi

Na tejto stránke je potrebné zobraziž základne informácie o prihlásenom používateľovi v čo možno najprehľadnejšej forme. Ako vhodný prvok sme zvolili tabuľku, v ktorej budú zobrazené potrebné informácie. Návrh tejto obrazovky je na obrázku (Obrázok 27).

Obrázok 27 Návrh obrazovky pre zobrazovanie info o používateľovi

Zobrazovanie transakcií

Pri zobrazovaní transakcií je taktiež potrebné mať v prehľadnej forme zobrazené vykonané transakcie prihláseného používateľa. Aj pre toto zobrazenie sme navrhli použitie tabuľky, kde budú všetky transakcie. Tieto transakcie je potrebné filtrovať podľa rôznych kritérií. Pre filtrovanie transakcií navrhujeme použitie filtra, ktorý bude nad tabuľkou transakcií. Zobrazenie detailných informácií bude taktiež v prehľadnej tabuľke pod tabuľkou transakcií. Návhr obrazovky je na obrázku ().

Obrázok 28 Návrh obrazovky pre zobrazovanie transakcií

Šprint č. 7

Pre tento šprint sme si stanovili viacero cieľov. Patrí sem implementácia webovej stránky pomocou knižnice SmartGwt podľa návrhu z predchádzajúceho šprintu, taktiež dotiahnutie nedostatkov pri implementovanom type transakcie, ktoré sa nestihli v predchádzajúcom šprinte a návrh obrazoviek pre ďalší typ transakcie.

Implementácia webovej stránky

Hlavná stránka

Ako sme identifikovali pri návrhu tejto obrazovky v predchádzajúcom šprinte, je potrebné v tejto obrazovke vytvoriť po funkcionálnej stránke prihlasovanie a registrovanie používateľa. Tieto časti už boli implementované pre samotného GWT klienta a tieto časti sú pri implementácii využité. Ide najmä a vytvorenie nového klienta metódou *create* a získanie klienta zo strany servera metódou *retrieve.* Pomocou týchto metód je teda umožnené prihlásiť sa a registrovať do systému. Implementácia hlavnej stránky potom spočíva iba vo vytvorení grafickej podoby stránky. Pre prihlasovanie i registrovanie sme zvolili vytvoriť inštanciu triedy *DynamicForm,* ktorá umožňuje jednoducho vytvárať formuláre podľa údajov, ktoré jej poskytneme.

Zobrazovanie informácií o používateľovi
Pre zobrazovanie informácií o používateľovi sme zvolili tabuľku, ktorá dokáže prehľadne zobraziť potrebné údaje. Tabuľku sme vytvorili ako inštanciu triedy *GridList*, ktorá pracuje so záznammy, ktoré sú inštanciou triedy *GridRecord.* Vytvorením takýchto riadkov tabuľky sme vytvorili riadok pre každý potrebný záznam o používateľovi. Navyše sme zoskupili údaje o používateľovi na základe toho, do akej skupiny patria (Osobné údaje, Kontaktné informácie atď). Jednotlivé údaje o používateľovi je taktiež možné zmeniť pomocou modálneho okna, ktoré sa zobrazí po kliknutí na záznam.

Zobrazovanie vykonaných transakcií
Podobne ako zobrazovanie informácií o používateľovi, tak aj zobrazovanie transakcií sme sa rozhodli vyriešiť pomocou tabuľky. Narozdiel od zobrazovania informácií o používateľovi ale táto tabuľka bude používať inštanciu triedy *DataSource*, ktorá bude pravidelne aktualizovaná a budú získavané stále nové transakcie zo servera. Na tejto stránke bude taktiež filter – inštancia triedy *FilterBuilder,* ktorý umožní jednoduché filtrovanie transakcií opäť pomocou DataSource. DataSource ešte využije aj zobrazovanie detailných informácií o transakcii pomocou inštancie triedy *DetailViewer*. Pre všetky spomínané komponenty sa DataSource nastaví metódou *setDataSource (dataSource)*.

