

ТЕХНОЛОГИЧЕСКО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Android клиент за Bugzilla сървър

Дипломант:

Евгений Янев

Научен ръководител:

гл. ас. инж. Любомир Чорбаджиев

С О Ф И Я

2 0 1 2

Съдържание

Увод	6
Първа Глава	8
Проучвателна част. Преглед на съществуващи подобни програмни системи и продукти и преглед на известните развойни средства и среди	8
1.1. Преглед на съществуващи системи за следене на бъгове	8
1.1.2. "Jira"	9
1.1.3. "MantisBT"	10
1.2. Методи за комуникация с Bugzilla сървър	11
1.2.1. "WebService" програмният интерфейс	12
1.2.2. REST програмният интерфейс	13
1.3. Преглед на съществуващи Android клиенти за комуникация с Bugzilla сървъри	15
1.3.1. "BugClubZilla"	15
1.3.2. "BugDroid"	16
1.3.3. "Bugz: Mobile Bugzilla Client"	17
1.4. Развойни среди	18
Втора глава	20
Изисквания към програмния продукт, описание на структурата на програмата и базата данни	20
2.1. Изисквания към програмния продукт	20
2.2. Избор на езика за програмиране и софтуерните средства	20
2.3. Структура на приложението	21
2.3.1. Адаптери ("adapters")	21
2.3.2. Слушатели ("listeners")	22

2.3.3. Пакетът "org.elsys".....	22
2.3.4. Пакетът "org.elsys.data"	22
2.3.5. Пакетът "org.elsys.dialogs"	23
2.3.6. Пакетът "org.elsys.parser"	23
2.3.7. Пакетът "org.elsys.requests".....	23
2.3.8. Пакетът "org.elsys.utilities"	23
2.4. Структура на базата данни	23
2.5. Осъществяване на заявки към сървъра	24
2.5.1. "GET" заявки	24
2.5.2. "POST" / "PUT" заявки	25
Трета глава	28
Описание на начина на реализация на приложението	28
3.1. Екрани на приложението.....	28
3.1.1. Main.....	28
3.1.2. AccountManager	29
3.1.3. BugList.....	31
3.1.4. BugComprehensiveView.....	32
3.1.5. FileBugChooseProduct и FileBugDetails.....	34
3.2. Основни компоненти на приложението	35
3.2.1. BugzillaActivity и общо Activity класа	35
3.2.2. Action Bar.....	36
3.2.3. ActionItemListener.....	37
3.2.4. Запис на акаунти и конфигурации	38
3.2.5. Криптиране на данни	39
3.2.6. Заявки към сървъра	40
3.2.7. Парсване на данни	52

3.3. Основни методи и класове, които са заложиени в Android операционната система	53
3.3.1. Интерфейс	53
3.3.2. Преминаване между екрани	54
Четвърта Глава	55
Ръководство на потребителя	55
4.1. Изисквания и инсталация на приложението	55
4.1.1. Инсталация на приложението през устройството: ...	55
4.1.2. Инсталация на приложението през браузър:	57
4.2. Работа с приложението	59
4.2.1. Мениджмънт на акаунти	59
4.2.2. Търсене и редакция на бъгове	62
4.2.3. Създаване на бър	67
Заклучение	69
Използвана литература	70

Увод

Известно е, че няма софтуерен продукт без бъгове. Дори най – добрите приложения имат такива, макар и още да не са забелязани. Ето защо, докладването на бъгове е толкова важно, както за големите софтуерни компании, така и за отделни разработчици, като за целта са създадени така наречените “системи за следене на бъгове” (“bug tracking systems”).

“Bugzilla” е такъв вид система. Тя е мощно средство, което помага на софтуерните екипи ефективно да осъществяват връзката помежду си, подобрява качеството на продукта и управлява целия процес по разработка на софтуера, тъй като успешните проекти често са резултат на добра организация и комуникация.

През последните години светът се обвързва все повече с развитието на мобилните технологии, което неминуемо води до промяна в нашия живот. Ако преди телефоните се използвали единствено за провеждане на разговори, то сега техните възможности са безкрайни. Но за да стане това възможно, е нужно нещо, което да управлява телефона и да го направи “умен”, а именно операционна система.

“Android” е една от най - широко разпространените и най - бързо развиващи се мобилни операционни системи в днешно време, управлявана от един от големите брандове в света на технологиите - “Google”. Linux базирана, отвореният код на системата неминуемо допринася за нейното разрастване, като по последни данни на ден в света 850, 000 Android устройства се активират, а броят на приложенията в “Google Play” наброяват повече от 450, 000. Водещият пазарен дял на Android показва, че макар новаторска и все още млада, тази операционна система си е извоювала своето място и не показва никакви признаци да спре своето развитие.

Целите на настоящата дипломна работа са създаване на Android клиент за Bugzilla сървър. Приложението трябва да може да изпълнява различни заявки към сървъра, като за целта се използва "REST" програмният интерфейс. Данните от сървъра трябва да се визуализират в удобен за потребителя интерфейс.

Първа Глава

Проучвателна част. Преглед на съществуващи подобни програмни системи и продукти и преглед на известните развойни средства и среди

1.1. Преглед на съществуващи системи за следене на бъгове

Системите за следене на бъгове са намерили широко приложение в целия технологичен свят и са неминуема част от работата на всички големи софтуерните компании, търсещи добра организация на своята дейност. Има най – различни такива, различаващи се по своите качества и услуги, които предлагат.

1.1.1. “Bugzilla”

“Bugzilla” е една от водещите системи за следене на бъгове на пазара. Разработена от компанията Mozilla като софтуерен продукт с отворен код и разпространяваща се под “Mozilla Public License”. Този лиценз позволява не само свободното използване на системата, но и нейното модифицирането. Първоначално написана на езика „Tcl“, по – късно пренаписана на „Perl“, с надеждата повече хора да допринесат за нейното развитие. За бази данни могат да бъдат използвани различни продукти като “MySQL”, “PostgreSQL”, “Oracle” и “SQLite”. Обикновено бива инсталирана на Linux операционна система и работеща с помощта на “Apache HTTP Server”, но всеки уеб сървър, който поддържа “CGI” като “Lighttpd”, “Hiawatha”, “Cherokee” може да бъде използван.

“Bugzilla” предлага широка функционалност, която много от комерсиалните системи от този вид не притежават. Част от услугите, които тя предлага са:

За потребителите:

- възможности за разширено търсене – бива два вида – опростено търсене, което улеснява новите потребители, и разширено търсене, което предлага възможност за създаване на различен тип заявки
- известия по електронна поща
- показване на листа с намерени бъгове в различни формати
- докладване за резултатие на определено търсене по график чрез електронната поща
- автоматично разпознаване на бъгове
- създаване/променяне на бъгове по електронна поща
- система за запитвания ("Request system")
- запазени и споделени търсения

За администраторите:

- отлична сигурност на системата - "Bugzilla" работи под така наречения "taint" режим на "Perl", за да предотврати "SQL Injection", като също така предлага много добра система за защита от "Cross-Site Scripting".
- механизъм за персонализиране на инсталацията
- локализация
- поддържане на "mod_perl" за отлична производителност
- "XML-RPC" и "JSON-RPC" интерфейси
- различни методи за заверяване ("authentication")

1.1.2. "Jira"

"Jira" е едно от най - добрите алтернативни решения и разбира се, една от най - разпространените системи. Продуктът се разработва от софтуерната компания "Atlassian" от 2002 година насам. "Jira" е комерсиален софтуерен продукт, като цената му зависи от максималния брой на потребителите. Освен платена версия,

“Atlassian” предлагат и безплатна версия за проекти с отворен код, които трябва да спазват определени критерии. За обучаващи се потребители пълният сорс код е наличен под “developer source license”, който позволява различни модификации. “Jira” е написана на Java, а за база данни може да бъде използвана “MySQL”, “PostgreSQL”, “Oracle”, “SQL Server”. За “отдалечено извикване на процедури” (“remote procedure call”) “Jira” поддържа “SOAP”, “XML-RPC” и “REST”. “Jira” може да се интегрира със системи за контрол на версиите като “Subversion”, “CVS”, “Git”, “Clearcase”, “Visual SourceSafe”, “Mercurial” и “Perforce”. Бива доставяна на различни преводи, включващи английски, японски, немски, френски и испански.

Удобната архитектура за плъгини към “Jira” е довела до създаването на голям брой такива допълнения. Също така трябва да се отбележи, че разработчиците имат възможност да интегрират различни приложения от трето лице към “Jira”. Всичко това е довело до използването на “Jira” от голям брой софтуерни продукти по целия свят, като “jBoss”, “Spring Framework”, “Skype”, както и много други.

