Slovenská technická univerzita

Fakulta informatiky a informačných technológii

Ilkovičova 2, 842 16 Bratislava 4

Tímový projekt

BREYSLET 2.0

Dokumentácia k riadeniu: prvá verzia

Názov tímu:	Watchmen (tím č. 6)
Web:	http://team06-17.studenti.fiit.stuba.sk
Kontakt:	watchmenteam2017@gmail.com
Akademický rok:	2017/2018
Členovia tímu:	Bc. Michal Oláh,
	Bc. Joachim Svítek,
	Bc. Dominik Kačmár,
	Bc. Martin Škodáček
	Bc. Jakub Jasaň,
	Bc. Lam Tuan Anh
Vedúci projektu:	Ing. Vladimír Kunštár

Obsah

1 Úvod

Tento dokument predstavuje dokumentáciu k riadeniu projektu v rámci predmetu Tímový projekt. Našou úlohou v akademickom roku 2017/2018 je vytvorenie náramku Breyslet.

V druhej kapitole tohto dokumentu sú predstavení členovia tímu Watchmen. Sú opísané ich úlohy v tíme a zodpovednosti.

Tretia kapitola je venovaná prevažne manažmentu v tíme. Opisujeme jednotlivé manažmenty, ktoré pomáhajú efektívnejšiemu riadeniu a organizácií v tíme. Táto kapitola zahŕňa aj význam identifikovaných manažérskych úloh.

V štvrtej kapitole opisujeme šprinty, ktoré už prebehli. Stanovené ciele v šprintoch, dosiahnuté výsledky s krátkym opisom a ku každému šprintu aj retrospektívu.

Kapitola päť obsahuje celkovú retrospektívu za šprinty, ktoré prebehli do kontrolného bodu. Sumarizujeme všetky klady a zápory práce tímu a taktiež to ako plánujeme náš tím organizovať do budúcnosti aby sme dokázali úspešne sfinalizovať projekt.

2 Tím Watchmen

2.1 Členova tímu

Vedúci tímu Watchmen je Ing. Vladimír Kunštár. členovia tímu sú menovite: Bc. Martin Škodáček, Bc. Michal Oláh, Bc. Joachim Svítek, Bc. Dominik Kačmár, Bc. Jakub Jasaň, Bc. Lam Tuan Anh a Bc. Milan Moravčík.

Avšak posledný zmienený člen náš tím opustil v piatom týždni semestra z dôvodu pretrvávajúceho nezáujmu o projekt. Neplnil svoje úlohy, nebol aktívny na stretnutiach, prípadne sa ich nezúčastnil. Viackrát bol požiadaný o to, aby si splnil svoju časť práce, no neúspešne. Preto sme sa ako tím zhodli na jeho vylúčení. Jeho povinnosti sme rozdelili medzi zvyšných členov tímu.

2.2 Zodpovednosti členov tímu

Michal Oláh
- Scrum master
- Implementácia Back-endu
- Udržiavanie webového sídla a servera

Martin Škodáček
- Riadenie dokumentácie
- Implementácia Front-endu a Firmware
- Návrh a imlementácia web stránky tímu

Joachim Svítek
- Dizajnové návrhy
- Kvalita kódu a Manažment verzií
- Implementácia Android aplikácie

Dominik Kačmár
- Implementácia dizajnových návrhov
- Implementácia iOS aplikácie
- Manažment testovania

Jakub Jasaň
- Hardvér
- Dohliadanie na aktualizovanie úloh v nástroji
- Export evidencie úloh na konci šprintu

Anh Lam Tuan
- Hardvér
- Firmvér
- Manažment rizík

3 Aplikácie manažmentov

3.1 Manažment vývoja a plánovania

Na manažment vývoja sa v tíme používa metóda SCRUM. Po prvom tímovom stretnutí bolo za úlohu zvoliť Scrum Master-a.

Zodpovedná osoba (Scrum Master) pomáha celému tímu dosiahnuť stanovených cieľov tým, že:

- Drží si dobrý prehľad o stave úloh jednotlivých členov .
- Pozná procesy v tíme.
- Snaží sa odstraňovať vzniknuté problémy a vyvarovať sa novým.
- Motivuje členov tímu k lepším výsledkom a efektívnejšej práci.
- Pravidelne komunikuje s jednotlivými členmi tímu a aj vedúcim (aj mimo tímových stretnutí).

Pri manažovaní v tíme je dôležité udržiavať tímovú pohodu, priateľskú a uvoľnenú atmosféru tak, aby sa nikto necítil nekomfortne alebo pod neustálym tlakom. Nesmierne dôležité je komunikovať, spoločne diskutovať nad problémami a snažiť sa o to, aby boli čo najskôr odstránené. Všetky nejasnosti sú poväčšine ihneď predmetom otvorených diskusií medzi členmi tímu, čo pomáha si veci rýchlejšie ujasniť a nebrzdiť tak zbytočne priebeh vývoja.

Pri plánovaní je potrebné upozorňovať na prípadné riziká, ktoré môžu ovplyvniť dodanie plánovaných častí v šprinte. Každý šprint má trvanie dvoch týždňov a preto stretnutia v rámci šprintu slúžia na diskutovanie aktuálneho stavu pridelených úloh. V prípade potreby sa členovia tímu stretávajú aj viackrát v týždni a to aj za neúčasti vedúceho. Na konci každého šprintu prebieha retrospektíva, ktorá sa spíše a nad ktorou sa následne diskutuje. Zhodnotí sa taktiež celková práca tímu a zároveň aj jej jednotlivých členov.