Revízia zmien v REST rozhraní k transakciám

V tomto šprinte sme odhalili niektoré nedostatky v transakciách, ktoré znemožňovali plnohodnotné použitie systému oboma typmi klientov (webová stránka i mobilné zariadenia). V REST rozhraní k transakciám sme preto vykonali nasledujúce zmeny.

Transakcie typu NEED_MONEY

V JSON objekte reprezentujúcom platbu bolo zmenené pole sum na payedSum. Požiadavky ovplyvnené touto zmenou sú:

Požiadavka
Zistenie podrobných informácií o transakcii
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácie o transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “typ transakcie”,
  “sum”: suma,
  “description”: “popis, za čo je potrebné zaplatiť”
  “state”: “stav transakcie”,
  “payment”: { “payer”: “e-mail platcu”, “payedSum”: zaplatená suma}
}
```

Poznámka: pole payment sa v odpovedi vyskytuje, iba ak je transakcia zaplatená.

Požiadavka
Odoslanie peňazí do transakcie
HTTP metóda: PUT
URI: /transaction/{identifikačné číslo transakcie}
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “payment”: {“payedSum”: suma}
}
```

Transakcie typu WILLING_TO_PAY

Aby mal používateľ našej aplikácie príjemnejší používateľský zážitok, nie je vždy možné, aby klient dopredu vedel typ transakcie zakódovanej v 2D kóde. Doteraz server obsluhoval každý typ transakcie na URI s rozdielnou schémou – transakcie typu NEED_MONEY boli obsluhované na URI *transaction/** a transakcie WILLING_TO_PAY boli obsluhované na URI *transaction/wtp/**. URI schémy pre oba typy transakcií boli zjednotené na *transaction/**.

Ďalšou požiadavkou, ktorá vznikla z dôvodu lepšieho používateľského zážitku pre klienta, je posielanie limitu transakcie pri vyžiadaní informácie o transakcii. V rámci bezpečnosti sa ale táto informácia do odpovede zahrnie iba pre autora transakcie.

Požiadavky ovplyvnené týmito zmenami sú:

Požiadavka
Vytvorenie novej transakcie typu WILLING_TO_PAY
HTTP metóda: POST
URI: /transaction
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “type”: “WILLING_TO_PAY”,
  “limit”: maximálna suma, ktorú chce používateľ zaplatiť
}
```

Požiadavka
Získanie peňazí z transakcie (zadanie sumy na zaplatenie)
HTTP metóda: PUT
URI: /transaction/{identifikačné číslo transakcie}
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “sum”: suma na zaplatenie,
  “description”: “bližší popis platby”
}
```

Požiadavka
Zrušenie platby
HTTP metóda: PUT
URI: /transaction/{identifikačné číslo transakcie}/state
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “state”: “CANCEL”
}
```

Požiadavka
Zistenie podrobných informácií o transakcii
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Odpoveď
Informácie o transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Telo: JSON objekt vo formáte:

```
{
  “author”: “iniciátor transakcie”,
  “type”: “WILLING_TO_PAY”,
  “sum”: suma,
  “description”: “bližší popis platby”
  “receiver”: “príjemca platby”
  “state”: “stav transakcie”,
  “limit”: limit transakcie,
  “payment”: {“payer”: “e-mail platiteľa”, “payedSum”: zaplatená
suma}
}
```

Poznámka: v závislosti od stavu, v ktorom sa transakcia nachádza, sa niektoré polia nemusia v odpovedi objaviť vždy – v PENDING transakcii je napr. len typ a autor. Suma, popis, príjemca a pole payment sa objaví až po zaplatení. Taktiež pole limit sa zobrazí iba autorovi transakcie.

Požiadavka
Zistenie stavu transakcie
HTTP metóda: GET
URI: /transaction/{identifikačné číslo transakcie}/state.json
Hlavičky: Authorization: base64(mail:heslo)\n
Telo: prázdne

Implementácia zmeny
V prípade tejto zmeny treba spomenúť aj implementačnú časť, ktorá je oproti ostatným zmenám netriviálna (pri ostatných zmenách postačovali iba zmeny v DTO – dátových prenosových objektoch). Na jednom URI sa totiž v rámci Restlet štandarne obsluhuje iba jeden typ REST zdroja (u nás je zdrojom transakcia). Preto bolo potrebné implementovať potomka triedy *Finder*, ktorý má funkciu akéhosi predspracujúceho prvku. V ňom sa najprv

zistí, o aký typ transakcie sa jedná a následne sa vytvorí inštancia triedy, ktorá požiadavku obslúži. Do smerovača sa potom k príslušnej URI priradí práve tento potomok triedy *Finder*.

Zistenie sa vykonáva dvomi možnými spôsobmi:

- Ak ide o vytvorenie transakcie, typ sa zistí z JSON objektu v tele požiadavky.
- Ak ide o získanie alebo úpravu transakcie, daná transakcia sa získa z databázy a typ sa zistí priamo z nej.

Problematický bol najmä prvý prípad, keďže sme narazili na problém s tým, že ak prečítame telo požiadavky vo Finder-i, v obslužnej triede už telo nie je dostupné (aj keď by podľa dokumentácie malo byť). To sme nakoniec vyriešili tak, že telo po prečítaní do požiadavky opäť vložíme. Príslušný výňatok kódu pre predstavu uvádzam:

```
String text = request.getEntityAsText();
json = new JSONObject(text);
request.setEntity(text, MediaType.APPLICATION_JSON);
```

Konkrétna implementácia sa nachádza v balíku *sk.fiit.mpayserver.rest.resources.impl* v triede *TransactionTypeFinder*.

Prehľad transakcií

Aby sa dala plne implementovať funkčnosť webovej stránky s prehľadmi transakcií, bolo potrebné v REST rozhraní urobiť tieto zmeny:

- Pole sum v informácii o každej transakcii pravdepodobne koliduje s niektorým kľúčovým slovom v zobrazovacom rámci použitom na webovej stránke, preto bolo potrebné pridať ďalšie pole s rovnakou hodnotou, ale iným názvom.
- Aby sa mohol prehľad transakcií na klientskej strane pravidelne prekresľovať (bez nutnosti obnovenia stránky používateľom), bolo potrebné pridať možnosť zistenia, či je aktuálne zobrazený stav aktuálny.
- Kvôli správnemu filtrovaniu a zoraďovaniu dát na stránke bolo potrebné pridať vhodne naformátovaný dátum transakcie, nie iba pole s timestampom pre každú transakciu.

Požiadavky ovplyvnené týmito zmenami sú:

Požiadavka
Získanie zoznamu transakcií
HTTP metóda: GET
URI: /transactions.json
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: prázdne

Odpoveď
Zoznam transakcií
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
[
  {
```

```
    “id”: identifikačné čislo transakcie,
    “author”: “e-mail iniciátora transakcie”,
    “type”: “typ transakcie”,
    “sum”: suma,
    “suma”: suma,
    “created”: timestamp vytvorenia transakcie
    “createdDate”: “dátum vytvorenie transakcie vo formáte
mm/dd/yyyy”
    “state”: “stav transakcie”,
    “payment”: {“payer”: “e-mail platcu”, “payedSum”: zaplatená
suma}
  },

  ...
]
```

Kvôli bodu 2 zo zoznamu spomenutého vyššie bola taktiež pridaná podpora pre nasledujúcu požiadavku:

Požiadavka
Získanie informácie o najnovšej transakcii
HTTP metóda: GET
URI: /transactions/latest.json
Hlavičky: Authorization: base64(mail:heslo)\n
Content-type: application/json; charset=UTF-8
Telo: prázdne