1.1.3. “MantisBT”

“MantisBT” е безплатна, с отворен код, уеб базирана система за следене на бъгове, лицензирана под условията на GNU General Public License (GPL). “MantisBT” е написана на PHP. Уеб базираният потребителски интерфейс е създаден, използвайки XHTML, стилизиран и представен чрез CSS. Данните на програмата се съхраняват в релационни бази данни като “MySQL”, “PostgreSQL”, Microsoft SQL, IBM DB2 и Oracle. “MantisBT” предлага на своите клиенти няколко вида услуги.

Плъгин системата е добавена с версия 1.2.0. Този вид система позволява разширението на програмата, както чрез официално поддържаните, така и чрез плъгини от трето лице.

“MantisBT” позволява изпращането на имейл нотификации при направени промени по системата. Потребителите имат различни възможности, като да определят типа на съобщенията, които получават, да налагат филтри, които да определят минималната суровост (“severity”) на бъговете, за които да получават нотификации, да се абонират за бъгове, които са важни за тях, добавена е функционалност за RSS фийдове. В допълнение към това, “MantisBT” поддържа интеграция с “Twitter”, което позволява публикуването на нотификации директно в социалната мрежа, когато съответният проблем бива разрешен. Възможно е вградената система за нотификации да бъде надградена, като се допълни с нови функции, като например изпращането на текстови съобщения (“SMS”) или обновяване на статусите във външна система за мениджмънт на проекти.

От 2010 година “MantisBT” поддържа интеграция с различни системи за контрол на версиите, като “Gitweb”, “GitHub”, “WebSVN” и “SourceForge”.

1.2. Методи за комуникация с Bugzilla сървър

С разрастването на мобилните технологии следва и промяната на нуждите на потребителите. Същите изискват продукти, които да им позволяват лесно и удобно да достъпват до услугите, които използват за своята работа, училище и други различни дейности. Bugzilla е един такъв вид услуга и като такава би следвало и нуждата от приложения, които да позволяват работата с този вид среда. Но преди да се направи преглед на вече съществуващите подобни приложения, ще бъдат разгледани начините, чрез които може да

бъде осъществена комуникация с един Bugzilla сървър. Bugzilla може да бъде достъпвана по два начина.

1.2.1. "WebService" програмният интерфейс

Първият начин е чрез "WebService" програмния интерфейс на Bugzilla. Това е стандартният интерфейс, с помощта на който външни програми могат да си взаимодействат с Bugzilla. Същият предоставя множество методи, разделени в различни модули. "Bugzilla Project" се стреми да запази интерфейса стабилен между различните версии, така че приложение, написано за една версия, да може да продължи да функционира нормално

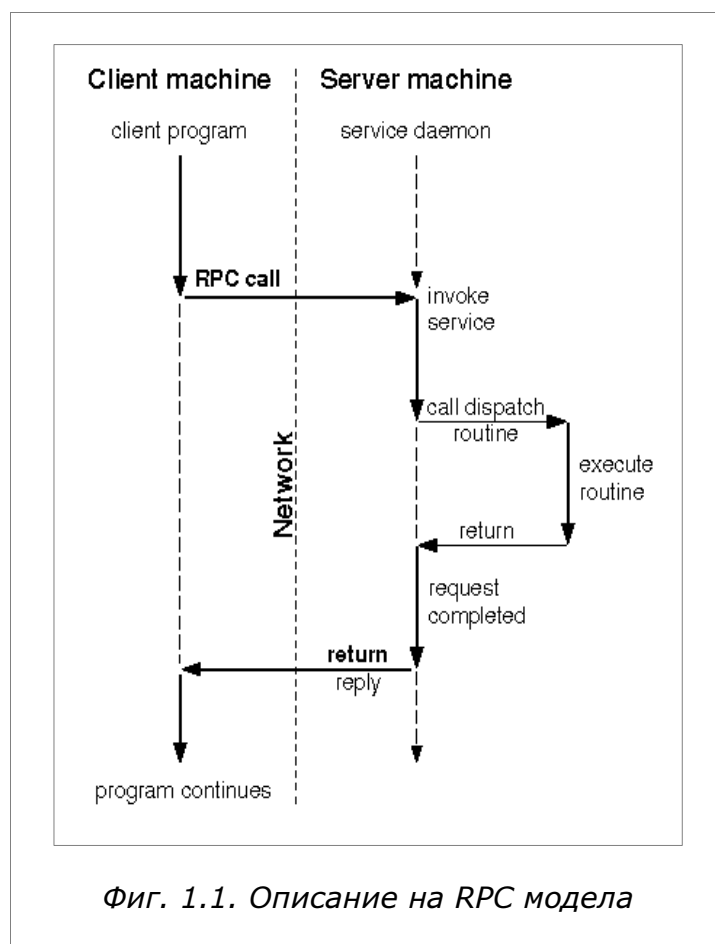
и за бъдещи обновления.

Този интерфейс може да бъде достъпван чрез използването на два вида протоколи - "XML-RPC" или "JSON-RPC".

Разликата между двата е във формата на съобщенията, чрез които се осъществява комуникацията

между клиентското приложение и сървърната част, съответно "XML" или "JSON". Има някои особености и за двата протокола, като типа

на параметрите, които се изпращат, както и нужните хедъри на заявките. Тук трябва да се отбележи, че за момента JSON-RPC интерфейсът към Bugzilla е експериментален, като пълна поддръжка има единствено за XML-RPC интерфейса.



Фиг. 1.1. Описание на RPC модела

В основата на двата вида протоколи стои така нареченото "Отдалечено извикване на процедури" ("Remote Procedure Call"). "Remote Procedure Call" (RPC) е протокол, който изисква услуга от програма, намираща се на отдалечен компютър през мрежа. За работата му се предполага наличието на транспортен протокол от ниско ниво като "TCP" или "UDP" за преноса на данни между комуникиращите програми. RPC използва клиент/сървър модела. Програмата, която изпълнява заявката, тоест тази, която изисква някаква услуга, е клиентът и програмата, която доставя тази услуга, е сървърът. Комуникацията между двете системи се осъществява в няколко стъпки. Първо, клиентският процес изпраща съобщение към сървърния процес, което включва параметрите, нужни на процедурата, която трябва да се извика и започва да чака за отговор (блокира) или докато се получи отговор, или докато не изтече зададеното време за връзка. След това процесът от сървърната страна, който до момента е бил "спящ", извлича параметрите за процедурата, изчислява резултата от изпълняващата услуга, връща отговор и започва да чака за следваща заявка. Накрая клиентският процес получава отговора, извлича резултатите и продължава своята работа.

1.2.2. REST програмният интерфейс

Вторият вариант за достъп до Bugzilla сървър е чрез така нареченият "REST" програмен интерфейс. При него все още няма поддръжка на някои от услугите, които биват предоставяни от "WebService" интерфейса, но трябва да се вземе предвид, че това са все още ранни негови версии, като последната "1.1" е от 5 март 2012 година. "Трансфер на репрезентацията на състоянието" ("Representational state transfer") или накратко REST е стил в софтуерната архитектура, използван за достъпване на ресурси през

интернет. Терминът за първи път е въведен от Рой Фийлдинг през 2000 година, който е един от авторите на "Хипертекстовия трансферен протокол" ("Hypertext Transfer Protocol" - HTTP). REST използва вече съществуващи технологии и протоколи, като HTTP, но не е ограничен единствено до него. REST архитектурите могат да се базират и на други протоколи от Приложния слой ("Application Layer"), оползотворявайки максимално вече съществуващите интерфейси на избрания протокол, като по този начин намаляват нуждата за добавяне на нови специфични за приложението сегменти. Най – голямата имплементация на система, съответстваща на този вид архитектура, е "Глобалната световна мрежа" ("World Wide Web").

Този вид архитектура се състои от клиенти и сървъри, Клиентите изпращат заявки към сървърите, които от своя връщат необходимия ресурс. Заявките и отговорите се основават на трансфера на репрезентациите на ресурсите. Ресурс може да бъде по същество всяка ясна и смислена концепция, която може да бъде адресирана чрез своя "Неизменен идентификатор на ресурса" ("Uniform resource identifier"). Репрезентацията на ресурс обикновено представлява документ, който представя сегашното или бъдещето състояние на ресурса. Клиентът започва да изпълнява заявки, когато е готов да премине към ново състояние. Докато една или повече заявки се изпълняват, се счита, че клиентът се намира в транзитно състояние. Репрезентацията на всяко състояние на приложението съдържа линкове, които могат да бъдат използвани, когато клиентът реши да премине в ново състояние.