Na plánovanie úloh využívame nástroj ScrumDesk. Rozsiahlosť projektu si vyžadovala rozdeliť product backlog na menšie, jednotlivé časti projektu - témy. Témy sú: Hardvér, Firmvér, Back-end, Android aplikácia, IOS aplikácia, Front-end a Dizajn. Pre každú časť (tému) sú rozpísané potrebné epicy (napr. Analýza, Návrh, Implementácia). Následne všetku prácu, na ktorej plánujeme počas šprintu pracovať, si rozdeľujeme do jednotlivých user stories, ktoré patria pod konkrétnu väčšiu časť (epic).

Rozsiahlosť projektu vyžadovala aby sme si vopred zadelili okruhy, ktorým sa budeme v projekte venovať. Preto po vytvorení všetkých user stories vieme, kto sa danej user story bude venovať a kto bude za jej dokončenie zodpovedný. Zodpovedný člen si následne user stories ďalej rozdeľuje na tasky na ktorých v šprinte pracuje. Tasky môžu mať stav TODO, IN PROGRESS alebo DONE. Stav taskov je daný podľa toho v akom štádiu rozpracovania sa daná úloha nachádza.

3.2 Manažment komunikácie

Oficiálnu tímovú komunikáciu medzi jednotlivými členmi tímu, ako aj s vedúcim projektu, vedieme prostredníctvom platformy *Slack*. Samotnú komunikáciu na Slacku máme rozdelenú na rôzne komunikačné kanály (tzv. angl. *channels*). V jednotlivých kanáloch, ak je potrebné, máme pripnuté dôležité oznamy.

Avšak nie sme vyhradený priamo len na komunikáciu na internete. Pri riešení zložitejších problémov preferujeme predovšetkým osobnú komunikáciu. Preto sa snažíme, čo najčastejšie stretávať a osobne nad problémami a projektom diskutovať. Na externú komunikáciu mimo tímu využívame tímový email.

3.3 Manažment verzií

Po diskusiu s vedúcim tímu sme sa dohodli na používaní kontrolného verziovacieho systému Git, konkrétne nástroj GitHub. Tento nástroj je široko používaný a každý člen tímu s ním už má skúsenosti. Poskytuje nám grafické rozhranie, ktoré poskytuje rôzne prehľady a poskytuje jednoduchú správu verzií jednotlivých častí projektov.

Pri práci s týmto nástrojom dodržiavame určité konvencie verziovania. Každý člen pri vypracovávaní jednotlivých úloh si vytvorí tzv. “Feature branch”, ktorá predstavuje celkovú úlohu. Pre takúto vetvu sa po jej dokončení vytvorí “pull request” pre ktorý iný člen tímu vykoná code review. Ak tento review dopadne pozitívne vetva sa “mergne” s *development* vetvou. Po dokončení určitého celku (napríklad ak softvér dosiahne stav ktorý sa dá považovať za použiteľný, napríklad funguje prihlasovanie, registrovanie a zobrazovanie údajov pre konkrétny účet) sa tento celok mergne do *master* vetvy.

3.4 Manažment prehliadok kódu

Pri riešení projektu náš tím produkuje množstvo kódu, ktorý je potreba udržiavať tak, aby bol stále dobre zrozumiteľný, čitateľný a pochopiteľný aj pre tých, ktorí sa na jeho tvorbe nepodieľali. Preto je nutné dodržiavať konvencie písania názvov tried, metód, premenných. Taktiež je nutné správne kód komentovať. Na to, aby toto všetko bolo dodržané slúži prehliadka kódu (angl. code review). Princípy prehliadok kódu sú podrobnejšie popísané v metodike prehliadok kódu.

3.5 Manažment dokumentácie

Dokumentácia sa vytvára priebežne, pričom každý člen tímu dokumentuje svoje úlohy, ktoré mal alebo má priradené. Pri vytváraní celkového dokumentu každý zapracuje svoju časť do tohto

dokumentu, pričom osoba zodpovedná za dokumentovanie kontroluje, či je zapracovanie korektné, prípadne upozorní daného autora na upravenie jeho časti.

3.6 Manažment testovania

Ešte predtým ako sa vytvorí merge *development* vetvy do *master* vetvy, sa vykoná alfa testovanie samotnými vývojármi. Testujú sa dopredu navrhnuté testovacie scenáre, pričom na to aby bolo testovanie považované za úspešné musia všetky testovacie scenáre skončiť úspešne. V prípade, že niektorý zo scenárov skončí neúspešne vytvorí sa nová úloha, na základe ktorej si jeden z členov, ktorý bude mať túto úlohu priradenú, vytvorí *feature branch* kde vytvorí zmeny v softvéri tak, aby testovací scenár skončil v ďalšom testovaní úspešne. Po vytvorení pull requestu a následom code review sa vetva mergne s development vetvou a zopakuje sa testovací scenár. V prípade opakovaného neúspešného výsledku sa celý proces zopakuje.

3.7 Manažment rizík

Za riziko pokladáme činnosť, ktorej výsledok môže byť neistý a v prípade neúspechu tejto činnosti hrozia následky, ktoré môžu mať negatívny vplyv na chod projektu či samotný projekt.

Pri analyzovaní rizík je vždy potrebné vopred určiť možné spúšťače rizika, pravdepodobnosť s akou môže potencionálne riziko nastať, možnosti akými sa dá predísť potencionálnemu riziku (teda opatrenia), rozsah dopadu v prípade, že by daná udalosť nastala a možnosti zmiernenia negatívnych vplyvov dopadu ak už udalosť nastane.