Odpoveď
Informácie o najnovšej transakcii
HTTP kód: 200 OK
Hlavičky: Content-type: application/json; charset=UTF-8
Location: URI vytvorenej transakcie
Telo: JSON objekt vo formáte:

```
{
  “id”: identifikačné čislo transakcie,
  “author”: “e-mail iniciátora transakcie”,
  “type”: “typ transakcie”,
  “sum”: suma,
  “suma”: suma,
  “created”: timestamp vytvorenia transakcie
  “createdDate”: “dátum vytvorenie transakcie vo formáte mm/dd/yyyy”
  “state”: “stav transakcie”,
  “payment”: {“payer”: “e-mail platcu”, “payedSum”: zaplatená suma}
}
```

Šprint č.8 a č.9

V týchto šprintoch chceme pomaly ukončovať prácu na iPhone klientovi a na webovej stránke. Pridávame ale novú funkcionalitu, ktorou chceme demonštrovať použitie nášho riešenia. Ide o pokladňu (podobnú pokladni v obchode), na ktorej chceme predstaviť jedno riešenie pomocou transakcie Willing to pay a taktiež vytvorenie jednoduchého E-shopu, pomocou ktorého budeme prezentovať druhý typ transakcie.

Web stránka – automatické pridanie novej transakcie

Pridanie novej transakcie bude prebiehať nasledovne:
Po vytvorení transakcie sa transakcia uloží na serveri. V pravidelných intervaloch – 3 sekundy sa bude stránka dopytovať na server, či existuje nejaká novšia transakcia, ktorá ešte nie je vložená na stránke. To je zabezpečené v REST rozhraní pomocou URL (*transactions/latest*), na ktorej sa nachádza najnovšia transakcia, uložená na serveri. Po získaní tejto transakcie sa čas tejto transakcie porovná s časom najnovšej transakcie na webovej stránke a v prípade, že čas transakcie je na webovej stránke starší ako na serveri, prebehne pridanie všetkých nových transakcií, ktoré sú novšie ako najnovšia transakcia na serveri.

Zabezpečenie opakovania sa dopytu na server sme použitili inštanciu triedy *Timer,* ktorá v závislosti od nastavenej doby v milisekundách v pravidelných intervaloch vykonáva určenú vec.

Implementácia získania najnovšej transakcie vyzerá nasledovne:

```
private void fetchTransactionsList() {
	transactionsListResource.getClientResource().setReference("/rest/transactions/latest");
	transactionsListResource.getClientResource().getClientInfo().getAcceptedMediaTypes().add(new
Preference<MediaType>(MediaType.APPLICATION_JAVA_OBJECT_GWT));

	transactionsListResource.retrieve(new Result<TransactionListDTO>() {
			@Override
			public void onSuccess(TransactionListDTO transactions) {
				transactionDTOList = transactions;
				latestTransaction = transactionDTOList.getFirst();
				if (latestTransactionList != null) {
					if
(latestTransactionList.getCreated().toString().equals(latestTransaction.getCreated().toString())) {
						doUpdate = false;
					}
					else {
						doUpdate = true;
					}
				}
				else {
					latestTransactionList = latestTransaction;
				}

			}

			@Override
			public void onFailure(Throwable throwable) {
				SC.say("Could not load new transactions.");
			}
	});
}
```