Самото име на този вид архитектура е предназначено да покаже, как всъщност трябва да се държи едно добре проектирано уеб приложение - мрежа от уеб страници (виртуална машина на състоянията), през която потребителят прогресира чрез избиране на линкове (смени на състоянието), които водят към следващи страници

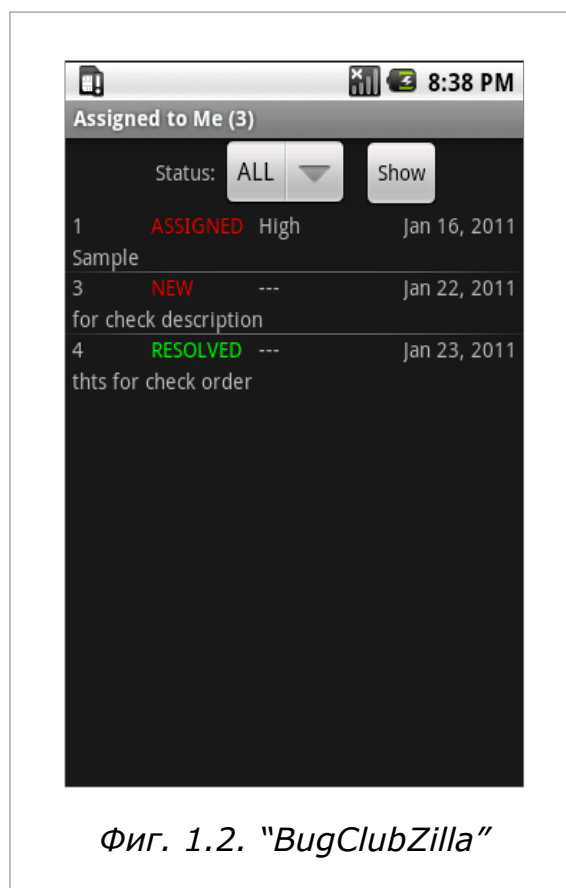
(представляващи следващите състояния на приложението), които биват пратени към потребителя и рендирани по нужния начин.

1.3. Преглед на съществуващи Android клиенти за комуникация с Bugzilla сървъри

На пазара за Android приложения за момента има няколко много интересни алтернативни решения, които заслужават да бъдат упоменати.

1.3.1. "BugClubZilla"

"BugClubZilla" е първото от решенията. За комуникацията си с Bugzilla сървърите приложението използва "XML-RPC" протокола. При стартирането на приложението се показва прозорец, където трябва да се въведат данните за акаунта на потребителя и адресът на сървъра, като може да се избира, дали използваният протокол да е "HTTP" или "HTTPS". Приложението не предлага никаква функционалност по отношение на търсене на бъгове по ключова дума или търсене на бъгове, на които потребителят е създател. Единственото, което може да покаже, са бъговете, на които потребителят е "assignee", тоест тези бъгове, които той трябва да



разреши. Има опция за филтриране на бъгове в зависимост от това какъв статус имат те.

1.3.2. "BugDroid"

Второто приложение се нарича "BugDroid". За разлика от останалите решения, това не предлага възможност да се въвежда адрес на Bugzilla сървър, тоест програмата работи с няколко сървъра, които са зададени по подразбиране. Въпреки това, продуктът има различни и интересни други качества. На първо място, потребителският интерфейс на приложението е много добре стилизиран, с лесни за навигация бутони. Трябва да се отбележи и анимацията, която се появява при стартиране на програмата. От началния екран имаме четири главни бутона в менюто и един, който е изведен най - горе на екрана и който ни позволява да правим бързо търсене за бг по неговия идентификационен номер.

При разширеното търсене на бъгове имаме избор да зададем от преди колко време трябва да са били създадени, техните статуси и продуктът на самия бг. Удобна е и предоставената възможност за запазване на търсенето. По време на обработването на резултатите от търсенето отново се показва друга интересна анимация. Листът

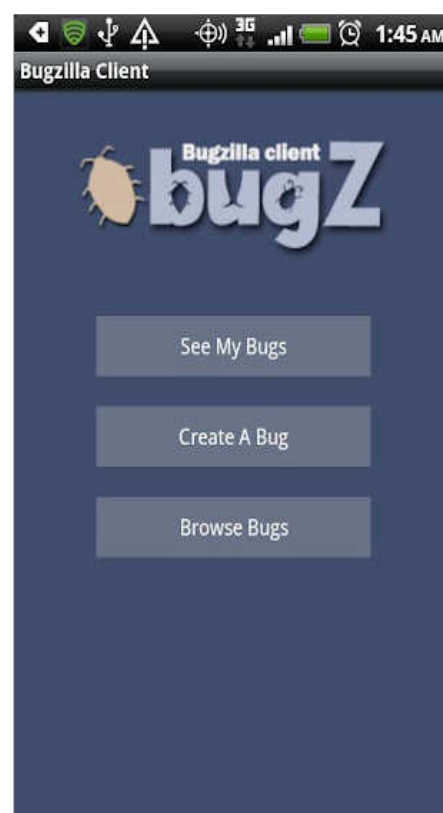
с бъгове е представен по разбираем начин, показвайки най - важната



информация. При избиране на бърг от листа се зарежда информация за прикачените файлове към бърга ("attachments"), детайлна информация за него, коментари, както и СС ("Carbon copy"). Интересна е предоставената възможност разглежданият бърг да се кешира, като по - късно може да се достъпва до него без необходимост от интернет достъп. Едно от неудобствата на програмата е, че при разглеждане на детайлите за бърга, неговият статус и резолюция се представят като падащи менюта и когато потребителят ги разглежда има идеята, че може да променя тези две настройки, което всъщност не винаги е така. Като цяло приложението е удобно и лесно за употреба при условие, че потребителят няма намерение да го използва за следене на бъргове на свой сървър.

1.3.3. "Bugz: Mobile Bugzilla Client"

Третото решение се нарича "Bugz: Mobile Bugzilla Client". Това е единственото от приложенията, при които връзката към Bugzilla сървъра не успя да се осъществи. Въпреки това, трябва да се отбележи, че приложението е все още в начална версия. В документацията към него се споменава, че услугите, които то ще предлага са показване на лист от бъргове в зависимост от техния продукт, показване на бърговете, които са възложени ("assigned") към потребителя, разглеждане на



Фиг. 1.4. "Bugz: Mobile Bugzilla Client"

подробна информация за бъга, изреждане на коментарите, както и създаването на нови бъгове. И въпреки че нито една от тези услуги за момента нямаше възможност да се осъществи, приложението изглежда обещаващо.

1.4. Развойни среди

За разработването на приложението е използван "Android Software Development Kit" (Android SDK), който осигурява всички необходими инструменти за това като компилатор, дебъгер, библиотеки, емулатор, документация, както и различни примери от код. Приложенията се пишат на програмния език "Java" и използват специална виртуална машина, така наречената "Dalvik Virtual Machine". Освен това "Google" предоставя и "Android Development Tools" ("ADT") плъгина за "Eclipse", който не само оползотворява многообразието от услуги, предоставяни от средата за разработка, но и добавя нови такива, което до голяма степен улеснява процеса на създаване на приложението.

Компоненти на едно Android приложение са:

- "Activity" - репрезентира презентационния слой на приложението
- "Content Provider" - предоставя структуриран интерфейс към данните на приложението, чрез който други приложения могат да достъпват до тази информация. Данните могат да бъдат съхранявани по различен начин, като например във файловата система или в "SQLite" база данни
- "Broadcast Receiver" - този вид компонент получава отговаря на системни съобщения и "Intent" - и. Той ще бъде уведомен от Android системата, ако възникне определено събитие, като промяна в нивото на батерията, смяна на езика и други

- “Services” - това са услуги, които изпълняват задачи на заден фон без да предоставят потребителски интерфейс. Те могат да уведомяват потребителите чрез системата за известия (“notifications”) в Android

Други използвани термини:

- Изгледи (“Views”) - представляват различни части от потребителския интерфейс като бутони, текстови полета и други. Базовият клас за всички изгледи е “`android.view.View`”.
- “Intent” - това са асинхронни съобщения, които изискват някаква функционалност от други компоненти като “Services” или “Activities”. Приложението може да извика компонента директно (“explicit Intent”) или да използва “Broadcast receivers” за получаването на “Intent” - и (“implicit Intent”)

Втора глава

Изисквания към програмния продукт, описание на структурата на програмата и базата данни

2.1. Изисквания към програмния продукт

Създаване на мобилно приложение за Android платформата, което да служи за комуникация с Bugzilla сървъри. Приложението трябва да може поддържа повече от един акаунт, тоест да може да се свързва с повече от един Bugzilla сървър. Комуникацията между приложението и сървъра да се осъществява посредством "REST" програмния интерфейс, чрез изпращане на различни "HTTP" заявки. Информацията, която се връща от сървъра, да бъде разпределена по такъв начин, че да бъде ясна за потребителя и нейната визуализация да бъде представяна в удобен за него интерфейс.