Odstúpenie člena tímu od projektu (predmet TP)
Pravdepodobnosť - nízka, 10 %
Príznaky - člen tímu nemá záujem riešiť a podieľať sa na projekte. Vyhýba sa úlohám, opakovane nedodá potrebné dohodnuté riešenia na ktorých mal pracovať.
Dopady - strata tímového člena, viac úloh rozvrhnutých medzi menší počet členov tímu, väčšia časová náročnosť pri riešení projektu
Preventívne opatrenia - motivovať členov tímu a navzájom si pomáhať, dostatočne komunikovať v tíme a riešiť problémy operatívne
Zmiernenie následkov - prerozdelenie úloh medzi členov, ešte väčšia súdržnosť a ochota si pomôcť

Nesprávne rozvrhnutie času pre riešenie konkrétnej úlohy
Pravdepodobnosť - podľa skúseností, stredná pravdepodobnosť - časom nižšia
Príznaky - zlé odhadnutie času, málo vedomostí v danej problematike

Dopady - oneskorené dokončenie úlohy
Preventívne opatrenia - diskusia ohľadom úlohy s ostatnými, usmernenie, včasné začatie riešenia úlohy
Zmiernenie následkov - pomoc ostatných členov pri riešení (ak je to možné)

Nesplnenie úlohy/úloh zadaných v danom šprinte

Pravdepodobnosť - zo začiatku vyššia, každým dokončeným šprintom nižšia
Príznaky - člen tímu nedodá kompletný výsledok konkrétnej úlohy ktorú mal splniť
Dopady - daná úloha sa bude musieť presunúť do ďalšieho šprintu, čím sa spomaľuje celkový vývoj produktu
Preventívne opatrenia - poker planning, každý člen tímu odhaduje zložitosť úlohy až kým sa celý tím nezhodne (diskusiou) na rovnakej obťažnosti konkrétnej úlohy. Takýmto spôsobom znížime pravdepodobnosť nedokončenia úlohy pretože dopredu odhadneme zložitejšie úlohy, ktoré budeme môcť rozbiť na menšie časti. Ďalším preventívnym opatrením je nabádanie členov tímu ku komunikácii počas celého šprintu medzi sebou a v prípade akéhokoľvek problému si navzájom pomôcť a poradiť.
Zmiernenie následkov - Ak sa už počas šprintu zistí, že niektorý člen tímu úlohu nestihne do konca šprintu splniť, najlepšie riešenie ak je to časovo možné, bude pomoc poskytnutá ostatnými členmi tímu.. V prípade nesplnenia úlohy po dokončení šprintu je potrebné vykonať dôkladnú retrospektívu aby sa v budúcnosti predišlo podobným situáciám.

Zlyhanie servera (výpadok)

Pravdepodobnosť - mierne zvýšená, 25 %
Príznaky - server nestíha obsluhovať požiadavky, malá dostupná pamäť, slabá priepustnosť, nestabilita serveru
Dopady - veľký, nemožno obsluhovať požiadavky aplikácií/stránky/náramku.
Preventívne opatrenia - pravidelná kontrola serveru, dostatočná priepustnosť servera, skúsenosti pri práci so serverom
Zmiernenie následkov - snaha o okamžitú nápravu a spojazdnenie servera v čo možno najkratšom čase

Zlyhanie aplikácií

Pravdepodobnosť - nízka, 5 %
Príznaky - Aplikácia sa pri vykonávaní niektorého z prípadov použitia samovoľne zavrie a operačný systém (závisí od toho aký) zobrazí chybovú hlášku o tom že aplikácia prestala reagovať.

Dopady - V lepšom prípade používateľ skúsi zopakovať prípad použitia a podľa typu chyby sa aplikácia môže ale nemusí znovu zlyhať na rovnakom mieste. V horšom prípade používateľ stratí záujem dokončiť prípad použitia a aplikáciu ohodnotí neuspokojivo.
Preventívne opatrenia - Code reviews, kontrola každej "feature vetvy", vykonanie dôkladného alfa testovania a napravenie akejkoľvek chyby v kóde, ktorá by zabraňovala niektorému z testovacích scenárov úspešné dokončenie.
Zmiernenie následkov - Ak by aplikácia zlyhala už v produkčnej verzii je nutné urýchlene chybu napraviť a vydať novú verziu aplikácie s už opravenou chybou. Je nutné konať rýchlo (závisí to od rozsahu chyby ale ideálne 1-2 dni) aby sa zminimalizoval dopad a teda aby rovnaký chybový scenár spustilo čo najmenšie množstvo používateľov.

Zlyhanie hardvéru

Pravdepodobnosť - nízka, < 1 %
Príznaky - Niektorý hardvérový prvok náramku prestal fungovať resp. dáva nezmyselné výsledky. V extrémnom prípade došlo k poškodeniu celého náramku.
Dopady - Nie je možné spoľahlivo sledovať nositeľove zdravotné funkcie.
Preventívne opatrenia - Dôraz na výber kvalitných modulov a zabezpečenie čo najväčšej odolnosti náramku.
Zmiernenie následkov - Poskytnutie náhradného náramku. Prešetrenie dôvodu zlyhania. V prípade, že by sa preukázalo, že niektorý modul často zlyháva, ho nahradíme vo výrobe kvalitnejším.

Strata internetového pripojenia

Pravdepodobnosť - vysoká, 75 %
Príznaky - Aplikácia používateľovi oznámi, že spojenie so serverom sa nepodarilo nadviazať
Dopady - Používateľ nebude môcť monitorovať osobu nosiacu náramok, rovnako ako prezerať históriu monitorovania. (nejedná sa o krízovú situáciu kedy by bol používateľ notifikovaný aj formou SMS)
Preventívne opatrenia - Oboznámiť používateľov s tým, že bez internetového pripojenia nebudú môcť na diaľku aktuálne monitorovať osobu nosiacu náramok.
Zmiernenie následkov - mobilný internet ako záloha

4 Podiel práce na jednotlivých častiach dokumentácie

Podiel práce na jednotlivých častiach dokumentácie je zobrazený v tabuľkách. V tabuľke 1 sa nachádza prehľad práce členov tímu na dokumentácií k inžinierskemu dielu a v tabuľke. 2 sa nachádza prehľad práce na dokumentácií k riadeniu.