Grafické rozhranie iPhone klienta

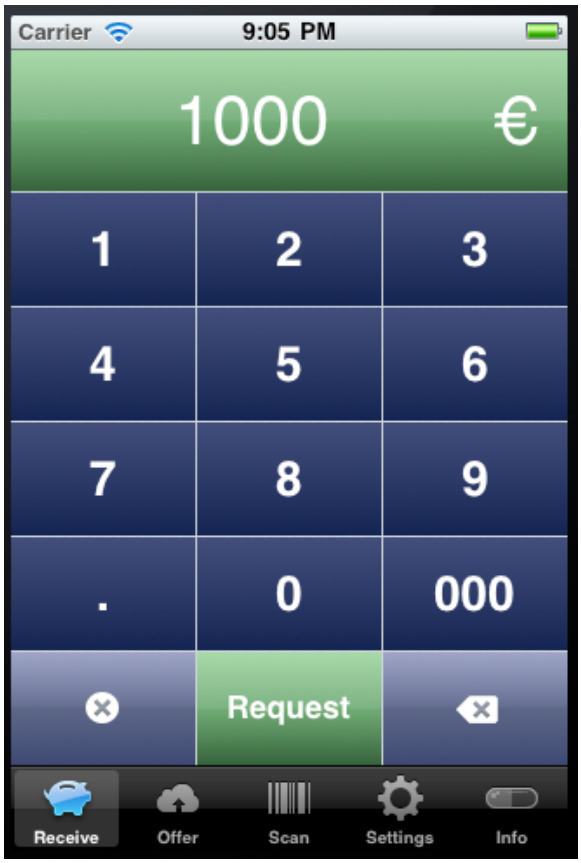
Oproti koncu 5. šprintu sme urobili zmeny v nasledujúcich obrazovkách:

- Receive (obrazovka, v ktorej sa nastaví suma, ktorú žiada jeden používateľ od druhého)
- Pay (obrazovka, v ktorej sa nastaví limit koľko peňazí je používateľ ochotný zaplatiť pri najbližšej transakcii)
- Zobrazenie 2D kódu

Receive

Na obrázku (Obrázok 29) je nová verzia obrazovky pre žiadanie sumy peňazí. Pôvodne bola používaná natívna klávesnica pre zadávanie sumy. Rozhodli sme sa vytvoriť vlastné tlačidlá, medzi ktorými budú iba tlačidlá, ktoré používateľ bude naozaj potrebovať. Zjednoduší to zadávanie sumy, pretože týmto spôsobom môžu byť tlačidlá väčšie, je ich menej, teda je to prehľadnejšie. Pri natívnej klávesnici bol prekrytý spodný panel, ktorým sa mohol používateľ pohybovať medzi obrazovkami. Nový spôsob zadávania sumy rieši aj tento problém.

Pozadie textového poľa, v ktorom je zobrazená suma, ktorú zadáva používateľ, je nastavené

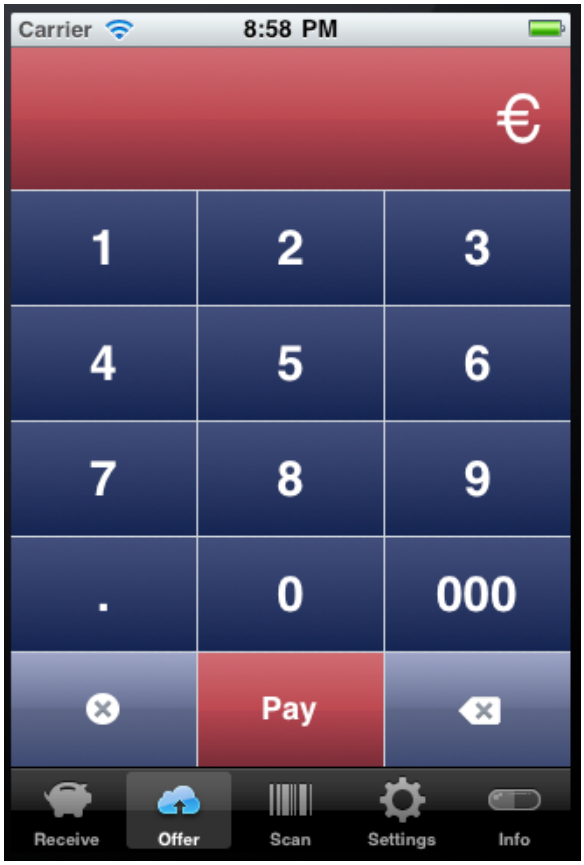


Obrázok 29 Obrazovka Receive

na zelenú farbu (s miernou grafickou úpravou). Rovnaké pozadie je nastavené aj pre tlačidlo "Request". Zelená farba bola zvolená kvôli tomu, že v tomto prípade používateľ, ktorý zadáva sumu, získa peniaze.