2.2. Избор на езика за програмиране и софтуерните средства

За създаването на приложението са използвани мобилната операционна система "Android", езикът "JAVA" и средата "Eclipse". Това е така поради три причини. Първо, "JAVA" е официалният език, на който се разработват Android приложения под "Eclipse" платформата, която освен, че предлага набор от различни инструменти, които улесняват работата на програмиста, е и официаната среда за създаване на програми, работещи под Android. Второ, вече две години голяма част от дейността ми в програмирането е свързана с разработването на софтуертни продукти под "JAVA" и "Eclipse", като тук се включват и Android приложенията. Опитът в тази област ми помага да съсредоточа цялото си внимание над архитектурата за решаване на дадения проблем, като това води до по – голяма производителност, което е

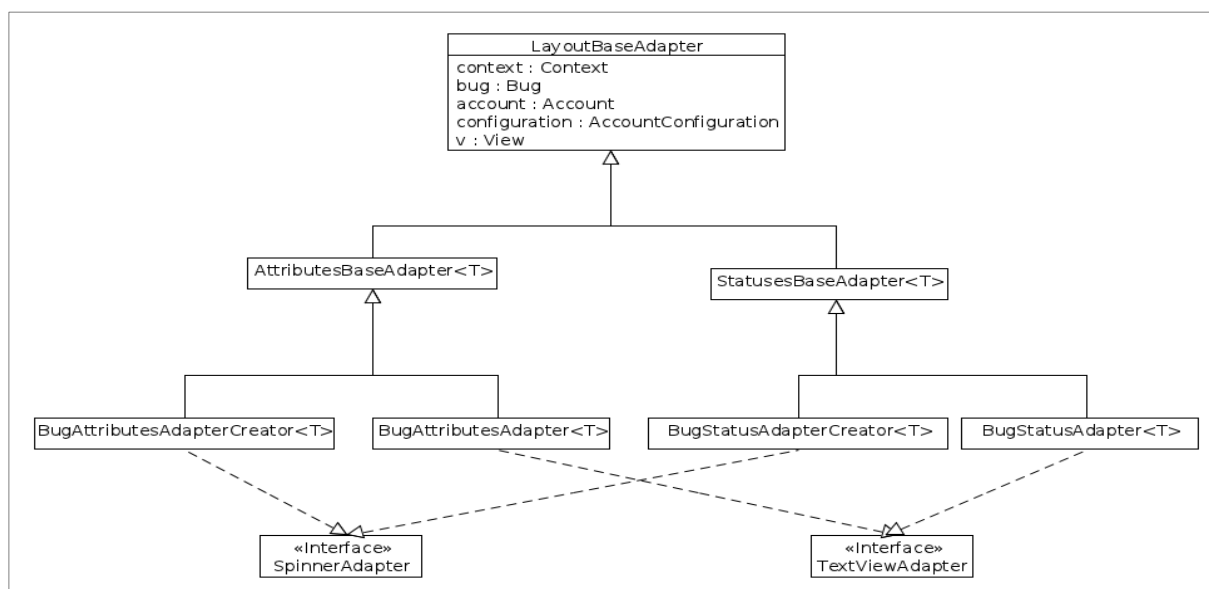
важна част от задачите на програмиста. И трето, желанието ми за бъдещето ми развитие в технологичния свят е свързано със създаването на различни системи за Android, тъй като това е една от най – бързо развиващите се мобилни операционни системи, с наистина големи възможности, което я прави и една от предпочитаните среди, както за крайните потребители, така и за различни компании, лидери в технологичния свят.

2.3. Структура на приложението

Приложението е разделено на осем пакета, като класовете във всеки пакет отговарят за определени проблеми.

2.3.1. Адаптери (“adapters”)

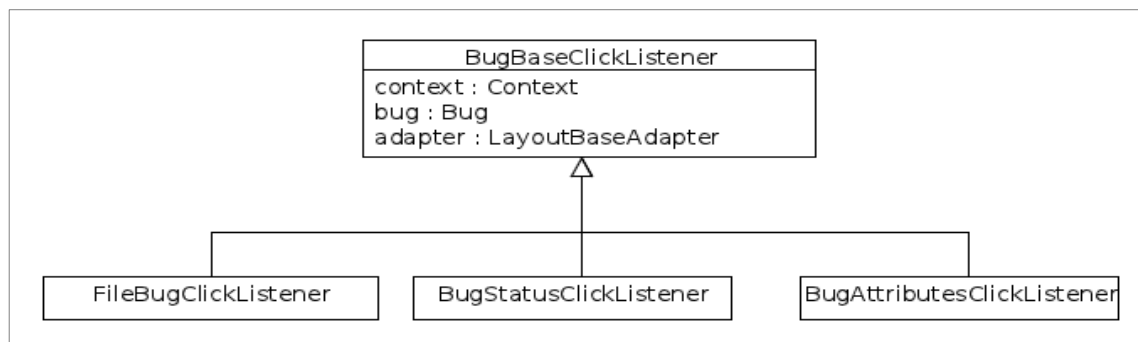
Класовете в този пакет служат за инициализиране на потребителския интерфейс. Главните класове в този клас са йерархично свързани.



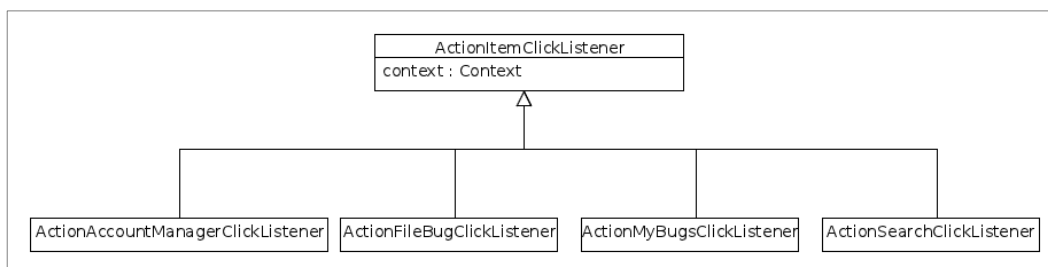
Фиг. 2.1. Йерархия от адаптери за представяне на подробната информация за буга

2.3.2 Слушатели ("listeners")

Класовете в този пакет служат за регистриране на различни събития по време на програмата. Те са пряко свързани с адаптерите. За главните класове в пакета има изградена йерархия.



Фиг. 2.2. Йерархия от слушатели за редактиране и създаване на бъл



Фиг. 2.3. Йерархия от слушатели за бутоните на главния екран

2.3.3. Пакетът "org.elsys"

В този пакет се съдържат всички Activity класове на приложението.

2.3.4. Пакетът "org.elsys.data"

В този пакет се съдържат всички класове, които представляват логическите единици, с които работи един Bugzilla сървър. Това са

класовете "Bug", "Account", "AccountConfiguration", "Attachment", "Comment", "Creator", "Product" и "Status"

2.3.5. Пакетът "org.elsys.dialogs"

Тук се съдържат двата персонализирани диалогови прозореца, с които приложението работи - "SearchDialog" и "LoginDialog"

2.3.6. Пакетът "org.elsys.parser"

Тук се намира класът "ParseJsonStream", отговорен за разбор ("parse") на данните, които се свалят от сървъра

2.3.7. Пакетът "org.elsys.requests"

В този пакет се намират всички класове, които служат за комуникация с Bugzilla сървъра.

2.3.8. Пакетът "org.elsys.utilities"

В този пакет се намират всички класове, които имат по – общи функции.