Tab. 1 - Prehľad práce na dokumentácii k inžinierskemu dielu

	Michal	Martin	Joachim	Dominik	Jakub	Li
Úvod	20%	-	80%	-	-	-
Globálne ciele pre ZS	15%	25%	15%	15%	15%	15%
Celkový pohľad na systém • architektúra	10%	50%	10%	10%	10%	10%
• dátový model	80%	-	20%	-	-	-
• prípady použitia	10%	40%	25%	25%	-	-
• požiadavky	10%	50%	10%	10%	10%	10%
Moduly • hardvér	-	-	-	-	90%	10%
• firmvér	-	60%	-	25%	-	15%
• back-end	80%	10%	10%	-	-	-
• front-end	-	85%	15%	-	-	-
• android aplikácia	-	-	60%	40%	-	-
• iOS aplikácia			40%	60%		

Tab. 2 - *Prehľad práce na dokumentácii k riadeniu*

	Michal	Martin	Joachim	Dominik	Jakub	Li
Úvod	30%	10%	-	60%	-	-
Zodpovednosti členov tímu	30%	30%	20%	20%	-	-
Aplikácie manažmentov • manažment vývoja a plánovania	90%	5%	5%	-	-	-
• manažment komunikácie	40%	30%	10%	10%	5%	5%
• manažment verzií	10%	-	80%	10%	-	-
• manažment prehliadok kódu	5%	-	5%	50%	-	40%
• manažment testovania	10%	5%	10%	60%	5%	10%
• manažment rizík	50%	5%	25%	15%	5%	-
Používané metodiky • písania komentárov v kóde	70%	-	20%	10%	-	-
• komunikácie pre nástroj Slack	30%	60%	10%	-	-	-
• písania dokumentácie	-	100%	-	-	-	-
• verziovania	5%	-	80%	5%	5%	5%
• testovania	-	-	-	70%	15%	15%
• prehliadok kódu	-	-	5%	80%	5%	10%
Opis 1. šprintu	30%	-	40%	30%	-	-
Opis 2. šprintu	30%	-	40%	30%	-	-
Opis 3. šprintu	30%	-	40%	30%	-	-
Retrospektíva	40%	10%	10%	40%	-	-

5 Sumarizácia šprintov

Prvých 5 šprintov v ZS sme sa rozhodli pomenovať podľa piatich štádií smútku (angl. Five stages of grief):
1. DENIAL
2. ANGER
3. BARGAINING
4. DEPRESSION
5. ACCEPTANCE

Zvyšných 5 šprintov v LS sme sa rozhodli pomenovať podľa piatich štádií stresu (angl. Five stages of stress):
1. FIGHT OR FIGHT
2. DAMAGE CONTROL
3. RECOVERY
4. ADAPTION
5. BURNOUT

5.1 Sumarizácia jednotlivých šprintov

5.1.1 DENIAL

Cieľom šprintu bolo zanalyzovať oblasti, ako firmware, hardware, softvér a dizajn a ich čiastočný návrh. Naplánovali sme si úlohy, ktoré sú vypísané v tabuľke 3.

Tab. 3 - Plán prvého šprintu

Autor	Názov
Michal Oláh	Analýza riešenia serverovej časti projektu
Martin Škodáček	Analýza webových aplikácií.
Joachim Svitek	Návrh mobilných aplikácií.
Dominik Kačmár	Analýza mobilných aplikácií.
Jakub Jasan	Analýza potrebných modulov
Anh Lam Tuan	Analýza hardvéru
Milan Moravčík	Analýza a čiastočný návrh dizajnu náramku.

V rámci prvého šprintu išlo o prípravu prostredí potrebných na vývoj. Taktiež prebiehala hromadná analýza, ktorú sme si rozdelili. Michal Oláh a Martin Škodáček mali za úlohu zanalyzovať riešenia týkajúce sa Backendu a Frontendu a čiastočný návrh. Joachim Svítek a Dominik Kačmár mali na starosti analýzu a čiastočný návrh mobilných aplikácií čo predstavovalo spísanie prípadov použitia a vytvorenie návrhov jednotlivých obrazoviek v grafickom editore. Jakub Jasan a Anh Lam Tuan analyzovali hardvér a firmvér. Milan Moravčík mal za úlohu analyzovať materiály vhodné pre náramok, aby spĺňali požiadavky ako napríklad vode odolnosť alebo odolnosť voči nárazom.

Retrospektíva k šprintu

Tab. 4 - Tabuľka start-keep-stop

Start	• Lepší opis úloh v ScrumDesk aplikácii
	• Aktívnejšia komunikácia počas šprintu
	• Kvalitnejšie verziovanie kódu
	  o Jeden commit musí byť logicky ucelená časť
	  o Po každom commite by mala byť aplikácia spustiteľná
	  o Jedna feature vetva nesmie obsahovať úpravy v rôznych častiach ktoré spolu nesúvisia (napríklad viacero spolu nesúvisiacich úloh)
	• Priebežne aktualizovať zoznam taskov v ScrumDesku
	• Včas si urobiť revíziu postupu práce

Keep	• Komunikácia tímu prostredníctvom Slacku • Písanie komentárov k jednotlivým častiam kódu • Priebežná tvorba dokumentácie • Pracovať priebežne
Stop	• Písanie komentárov v slovenskom jazyku • Písanie “commit” správ v slovenskom jazyku

5.1.2 ANGER

V druhom šprinte sme pokračovali v návrhu jednotlivých častí a taktiež sme začali s implementáciou niektorých častí. Úlohy, ktoré sme si naplánovali sú vypísané v tabuľke. 5.