Pay

Na obrázku (Obrázok 30) je nová verzia obrazovky, v ktorej sa zadáva maximálna suma, ktorú je ochotný používateľ zaplatiť. Táto obrazovka je vytvorená podľa obrazovky Receive. Rozdiel oproti obrazovke Receive je v tom, že táto obrazovka má pozadie textového poľa a tlačidla na potvrdenie sumy ladené do červena, keďže v tomto prípade používateľ príde o určitú sumu peňazí.



Obrázok 30 Obrazovka Pay

Zobrazenie 2D kódu

V obrazovke s 2D kódom (Obrázok 31) sme zmenili spôsob zisťovania stavu transakcie (či ešte transakcia prebieha, alebo či ju už druhý používateľ zaplatil). Pôvodne bolo na túto funkcionalitu používané tlačidlo "Check", ktoré zobrazilo krátku správu o stave transakcie. V tejto správe bolo ďalšie tlačidlo, ktorým mohol používateľ znovu zistiť stav transakcie.

Zmenili sme to tak, že tlačidlo "Check" sa už na zistenie stavu nebude používať, ale stav transakcie sa zobrazí nad 2D kódom a každých 5 sekúnd sa znovu získajú informácie o stave transakcie. V prípade, že sa stav transakcie zmenil, zobrazí sa nad 2D kódom nový stav transakcie.

Obrázok 31 Obrazovka s 2D kódom

Vytvorenie pokladne systému

Vyvíjaná „MPay“ služba predstavuje bohaté možnosti uplatnenia vo forme vykonávania platobných či dátových transakcií. Avšak služba bola často chápaná ako iPhone na iPhone aplikácia bez všeobecného a pestrejšieho pojmu uplatnenia. Preto sme sa rozhodli rozšíriť prípady použitia „MPay“ aj na iné klientské platformy a formy transakcií. Pomocou existujúcich foriem transakcií (NEED_MONEY, WANT_TO_PAY) sme realizovali jednoduchý prípad použitia pokladne. Pôvodným zámerom bolo rozšíriť existujúci platobný softvér o potrebné funkcie, avšak nenašli sme žiadnu vhodnú implementáciu (voľne dostupná, jazyk Java, rozšíriteľná). Pokladňa umožňuje jednoduchú správu produktov, vytváranie a platbu nákupov pomocou mobilného klienta. Pokladňa skenuje QR kódy produktov a transakcií pomocou spracovania vstupu z kamery.

Platforma a nasadenie pokladne

Pokladňa bola vytvorená v Jazyku Java, ako oknová aplikácia. Využíva ZXing knižnicu pre spracovanie QR kódu. Pre prácu s kamerou musí pokladňa používať „Java Media Framework“ (JMF). Toto rozšírenie platformy Java umožňuje pracovať s video vstupom. Pred použitím aplikácie je potrebné toto rozlíšenie nainštalovať na používané zariadenie. Po inštalácii je potrebné skopírovať súbor „inštalačný priečinok/lib/jmf.properties“ do priečinka so spustiteľným súborom (.jar). Ten sa nachádza v SVN úložisku projektu ako „cashier.jar“.

Následne by mala byť pokladňa spustiteľná. Pred spustením aplikácie je vhodné spustiť JMF ukážkové aplikácie pre overenie funkčnosti kamery.

Práca s pokladňou

Pokladňa je rozdelená na dve okná a dva modálne dialógy:

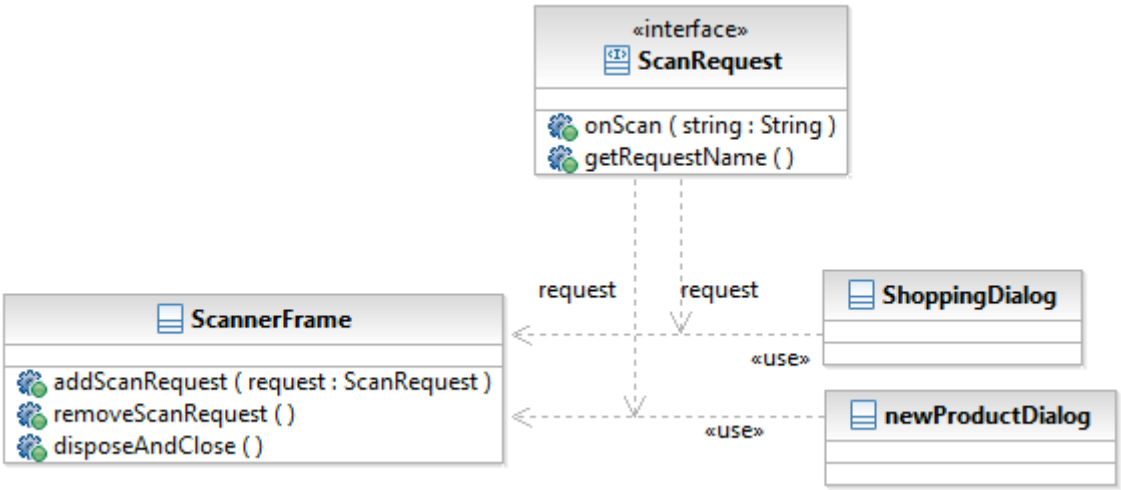
[1] Skenovacie okno
   a. umožňuje výber vstupného video zariadenia
   b. zobrazuje vstup z kamery
   c. prijíma požiadavky na skenovanie z ostatných okien

[2] Hlavné okno
   a. obsahuje záložky s nákupmi a produktmi, ktoré umožňujú úpravy v týchto kategóriách

[3] Dialóg nového produktu
   a. definuje názov a cenu nového produktu,

[4] Dialóg nákupu
   a. zobrazuje položky nákupu a celkovú cenu
   b. umožňuje nastaviť počet položiek ktorý sa vloží k nákupu

Pre prezentačné účely sme sa snažili udržať rozhranie pokladne čo najjednoduchšie. Aj z tohto dôvodu sa produkty nemusia pridávať pomocou vkladania ID produktu ale priamym skenovaním QR kódu produktu.
Architektúra systému je jednoduchá, za zmienku stojí iba použitie vzoru „Observer“ pre realizáciu skenovania, kedy je dialógové okno „pozorovateľ“ a skenovacie okno subjekt. Dialógové okno posunie skeneru požiadavku na skenovanie QR kódu. Ten nasledovne spracováva video vstup dokiaľ nedôjde ku úspešnému prečítaniu QR kódu alebo k zrušeniu požiadavky.



Obrázok 32 UML diagram pre požiadavky skenovania

Realizácia komunikácie so serverom je porovnateľná s implementáciou na klientovi iPhone (používa JSON sa formát a http komunikácia). Rozdiel je iba v implementačnom jazyku.
Aktuálna implementácia používa transakciu „WANT_TO_PAY“ kedy platbu iniciuje platiaca osoba, pričom určuje maximálny limit platby. Následne platiaca osoba ukáže vygenerovaný

QR kód na kameru pokladne. Tento prístup bol zvolený pretože v skutočnej aplikácii pokladne predstavuje kamera pokladne výkonnejšie zariadenie, teda zosnímať a spracovať QR kód je jednoduchšie a rýchlejšie. Táto forma predstavuje čiastočne bezpečnostné riziko, avšak toto riziko by sa v budúcnosti znížilo pridaním kontroly klientov, kedy by mohol platiaci určiť akým osobám dovoľuje prijať platbu. Možnosť potvrdzovania príjemcu peňazí platiacim sme vylúčili nakoľko predstavuje komplikácie pri platbe a nezhoduje sa s predstavami projektu o zjednodušovaní platieb.
Druhou možnosťou by bolo použiť platbu typu „NEED_MONEY“, ktorú iniciuje príjemca, teda odpadá bezpečnostné riziko pre platiaceho. Zmena na tento druh transakcie by vyžadovala len jednoduché zmeny.