2.4. Структура на базата данни

За база данни приложението използва така наречените "SharedPreferences". "SharedPreferences" физически всъщност представляват един "XML" файл. Информацията в тях се записва под формата на Map<String, ?>, където ключът винаги трябва да е "String", а стойността може да бъде всякакъв примитив. Извличането на данни става посредством ключа, а записът на данни става чрез класа SharedPreferences.Editor. Самото приложение използва този вид услуга, за да запише информацията за акаунтите на потребителя и конфигурациите към тези акаунти. Информацията е разделена на три части. В първата част се записват само данните за акаунта, като за ключ се използва идентификационният номер на акаунта. Във втората част се записват само конфигурациите, като за ключ отново се използва идентификационният номер на акаунта. И в третата

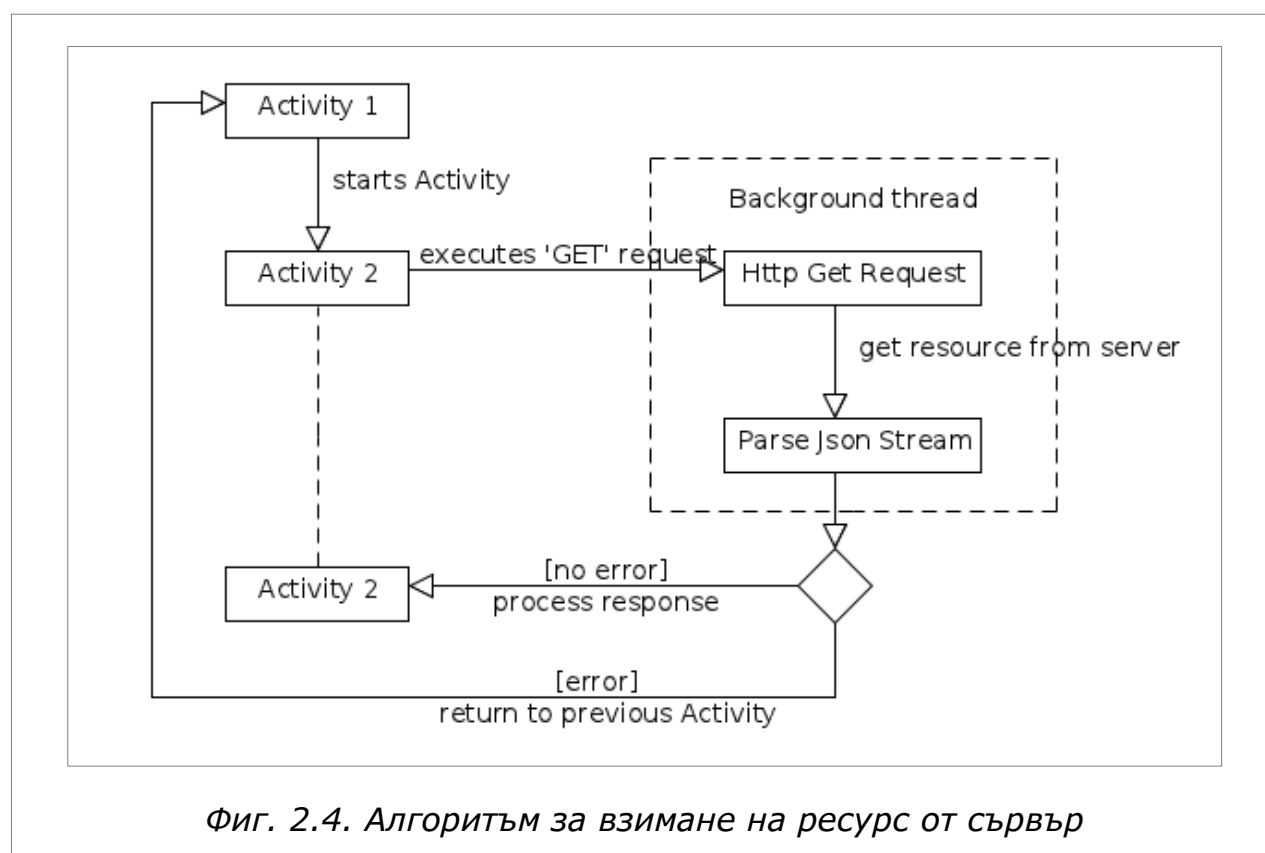
част, която е най – малка, се записва единствено идентификационният номер на текущият акаунт. По този начин, когато искаме да извлечем текущия акаунт, то взимаме информацията от последната част, която ни предоставя идентификационния номер на текущия акаунт и по този номер извличаме цялата информация за акаунта от първата част, след което по същия номер извличаме и конфигурацията за този акаунт.

2.5. Осъществяване на заявки към сървъра

Приложението следва определен алгоритъм за изпълнение на различните заявки към сървъра и определяне на резултата.

2.5.1. "GET" заявки

Стъпките, през които минава приложението от натискане на определен бутон, стартиращ заявка, до презентиране на отговора, са показани на фиг. 2.4.

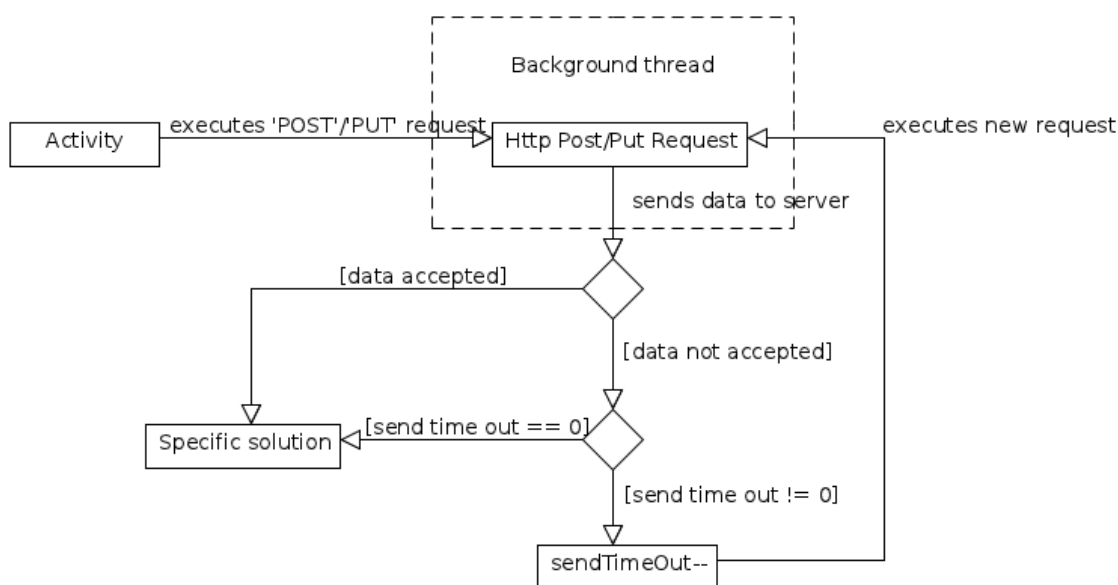


Основната идея на алгоритъма може много лесно да бъде обяснена чрез пример от самото приложение, като за такъв ще се използва поредицата от операции, необходими за показване на листа от бъгове.

Нека предположим, че потребителят се намира на главния екран на приложението, като Activity – то, което стои зад този екран, е "Main" класът и този потребител иска да му бъде показан листът с бъгове, на които той е създател. Потребителят натиска "My bugs" бутона и избира опцията "Reporter". След избирането на тази опция се стартира новото Activity, което се грижи да покаже листа с бъгове, тоест класът "BugList". Още при самото инициализиране на "BugList" в метода "onCreate" се създава нов обект от "SearchRequest" класа, в чийто конструктор се изчислява точният адрес на заявката и се изпълнява самата заявка в нова нишка, като по този начин приложението остава отзивчиво ("responsive"). През времето за изпълнението на заявката "BugList" класът не извършва никакви операции. След като заявката върне отговор, се проверява неговият статус и ако той означава, че необходимите данни са намерени, те се парсват чрез класа "ParseJsonStream". Ако прочитането на данните е било успешно, променливата "error", декларирана в класа "BugzillaHttpRequest" като "true", става равна на "false". След края на изпълнението на заявката и парсването на данните приложението се прехвърля отново в "UI" нишката на приложението, където се проверява стойността на "error". Ако тя е "true", се връщаме в първото Activity, тоест в "Main" класа, тъй като е възникнала някаква грешка. Ако обаче тя е равна на "false", се извиква "callback" методът ((BugList) context).processData(bugs) на второто Activity, тоест на "BugList" класа, в който данните се представят по необходимия начин.

2.5.2. "POST" / "PUT" заявки

Поредицата от операции, необходими за извършването на този вид заявки, се различава в зависимост от специфичната заявка, която може бъде: "BugPostRequest", "BugPutRequest" и "CommentPostRequest". Основният алгоритъм за самото изпълнение на заявките обаче е общ за всички такива. Той е показан на фиг. 2.5.



Фиг. 2.5. Качване на ресурс на сървър

Отново идеята на алгоритъма ще бъде обяснена с помощта на пример от самото приложение, като за такъв ще се използва операциите, необходими за създаване на нов бър.