Tab. 5 - *Plán druhého šprintu*

Autor	Názov
Michal Oláh	Návrh databázového modelu, inštalácia aplikačného servera a implementácia databázového modelu
Martin Škodáček	Imlementácia responzívneho dashboardu webovej aplikácie, implemenácia prihlasovacej a registračnej stránky, oboznámenie sa GSM modulom
Joachim Svitek	Implementácia obrazoviek pre Android aplikáciu. (Registrácia, Login, PIN prihlásenie, domovská obrazovka)
Dominik Kačmár	Implementácia obrazoviek pre iOS aplikáciu. (Registrácia, Login, PIN prihlásenie, domovská obrazovka)
Jakub Jasan	Vytvorenie schém chýbajúcich modulov v nástroji Eaegle, počiatočná tvorba celkovej schémy
Anh Lam Tuan	Príprava schémy procesora na EAGLE
Milan Moravčík	Analýza a čiastočný návrh dizajnu náramku.

V druhom šprinte bol navrhnutý fyzický model databázy, boli vykonané práce spojené s inicializáciu servera a začala sa tvoriť schéma hardvéru pre náramok. Po inštalácii potrebných aplikácii na server bola implementovaná aj navrhnutá databáza. Spustenie, návrh a implementáciu databázového modelu vykonal certifikovaný scrummaster Michal Oláh s menšou pomocou v návrhu (Joachim Svítek). Martin Škodáček, Dominik Kačmár a Joachim Svítek mali na starosti implementáciu prezentačnej časti celkového projektu teda aplikácie a web. Jakub Jasaň a Anh Lam Tuan začali vytvárať návrh schémy pre náramok. Nakoľko tieto časti sú rozsiahlejšieho formátu sú rozdelené do viacerých šprintoch. V druhom šprinte sa podarilo implementovať registráciu a prihlásenie používateľa rovnako ako aj domovskú stránku aplikácie, zatiaľ však bez komunikácie so serverom. Z pohľadu hardvéru sa v druhom šprinte podarilo vytvoriť schémy modulov, ktoré neboli voľne dostupné.

Na konci druhého šprintu sa jeden člen tímu - Milan Moravčík - ukázal ako neochotný spolupracovať (nedodal žiadne výsledky po dvoch šprintoch), čo viedlo k jeho následnému vylúčeniu z tímu.

Retrospektíva k šprintu

Tab. 6 - *Tabuľka start-keep-stop*

Start	• Stretávať sa aj mimo oficiálnych stretnutí ak je to nutné • Intenzívnejšia komunikácia s vedúcim • Priebežne aktualizovať zoznam taskov v ScrumDesku • Prísnejšie prehodnotiť prácu a progres jednotlivých členov tímu
Keep	• Opisovanie taskov ScrumDesku • Stále aktívne komunikovať počas šprintu • Písanie komentárov k jednotlivým častiam kódu v angličtine • Priebežná tvorba dokumentácie
Stop	• Nemiešať komunikáciu na Slacku (je to veľmi zmätočné) • Nespoliehať sa na prácu jedného člena (už len 6 členov)

5.1.3 BARGAINING

V treťom šprinte sme pokračovali v implementácii jednotlivých častí. Úlohy, ktoré sme si naplánovali sú uvedené v tabuľke 7.

Tab. 7 - Plán tretieho šprintu

Autor	Názov
Michal Oláh	Implementácia aplikačnej logiky na serveri.
Martin Škodáček	Implementácia responzívnych grafov v dashboarde, práca s vývojovou doskou pre ATXMEGA128A1
Joachim Svitek	Prepojenie Android aplikácie so serverom.
Dominik Kačmár	Prepojenie iOS aplikácie so serverom.
Jakub Jasan	Finalizácia vyberania potrebných modulov a asistencia pri práci s vývojovou doskou pre ATXMEGA128A1
Anh Lam Tuan	Príprava schémy modulov na EAGLE

V treťom šprinte sa podarilo nadviazať komunikáciu so serverom kde na aplikačenj, serverovej logike pracoval Michal Oláh. Martin Škodáček pokračoval v implementácii prezentačnej - webovej - časti projektu, kde mal za úlohu dokončiť responzívny dizajn grafov. Po dokončení tejto úlohy pracoval s vývojovou doskou pre ATXMEGA128A1. Dominik Kačmár a Joachim Svítek pracovali na komunikácii so serverom na obrazovkách implementovaných v predchádzajúcom šprinte. Spolu s prihlásením a registráciou pracovali aj na zobrazení prvých testovacích dát v aplikácii. Jakub Jasaň finalizoval výber potrebných modulov a zaradil ich do schémy.

Retrospektíva k šprintu

Tab. 8 - Tabuľka start-keep-stop

Start	• Prerozdeliť prácu na projekte medzi 6 členov • Priebežne aktualizovať zoznam taskov v ScrumDesk
Keep	• Stretávanie sa aj mimo oficiálnych stretnutí • Komunikovať viac s vedúcim • Komunikovať v tíme. • Pokračovať v nastolenom tempe pri riešení taskov
Stop	• Robiť Pull Requesty s väčším predstihom • Komentovanie kódu.

5.1.4 Globálna retrospektíva do kontrolného bodu

Celkovo, za obdobie od začiatku riešenia projektu až po kontrolný bod, môžeme zhodnotiť, že sa nám podarilo plniť takmer všetky stanovené úlohy. Tímová komunikácia nebola zo začiatku na dobrej úrovni avšak po prvom šprinte sa to rapídne zlepšilo a odvtedy sa to ustálilo. Nakoľko je projekt pre každého z nás výzvou neustále sme sa snažili navzájom si pomáhať, radiť si a spoločne sa motivovať. Počas tohto obdobia to nebolo vždy dokonalé, niekedy práca nebola dodaná načas, nestíhali sme alebo vznikli nejaké nejasnosti. Avšak všetci sme si vedomí nedostatkov, ktoré sa vyskytli hlavne v prvých týždňoch práce na projekte a systematicky sme pracovali na ich odstránení. Takto zvolený prístup nám dovolil pracovať systematickejšie a efektívnejšie. Do budúcna však potrebujeme zlepšiť evidenciu úloh a sním spojené zaznamenávanie progresu na jednotlivých úlohách.

6 Používané metodiky

6.1 Metodika písania komentárov v kóde

Komentáre v zdrojovom kóde k hotovým implementovaným častiam píšeme v anglickom jazyku. Každá metóda, ktorá je žiadúca aby bola okomentovaná, je okomentovaná vo forme multikomentára. Nakoľko je projekt rozsiahli a programuje sa v rôznych jazykoch, tak forma akou sa píše komentár už závisí od konkrétneho implementačného jazyka. Rovnaké konvencie platia aj pre premenné v kóde a všetky časti, ktoré je potrebné presnejšie opísať.

Príklad okomentovania metódy v Java kóde:

```
/*
 * TestMethod
 * If breyslet will be successful, we will pass the TP
 * Returns (boolean): true - if yes | false otherwise
 */
public boolean testMethod(){
    boolean passTp = false;

    if(breyslet.isSuccessful() == true){
        passTp = true;
    }

    return passTp;
}
```

Obr. X - Príklad okomentovanej metódy

6.2 Metodika komunikácie pre nástroj Slack

Tento dokument približuje pravidlá komunikácie pri riešení projekt. V projekte na formálnu a neformálnu komunikáciu používame Slack. Prostredníctvom tohto nástroja vedieme množstvo konverzácií týkajúcich sa nielen vývoja systému, ale aj fungovania celého tímu. Metodika je záväzná pre všetkých členov tímu a produktového vlastníka, s ktorým taktiež komunikujeme prostredníctvom tohto nástroja mimo oficiálnych stretnutí.

Slack umožňuje dva spôsoby komunikácie:

- Komunikácia člen-člen - komunikácia, ktorá nie je zaujímavá pre celý tím alebo je smerovaná len konkrétnemu členovi za určitým cieľom.
- Kanály - komunikácia prebiehajúca medzi všetkými členmi alebo len členmi pracujúcimi na istom probléme. Vytvárajú sa hlavne z dôvodu udržiavania prehľadnosti.

6.2.1 Kanály

Doposiaľ sme vytvorili nasledujúce kanály:

- #organizacne - Kanál pre riešenie organizačných záležitostí ako napr. dátumy odovzdaní.
- #scrum - Kanál pre plánovanie šprintov.
- #general - Kanál pre riešenie všeobecných otázok týkajúcich sa projektu.
- #stretnutia - Kanál pre plánovanie oficiálnych a neoficiálnych stretnutí, zdieľanie poznámok, nápadov a návrhov zo stretnutí.
- #random - Kanál pre neformálnu komunikáciu.

6.2.2 Pravidlá

- Slušné správanie.
- Používať kanály na to, na čo sú určené.
- V prípade potreby kontaktovania konkrétneho človeka, nepoužívajte skupinové kanály (výnimka iba v prípade, ak to môže priniesť poznatok celému kanálu).
- V prípade potreby vytvorenia nového kanálu (napr. začiatok riešenia väčšieho problému, na ktorom pracuje viac členov) kontaktujte osobu zodpovednú za správu nástroja Slack.

6.3 Metodika písania dokumentácie

Tento dokument slúži ako návod na písanie dokumentácie. Metodika je záväzná pre všetkých členov tímu.

6.3.1 Požiadavky

Dokumentácia sa vytvára v editore Microsoft Word ako súčasť balíku Microsoft Office, ktorý musí byť licencovaný. Dokumentácia je písaná v slovenskom jazyku. V prípade výskytu fráz v cudzích jazykoch musia byť ihneď, v zátvorke za frázou, preložené do slovenčiny (ak tak nebolo urobené skôr v texte). Dokumentácia musí obsahovať aj diakritiku. Je odporúčané používať nástroj na kontrolu gramatiky integrovaný priamo v editore Microsoft Word.

6.3.2 Pravidlá

- Normálny text - font: Times New Roman; veľkosť: 12px; farba: čierna; riadkovanie: 1,25; zarovnanie: do bloku.
- Nadpis úrovne 1 - font: Times New Roman; veľkosť: 20px; farba: čierna.
- Nadpis úrovne 2 - font: Times New Roman; veľkosť: 16px; farba: čierna.
- Nadpis úrovne 2 - font: Times New Roman; veľkosť: 13px; farba: čierna.

- Zoznam - v prípade zvolenia zoznamu využívajte odrážky.
- Obrázky - musia byť správne očíslované, s popisom pod obrázkom, referencované v texte.
- Tabuľka - musia byť správne očíslované, s popisom nad tabuľkou, referencované v texte.
- Kapitola - každá nová kapitola začínajúca nadpisom úrovne 1 musí začínať na novej strane.
- Odkazy - odkazy na nejaký externý zdroj musia mať pridané hyperlinky.