Нека предположим, че потребителят се намира на екрана за създаване на нов бър, като Activity – то, което стои зад този екран, е "FileBugDetails" класът. При натискане на бутона "Предаване на бър" ("Submit bug") се създава нов обект от "BugPostRequest" класа, който стартира самата заявка в нова нишка, като по този начин приложението остава отзивчиво ("responsive"). Заявката се изпълнява и се проверява отговорът, който тя връща, като ако той удовлетворява типа на заявката променливата "error" става равна на "false", тоест данните, които се пращат, са приети от сървъра. Ако обаче отговорът на заявката уведомява за определен тип грешка, то "error" си остава равна на "true", тоест данните, които се пращат, не са приети по някаква причина от страната на сървъра. Ако те не са приети, се прави проверка, дали променливата "sendTimeout", която първоначално е инициализирана със стойност "2", е различна от нула. Тази променлива указва броят опити за извършване на заявката. Ако е различна от нула, то тя се намалява с едно, като по този начин броят на опитите намалява с един. След което

автоматично се стартира нова заявка от същия тип и пак се правят същите проверки. Ако и след последния опит за изпращане на данните, "error" е останала "false", то данните, които се пращат, не са приети от сървъра.

Трета глава

Описание на начина на реализация на приложението

Ще разгледаме програмата, описвайки работата, която бива извършвана, във всеки един от екраните (Activity класовете) на приложението, които са шест на брой – “Main”, “AccountManager”, “BugList”, “BugComprehensiveView”, “FileBugChooseProduct” и “FileBugDetails”. Преди обаче това да стане трябва да се спомене начинът, по който се прехвърля информация между отделните екрани, тоест между отделните Activity класове. За тази цел служи класът “App”, който наследявайки Application прави възможно досъпването му от всички Activity класове, които са част от това приложение. Класът App по същество представлява един “singelton” клас, като по този начин има само една негова инстанция и информацията между екраните остава непроменена и еднаква за всички. Класът App има няколко полета. Първото е самата инстанция на App, която е част от “singelton” архитектурата. Второто е текущият акаунт. Удобното при наследяването на Application е, че този клас App автоматично се инициализира при стартирането на приложението, което ни позволява още при самото стартиране на програмата от SharedPreferences да се извлече текущият акаунт, ако има такъв. Третото поле е бъгът, който се прехвърля между BugList и BugComprehensiveView.

3.1. Екрани на приложението

3.1.1. Main

Началната точка на програмата е класът Main.class, който наследява BugzillaActivity (виж т. 3.2.1.). Този клас е зададен като стартиращ в Manifest файла на приложението. Най – отгоре на екрана се намира Action Bar – а. (виж т. 3.2.2. Action Bar). След това

имаме четири бутона, които са основните навигационни средства. За всеки бутон слагаме по един listener (виж т. 3.2.3. `ActionListener`). На този екран при натискане на Android бутон за меню излиза меню с една или две опции. Това става чрез имплементиране на метода `onPrepareOptionsMenu(Menu menu)`, където на подадения аргумент `menu` извикваме метод за добавяне на опция, като на този метод се подават като основни аргументи името на опцията и иконката за нея. Ако потребителят има текущ акаунт, менюто ще има две опции – за опресняване на конфигурацията на акаунт (`Refresh configuration`) и опцията `Относно` (`About`). Ако потребителят няма текущ акаунт, менюто ще има единствена опция `Относно`. Регистрирането на избор на някоя от опциите става в метода `onOptionsItemSelected(MenuItem item)`. Проверката за това коя опция е избрана става по името на самата опция, чрез извикване на метода `item.getTitle()`. Ако избраната опция е `Относно`, то се показва `AlertDialog`, който дава информация за приложението. Ако избраната опция е за опресняването на конфигурацията, то се изпълнява заявка за изтегляне на конфигурацията (виж т. 3.2.6.1.1. `AccountConfigurationRequest`).

3.1.2. AccountManager

Този екран се грижи за всички операции, отнасящи се до създаване, редактиране и изтриване на акаунти. Основната част от екрана е заемана от листа с наличните акаунти, който се зарежда чрез извикването на метода `loadList()`. Този метод от своя страна взима всички записани в приложението акаунти (виж т. 3.2.4. Запис на акаунти и конфигурации) и ги подава на адаптера `AccountManagerAccountsAdapter`, който попълва листа. Всяка клетка от листа представлява един акаунт, като за всеки такъв се показва

описанието на акаунта и адреса на Bugzilla сървъра. За редакцията и изтриването на даден акаунт се грижи класът "QuickAction", който използва много известна тенденция в Android, а именно QuickAction. Този клас е част от външна библиотека и работата му се състои в това при продължително задържане на някой акаунт да се покаже меню с две опции - "Edit" и "Delete". Двете опции представляват инстанции на "ActionItem", които биват добавяни към QuickAction. Всеки ActionItem е уникален по своя идентификационен номер, по който по - късно се разпознава, за да се разбере коя опция е избрана. Ако избраната опция е за изтриване на акаунт се показва обикновен AlertDialog със запитване към потребителя, дали акаунтът да бъде премахнат. Ако потребителят избере да изтрие акаунта се извикват съответните методи от AccountPersistor и AccountConfigurationPersistor за изтриване на данни. Тук трябва да се вметне, че на самия лист се добавя и "listener" за дълъг натиск ("OnItemLongClickListener"), с помощта на който се извлича от листа този акаунт, който трябва да се редактира. Това се налага, тъй като в listener-а на QuickAction няма налична възможност да се вземе този акаунт, а той ни трябва, тъй като за изтриването на акаунта в AlertDialog-а ни трябва неговият идентификационен номер, а за редактирането на акаунта пък ни трябва целият акаунт (виж т. 3.2.1. Добавяне и редакция на акаунти).

3.1.2.1. Добавяне и редакция на акаунти

Добавянето на акаунт става чрез бутона "Добави акаунт" ("Add account") в екрана AccountManager. Натискането на този бутон води до отваряне на диалоговия прозорец LoginDialog. Когато искаме да редактираме акаунт също се отваря този прозорец. LoginDialog има два конструктора в зависимост от това какво ни трябва. Единият конструктор се използва, когато искаме да добавяме акаунт, а

другият добавя като аргумент Account и се използва в случаите, когато искаме да редактираме. LoginDialog –ът има няколко текстови полета, които потребителят трябва да попълни. Тези текстови полета всъщност представляват полетата на класа Account, без това за идентификационния номер. Ако обаче сме в случая, когато редактираме акаунт, тези текстови полета се попълват автоматично от данните на акаунта, който трябва да редактираме. Когато потребителят е готов с данните и натисне бутона “Добави” (“Add”) се извиква методът addAccount(), който извлича всички попълнени данни и проверява, дали са въведени задължителните полета за описание на акаунта и адрес на Bugzilla сървъра. Прави се също така проверка за имейла и паролата. Трябва или и двете полета да са попълнени, или и двете полета да са празни. Ако всичко е наред и сме в режим на редактиране на акаунт се изпълнява заявка към Bugzilla сървъра за изтегляне на конфигурацията. Ако пък сме в режим на добавяне на акаунт, трябва преди да се направи заявката да се генерира идентификационен номер за новосъздадения акаунт, което става просто чрез инкрементиране на акаунта с най – голям идентификационен номер. (виж т. 3.2.6.1. AccountConfiguration заявка)

3.1.3. BugList

Това Activity показва листа с намерените бъгове. До него може да се достъпи по два начина. Единият начин е през SearchDialog – а (виж т. 3.1.3.1. SearchDialog), а вторият начин е през бутона “My bugs” в главното меню. Идеята на BugList е, че този екран се зарежда още преди стартирането на заявката за търсене, като по този начин веднага след като заявката приключи листът с бъгове се попълва. Това е установена практика в Android. Но за да стане това, трябва BugList по някакъв начин да разбере, че заявката е приключила и

това става чрез "callback" метода **void** processData(List<Bug> bugs), който се извиква от SearchRequest заявката (виж т. 3.2.6.1.2. SearchRequest) след като тя приключи. Ако няма намерени бъгове вместо лист се показва "TextView" със съобщение, че няма намерени бъгове. Ако обаче има намерени бъгове, то листът се попълва с тях благодарение на "BugListAdapter" – а. При избор на някой бъг той се записва в App класа, за да може да се прехвърли към следващото Activity, след което се инициализира екранът BugComprehensiveView. Тук бе възможно бъгът да не се записва в App класа, а да се прехвърли чрез Intent – а, който стартира BugComprehensiveView, но се оказва, че Intent – ите има ограничение по отношение на това какво количество информация може да пренасят. При бъгове, съдържащи малко информация няма проблеми, но ако бъгът има много коментари или прикачени файлове приложението може да блокира.