6.3.3 Správa dokumentácie

Každý vytvorený dokument musí byť nahraný na GitHub, aby bol prístupný aj ostatným členom tímu. Na tomto úložisku sa nachádza aj finálny dokument. Pri jeho tvorbe autor každého dokumentu zapracuje svoju časť tak, aby to spĺňalo logickú a formálnu formu. Následne požiada o kontrolu Pull-Requestom (pozri Metodika verziovania). Osoba zodpovedná za finálny dokument kontroluje požiadavky Pull-Request. Ak je všetko v poriadku, požiadavku potvrdí. Inak oznámi autorovi pridaním komentáru alebo prostredníctvom komunikačného nástroja, aké našiel chyby. Ten je následne povinný tieto pripomienky opraviť.

6.4 Metodika verziovania

Na manažment verzií využívame systém verzií git a pre zálohu jednotlivých repozitárov ku projektu používame GitHub od spoločnosti GitHub, Inc.

6.4.1 Slovník

- **Branch** – Vetva s projektovým kódom, prípadne jeho časťou.
- **Commit** – Obsahuje všetky nové zmeny ktoré nastali v repozitári.
- **Merge** – Zlúčenie dvoch vetiev.
- **Konflikt** – V prípade že v rovnakom súbore na rovnakom mieste sú v rôznych vetvách rôzne kódy nastane pri akcii "merge" konflikt, ktorý musí programátor ručne vyriešiť.
- **Pull-Request** - Pull-request umožňuje vývojárovi oboznámiť kolegov o zmenách ktoré plánuje zlúčiť do jednej z vetiev. Pokiaľ je pull-request otvorený je možné vykonať code-review a prípadné zmeny vo forme ďalších commitov pred tým ako sa vetva zlúči s inou vetvou.
- **Rebase** - V prípade, že sa po vytvorení pomocnej vetvy B, z hlavnej vetvy A, v hlavnej vetve A pridali nové zmeny musí sa na pomocnej vetve B vytvoriť rebase. Pri tejto akcii sa všetky nové commity z hlavnej vetvy A najskôr aplikujú do pomocnej vetvy B (z ktorej boli ešte predtým dočasne odstránené pôvodné commity) a následne sa jeden po druhom aplikujú pôvodné commity z vetvy B. Na to aby sa rebase ukončil úspešne musí vývojár úspešne vyriešiť všetky konflikty ktoré počas akcie rebase nastanú.

6.4.2 Branches

V projekte sú vytvorené dve hlavné vetvy pričom akákoľvek úloha predstavuje ďalšiu vetvu ktorá po úspešnom dokončení musí byť zlúčená do jednej z hlavných vetiev - *develop*.

- **master** – Jedná sa o hlavnú projektovú vetvu, ktorá obsahuje len overenú a skontrolovanú funkcionalitu. Len jeden člen tímu môže do tejto vetvy pripájať nové funkcionality a má na starosť to aby táto vetva obsahovala len funkčné spustiteľné a otestované celky. V tejto vetve sa nachádzajú produkčné verzie aplikácie.
- **develop** – Jedná sa o druhú z dvoch hlavných vetiev projektu. Je to vetva z ktorej sa vytvárajú pomocné vetvy pre jednotlivých programátorov a do ktorej sa tieto pomocné vetvy zlučujú. Do tejto vetvy sa rovnako ako do vetvy *master* nesmú priamo vytvárať commity. Každá ďalšia funkcionalita musí byť najskôr implementovaná v pomocnej vetve, ktorá môže byť do *develop* vetvy zlúčená až po úspešnom code review. Po každom zlúčení pomocnej vetvy musí byť posledná verzia aplikácie v tejto vetve spustiteľná.
- **Pomocné vetvy** – Každý z vývojárov si pred implementovaním novej funkcionality vytvorí pomocnú vetvu do ktorej bude vytvárať commity a ktorá bude ako celok predstavovať novú funkcionalitu. Pod pojmom "nová funkcionalita" sa myslí jedna úloha - backlog item. Takáto pomocná vetva sa vytvára z hlavnej vetvy *develop* pričom sa môže vytvoriť aj z vetvy *master* avšak vždy musí byť zlúčená do vetvy *develop*, preto sa vytváranie z vetvy *master* neodporúča. Ak chce vývojár zlúčiť pomocnú vetvu, musí dodržať isté zásady spísané nižšie v dokumente. V jednej pomocnej vetve pracuje vždy len jeden vývojár.

Pred tým ako sa pomocná vetva zlúči do nadradenej vetvy musí prejsť kontrola kódu. Kód musí byť dostatočne okomentovaný, skontrolovaný formou code review a otestovaný. Ak všetko prebehne v poriadku pomocná vetva môže byť zlúčená do nadradenej vetvy.

6.4.3 Pull-Request

Pull-Request vetvy podlieha nasledujúcim pravidlám:

- Vo vetve sa nenachádzajú žiadne chyby a funkcionalita vetvy je otestovaná.
- Vetva je kompilovateľná a pri kompilácii nehlási žiadne chyby ani upozornenia.
- Pravidlá vetvy do ktorej sa pridáva podradená vetva sú nadradené.

Každý pull-request musí byť schválený formou code review.

Žiaden vývojár si nesmie pull-request schváliť sám.

Pull-request je požiadavka na kontrolu vývojárom iným ako tým, ktorý sa podieľal na práci na vetve ktorá o pull-request žiada. Vývojár si z boardu vezme úlohu ktorá reprezentuje skontrolovania a schválenie vetvy žiadajúcej o pull-request, vykoná kontrolu a v prípade že je kontrola úspešná povolí zlúčenie vetvy. Ak kontrola úspešná nebola nájdené chyby a nedostatky oznámi vývojárovi ktorý na vetve pracoval a ten musí chyby odstrániť.