3.1.3.1. SearchDialog

Това е диалоговият прозорец, през който се правят прости търсения към Bugzilla сървъра. Той може да бъде отворен чрез който и да е бутон, изобразен чрез лупа. Този прозорец има няколко полета. Първото е статусът на бъговете, за които да търси, като на всеки статус съответстват определени стойности. Например на статуса "Open" съответстват "UNCONFIRMED", "NEW", "ASSIGNED", "REOPENED", като тези стойности се съхраняват в конфигурацията на акаунта. Второто поле показва всички налични продукти, които се взимат от конфигурацията, а в последното поле се попълва ключова дума за търсенето.

3.1.4. BugComprehensiveView

Този екран представя датайлна информация за бъга, като неговите коментари, прикачени файлове и различни атрибути. И този

екран също като BugList се показва преди още преди да е била изпратена заявката за детайлна информация за бѳга (виж т. 3.2.6.1.3. BugDetailsRequest), като чак след като заявката приключи своята работа се извиква "callback" методът ((BugComprehensiveView) context).processResponse(), който зарежда екрана с информацията за бѳга. За работата на екрана са използвани две външни библиотеки. Едната е официално предоставена за Android и се нарича android.support.v4.view.ViewPager. Тя служи за преминаване между отделните табове на BugComprehensiveView чрез плъзгането на прѳста наляво и надясно. Втората библиотека се нарича "viewpagerindicator" и служи за това различните табове да имат от горната страна заглавия и за преминаването от един таб на друг чрез избирането на тези заглавия. Самият екран се зарежда чрез "ViewPagerAdapter" адаптера, който зарежда табове на екрана, като се инициализират само текущият таб (този, на който потребителят се намира в момента) и тези табове, които се намират от лявата и дясната страна на текущия таб, като по този начин се оптимизира скоростта на показване на данните.

Табове обаче се различават в зависимост от няколко фактора. Първо, ако потребителят, използващ приложението не е въвел своя имейл и парола в данните за акаунта, то той няма да може да коментира бѳга, тъй като тази секция от "XML", която съдържа полето за добавяне на коментар за него няма да съществува. Това става, чрез добавяне на параметър visibility = View.GONE. Ако обаче потребителят има необходимите данни, за да може да създава коментар след като натисне бутона "Изпрати" ("Send") се изпраща заявка към сървъра, която добавя коментара към бѳга (виж т. 3.6.2.2.3. CommentPostRequest). Второ, секциите "Attributes" и "Status" имат два типа изглед в зависимост от това, дали създадеят на разглеждания бѳг е потребителят на текущия акаунта и дали този

потребител е въвел своя имейл и парола, като трябва и двете условия да са едновременно изпълнени. Ако е така, то това значи, че потребителят на приложението има възможността да редактира бѐга. В този случай полетата, които показват данните за бѐга, се показват под формата на падащи менюта (в Android този вид полета се наричат "Spinner"), с помощта на които могат да се нанасят промени. Но ако не е така, то потребителят на приложението не трябва да има възможност да редактира бѐга и всички данни за него се показват под формата на текстови полета (TextView). Проверката, която определя кой от двата типа изгледи ще се покаже се прави във ViewPagerAdapter адаптера, където в зависимост от резултата на проверката за всеки от двата таба "Attributes" и "Status" ще се използват или двойка адаптери за възможност за редакция на бѐг или двойка адаптери за липса на такава възможност. Адаптерите са параметризирани, като параметърът може да бъде или TextView или Spinner в зависимост от случая. Двойката адаптери за възможност за редакция на бѐг са BugAttributesAdapterCreator<T> и BugStatusAdapterCreator<T>, а двойката адаптери, при които липсва такава възможност са BugAttributesAdapter<T> и BugStatusAdapter<T>. При използването на първата двойка адаптери на екрана (BugComprehensiveView) има добавен и бутон за запазване на промените, като при неговото натискане се взима цялата информация за бѐга, независимо дали е променяна или не и стартира заявка за редактиране на бѐга (виж т. 3.2.6.2.1. BugPutRequest).

3.1.5. FileBugChooseProduct и FileBugDetails

До този екран може да се достигне чрез избиране на бутона за регистриране на бѐг ("File bug") от главното меню, но при условие, че има текущ акаунт и потребителят е въвел своите имейл и парола за този акаунт. Екранът представлява разширяем лист

(ExpandableListActivity) с наличните продукти, които се съхраняват в конфигурацията на акаунта, като при избор на някой продукт, се показват наличните компоненти за този продукт. При избор на някой продукт се отваря екранът FileBugDetails, като в Intent - а, който стартира този екран се записва името на продукта, който е избран. Записва се само името на продукта, а не целият продукт, за да се праща по – малко количество информация през Intent – а. По това име на продукта от конфигурацията на акаунта се намира целият обект на продукта, чийто данни се изпращат заедно с тези въведени от потребителя за бѳга (виж т. 3.2.6.2.2. BugPostRequest).

3.2. Основни компоненти на приложението

3.2.1. BugzillaActivity и общо Activity класа

Всички екрани, на които се намира потребителят, представляват отделни Activity класове или подкласове. Чрез тях той получава информация и въвежда такава. Един Activity клас има няколко различни метода, които биват извиквани автоматично при инициализирането на класа. Най – често използваният такѳв метод е onCreate(Bundle bundle), който се извиква първи в цикѳла на живот на едно Activity и където се слага необходимият код за инициализирането му, като например избиране на начина, по – който ще изглежда това Activity чрез метода setContentView(int id). За работата на приложението е създаден класът BugzillaActivity, който наследява Activity. Така повечето класове, които трябва по принцип пряко да наследяват Activity, всъщност наследяват BugzillaActivity. Това е така по две причини. Първата причина ще бѳде обяснена, когато се разглежда точката за заявките към сървърѳа. Втората причина е, че за работата на всяко едно Activity, тоест на всеки отделен екран, се изисква обект от класа Account. Ето защо вместо този обект да бѳде представен като поле на всеки отделен клас, то имаме само едно

такова поле в BugzillaActivity. Проблемът е в това, че това поле се инициализира само веднъж в метода onCreate(Bundle bundle), а този метод не се извиква никога повече в жизнения цикъл на едно Activity, освен ако то не бъде унищожено. Следователно, щом този метод се извиква само веднъж, ако по време на работата на приложението потребителят изтрие акаунта или пък по някаква причина GC унищожи обекта от класа Account към който сочи препратката (тоест полето) , то в зависимост от случая тази промяна или няма да се отрази на полето или полето ще започне да сочи към null и при опит за достъп до елементите на това поле ще възникне "NullPointerException". За да се предпазим от това, просто в метода onResume() на класа BugzillaActivity взимаме текущия акаунт и го присвояваме на полето. По този начин винаги това поле ще е актуализирано, а ако пък няма текущ акаунт (тоест е null), то извикваме метода returnToPreviousActivity(), дефиниран в класа BugzillaActivity.

3.2.2. Action Bar

Action Bar – ът е един от най – известните стилове, свързани с потребителския интерфейс, в разработката на Android приложения. В приложението той е имплементиран като част от "XML" файла, който определя интерфейса за всеки екран (виж т. 3.2.1. Интерфейс). Представлява един "LinearLayout", в който се дефинират няколко "ImageButton" – а. Удобното при него е, че на всеки "ImageButton" може още на ниво "XML" да се каже какво да стане при натискането на този бутон. Това става чрез тага "onClick", на който като стойност му се дава име на метод, който по – късно трябва да имплементираме. Например в повечето случаи в Action Bar – а има бутон за търсене, който е под формата на лупа. На този бутон му слагаме таг android:onClick:"onActionBarSearchClick". По този начин във всяко

Activity, където имаме този Action Bar, просто трябва да създадем метод `public void onActionBarSearchClick(View v)`, който се извиква при натискането на бутона за търсене. По този начин кодът става по – опростен , тъй като няма нужда да слагаме “listener” – и за тези бутони.

3.2.3. ActionItemClickListener

Всеки един бутон от главния екран трябва при натискането си да извърши определени операции. Но преди да се извършат тези действия, трябва да бъде преминато през определени проверки, като например това дали имаме текущ акаунт или пък дали има въведено потребителско име и парола за този акаунт. И вместо за всеки един бутон отделно да се правят тези проверки има много по лесен начин за това. За целта служи класът “ActionItemClickListener”, който имплементира интерфейса “OnClickListener” и който има един метод `onClick(View v)`. Като аргумент на този метод се предава елементът (View), който е натиснат. В този метод се прави проверка, дали имаме текущ акаунт чрез извикване на “Utilities.isAccountSelected”. Резултатът от това извикване се записва като таг на елемента, който е натиснат. Тук трябва да се вметне, че всеки елемент си има таг, в който могат да се записват различни данни, като в случая записваме резултата, който е от тип “boolean”. Записването става чрез извикване на метода “setTag(Object ob)” на класа View, от какъвто тип е и натиснатият елемент. За всеки бутон от главния екран има дефиниран “listener”, който всъщност е подклас на класа ActionItemClickListener. В този подклас методът “onClick” е предефиниран, като в началото му се извиква същият метод от класа родител (чрез извикването на `super`). По този начин проверката за наличност на акаунт става не във всеки отделен подклас, а само на едно място – в родителския. (извикването на `super` не става единствено за бутона,

водещ до екрана за мениджмънт на акаунти, тъй като за него за него не е нужен текущ акаунт.) След извикването на `super`, се взима записаният таг и ако той удовлетворява подкласа, то работата на метода `onClick` продължава.