6.4.4 Commit

Commit obsahuje všetky nové zmeny ktoré v repozitári nastali. Všetky commity ktoré vývojár vytvorí sú uložené lokálne do momentu kedy sa vývojár nerozhodne commity nahrať na vzdialené úložisko, v našom prípade GitHub. Každý commit obsahuje správu vo forme zoznamu zmien vo funkcionalite a ich krátkym opisom. Správa musí byť krátka, jednoznačná a musí byť jasné čo je v kóde nové.

6.4.5 Súhrn pravidiel odosielaného kódu

Kód, ktorý chce developer pridať do nadradenej vetvy podlieha nesledujúcim pravidlám.

Odosiela sa výlučne:

- fungujúci kód
- otestovaný kód
- okomentovaný kód
- stabilný kód

6.5 Metodika testovania

Táto metodika určuje, ako testovať jednotlivé časti kódu a je záväzná pre všetkých členov tímu, pretože každý člen tímu je zodpovedný za správnosť svojho kódu. Pomocou testovania vieme zabezpečiť, v prípade úpravy kódu, zachovanie pôvodná funkcionalita kódu. V rámci Tímového projektu implementujeme veľké množstvo funkcionality, ktorá je navzájom prepojená. Na implementácii jednej funkcionality môžu pracovať viacerí členovia tímu. Potrebujeme zabezpečiť, aby svojou implementáciou programátor neporušil korektnú funkcionalitu, ktorá bola už v minulosti implementovaná. Preto sme zaviedli v tíme povinnosť unit testovania a integračného testovania.

6.5.1 Pojmy

Unit testovanie je testovanie, ktoré sa zameriava na špecifickú časť kódu, resp. konkrétnu funkcionalitu. Môže byť vykonávané automaticky alebo manuálne.

Integračným testovaním sa testuje správnosť implementácie metódy, funkcie, modulu a ich správnej spolupráce s inými časťami kódu po ich integrácií do celého systému. Testuje sa zachovanie správnej funkcionality už predtým integrovaných častí systému, ktoré ovplyvňuje daný modul.

Používateľské testovanie sa týka hlavne používateľského rozhrania prostredníctvom používateľských scenárov.

6.5.2 Pravidlá

- Testovanie sa musí vykonať pred pull-requestom.
- Každý je zodpovedný za testovanie svojho kódu.
- Pre každú funkcionalitu sa vykonáva unit test.
- Po unit testoch sa vykoná integračné testovanie.
- Po úspešnom otestovaní sa pokračuje v pull-requeste.

6.5.3 Priebeh testovania

Testovanie prebieha v podobe unit testov, ktoré vykonávajú samotní programátori. Pre každý task z backlogu si zodpovedný programátor vytvorí branch. V rámci vytvorenej branche pracuje na tasku a po dokončení programovania a okomentovaní kódu otestuje svoju prácu. Testovanie prevedie hlavne na svoju časť kódu, ale taktiež otestuje celú integritu kódu po vytvorenej zmene. Pokiaľ všetky testy úspešne prebehli, testovanie končí. Pokiaľ aspoň jeden z testov neprebehol správne, implementáciou odstránime chybu v kóde a opäť testujeme celú integritu kódu.

6.6 Metodika prehliadok kódu (code review)

Predmetom tejto metodiky je definovanie spôsobu kontrolovania kódu iným programátorom. Tým je zabezpečené zbytočné opakovanie kódu, čitateľnosť a pochopiteľnosť kódu pre osoby, ktoré sa nezúčastnili na implementácii daného programu. Táto metodika je záväzná pre všetkých programátorov tímu.

6.6.1 Postup code review

Účastníci code review sú: programátor a posudzovateľ. Programátor je ten, ktorý vytvoril daný kód a posudzovateľ je ten, ktorý kontroluje kód a dáva posudky.

Aby code review prebiehalo efektívne, treba dodržať nasledujúci postup:

1. Programátor sa pokúsi spraviť si code review vlastného kódu, t.j. skontroluje či dodržal základné konvencie písania kódu.

2. Programátor po dokončení svojho kódu a jeho otestovaní vytvorí pull-request.
3. Programátor požiada prostredníctvom komunikačného kanála iného programátora o code review.
4. Posudzovateľ vykoná posudzovanie kódu. Pozrie kód, hodnotí kvalitu kódu, skontroluje splnenie základných funkcii.
5. Ak kód je hodnotený ako korektný, zmena v kóde môže byť mergnutá do hlavnej vetvy a proces code review skonči. Ak nie, programátor podľa spätnej väzby urobí zmenu v kóde a proces opakuje v kroku 2.

6.6.2 Ustanovenia code review

Pri code review treba dodržať tieto pravidla:

- Dodržiavať základné konvencie písania kódu ako napr.:
  - Názov triedy má formát *ThisIsClass().*
  - Názov metódy má formát *thisIsMethod().*
  - Názov premennej má formát *thisIsVariable().*
  - Dodržať princípy DRY (Don't Repeat Yourself).
  - Dodržiavať formátovanie (odsadenie, odriadkovanie a pod.).
  - Komentovanie tried, metód, parametrov.
- Jedna osoba nevykonáva code review pre kód dlhší ako 300 riadkov, príp. pre kód ktorého kontrola môže zabrať viac ako hodinu (komplikované metódy, previazané moduly).
- Code review sa vykonáva až po testovaní.
- Spätná väzba musí byť ľahko pochopiteľná pomocou komentárov na miestach v kóde alebo v komunikačnom kanáli.

Pozn.: Doporučujeme aj metódu programovania *side-by-side*, kedy programátori programujú fyzicky vedľa seba a priebežne si vzájomne konzultujú svoje programy. V tomto prípade nie je nutné vykonávať code review.