3.2.4. Запис на акаунти и конфигурации

Първоначалната идея за съхранение на акаунтите и на конфигурации към тях беше те да бъдат записани в база данни SQLite. Но след щателно проучване установих, че по – добре ще е да използвам друг похват за запис на данни в Android, а именно “`SharedPreferences`”. Причините за това са две - първо, потребителят на приложението едва ли ще има повече от един до два акаунта. В този ред на мисли работата на базата данни не е най – доброто решение, тъй като информацията, която ще бъде записвана е прекалено малко. Второ, достъпът до `SharedPreferences` е по – бърз, отколкото до базата от данни, а до записаната информация се достъпва често. `SharedPreferences` обаче имат един доста сериозен недостатък – данните, които се записват в тях не са сигурни. Тоест някой, който има желание, може да достъпи до тях. Ето защо се наложи преди да записвам данните те да бъдат криптирани (виж т. 3.2.5. Криптиране на данните). За самия запис на акаунтите и конфигурациите се грижат класовете `AccountPersistor` и `AccountConfigurationPersistor`.

3.2.4.1. AccountPersistor

Този клас се грижи за записа и четенето на акаунти от `SharedPreferences`. Класът има няколко публични метода, с които останалата част от приложението работи. Това са методите `persistAccount(Context context, Account account)`, който се грижи за записа на акаунта в `SharedPreferences`, `Account getPersistedAccount(Context context, int`

accountId), който връща като резултат акаунта с идентификационен номер accountId, List<Account> getPersistedAccounts(Context context), който връща лист от всички записани акаунти, persistCurrentAccount(Context context, int accountId), който записва идентификационния номер на акаунта, който е текущ, Account getCurrentPersistedAccount(Context context), който връща като резултат акаунта, който е текущ и void deleteAccount(Context context, int id), който изтрива акаунта с дадения идентификационен номер (id).

3.2.4.2. AccountConfigurationPersistor

Този клас се грижи за записа и четенето на конфигурации към акаунтите от SharedPreferences. Той има няколко метода, които са фундаментални - void persistConfiguration(Context context, int id, AccountConfiguration configuration), където int id е идентификационният номер на акаунта, към който принадлежи тази конфигурацията, AccountConfiguration getConfiguration(Context context, int accountId), където int accountId е идентификационният номер на акаунта и void removeConfiguration(Context context, int id), където int id отново е идентификационният номер на акаунта.

3.2.5. Криптиране на данни

За криптирането на данните се грижи класът "SimpleCrypto". Той има два публични метода - String encrypt(String clearText) и String decrypt(String encrypted). "clearText" аргументът представлява текстът, който трябва да се енкриптуе, а "encrypted" аргумента на втория метод представлява енкриптираният текст, който трябва да се декриптира. За криптирането на данните се използва "AES" алгоритъмът. Преди да върнем енкриптирания текст от първия метод, той преминава през "Base64" кодировка, което води до условието преди да декриптираме декриптирания текст от втория метод той

трябва да се премине през Base64 декодировка. Същинската част на криптирането е в методите `byte[] encrypt(byte[] raw, byte[] clear)` и `byte[] decrypt(byte[] raw, byte[] encrypted)`, където се използват класът "Cipher" и класът "SecretKeySpec", който приема като аргумент на своя конструктор аргумента "raw", който от своя страна представлява тайният ключ за криптирането. Този таен ключ се връща от метода `byte[] getRawKey()`, който използва зададения "seed" за неговото генериране.

3.2.6. Заявки към сървъра

Главният клас, който се грижи за изпълняването на заявките е "BugzillaHttpRequest". Тъй като заявките са задачи, които се изпълняват за по – дълъг период от време ("long running tasks"), ако се опитаме да направим заявка от главната (UI) нишка на програмата то след някакъв период от време операционна система ще отчете, че приложението не отговаря ("not responding") и ще предложи на потребителя то да бъде затворено. Ето защо се налага този вид операции да се изпълняват на отделна нишка. Но в Android това е много улеснено, тъй като за тази цел е създаден класът "AsyncTask", който позволява да се изпълняват операции на заден фон, а резултатите от тези операции да се публикуват на "UI" нишката. Класът AsyncTask притежава няколко метода, като само "doInBackground" е задължителен. Този метод се изпълнява на нишка различна от UI нишката и затова в него не трябва да се опитваме да променяме интерфейса. Той всъщност казва какво трябва да се върши на заден фон, в случая със самото приложение трябва да се изпълнява заявката и да се обработят резултатите. Освен него AsyncTask има още няколко метода, като в приложението се използват "onPreExecute", който се изпълнява преди да се влезе в doInBackground и се изпълнява на UI нишката. В конкретния случай с

BugzillaHttpRequest в този метод правим проверка, дали има устройството има интернет връзка. Следващият метод е "onPostExecute", който се изпълнява след като doInBackground приключи и също се изпълнява на UI нишката. Има още един метод "onCancel", който се изпълнява, когато се опита да прекъснем изпълнението на заявката (тук трябва да се вметне, че изпълнението на "POST" и "PUT" заявките не може да бъде прекъсвано). Изпълнението на заявката може да бъде прекъснато поради няколко причини. В "BugzillaHttpRequest" това може да стане, ако в метода "onPreExecute" се установи, че устройството няма интернет връзка. Останалите причини са обяснени в следващите точки. Всеки един от тези четири метода се предефинира от подкласовете на "BugzillaHttpRequest", в зависимост от нуждите на самия клас. "AsyncTask" е параметризиран клас с три параметъра. Първият параметър е типът на аргумента, който се подава на doInBackground. В случая се подава "String". Вторият параметър е типът на аргумента, който се подава на метода "onProgressUpdate" и тъй като този метод не се използва в приложението, вторият параметър е "Void". Третият параметър е типът на резулата, който трябва да върне "doInBackground". В случая не връщаме нищо от този метод и затова и третият параметър е "Void". BugzillaHttpRequest включва в себе си някои основни метода, които подкласовете му трябва да притежават. Такива операции са конструирането на точния адрес за заявката чрез метода `URL constructUrl(String param)` и основното и най – важното – самото отваряне на връзка към сървъра, с помощта на която можем да четем и пишем. Отварянето на връзката става с помощта на метода **void** `openConnection(String param)`.

```
protected void openConnection(String param) throws Exception {  
  
    url = constructUrl(param);  
  
    trustAllHosts();  
}
```

```
System.setProperty("http.keepAlive", "false");  
  
https = (HttpsURLConnection) url.openConnection();  
  
https.setHostnameVerifier(DO_NOT_VERIFY);  
  
}
```

Фиг. 3.1. Отваряне на "HTTP" връзка към сървъра

Основен проблем, който срещнах при отварянето на връзка беше, че за да се осъществи комуникация с Bugzilla сървъра, трябва да се използва "HTTPS" протоколът, а той изисква хостът да бъде сертифициран. Идеята на самото приложение обаче ни ограничава в това, което и наложи решението, просто всички хостове да се считат за сигурни. Тъй като приложението трябва да изпълнява, както "GET", така и "POST" и "PUT" заявки, то BugzillaHttpRequest се разделя на два подкласа съответно – "BugzillaHttpGetRequest" и "BugzillaHttpPostPutRequest".

3.2.6.1. BugzillaHttpGetRequest

Този клас се грижи за изпълнението на "GET" заявките към Bugzilla сървъра. При изпълнението на всяка заявка, на която в конструктора на класа е подаден аргумент String showMessage различен от "null", в onPreExecute създаваме нов "ProgressDialog", който уведомява потребителя, че се извършва някаква дейност. Съобщението за ProgressDialog – а всъщност е аргументът "showMessage". Едно от най – големите предизвикателства на приложението беше именно това как да се осъществяват "GET" заявките. Имайки опит в други Android приложения, кореспондиращи си със сървъри, повечето пъти просто трябваше в doInBackground да заявката да се изпълни, резултатът да се върне като String в някакъв формат, например "JSON" или "XML" и да се подаде на onPostExecute, където в зависимост от класа наследник, който предефинира

onPostExecute, този резултат да се прочете по някакъв начин. Но в предишните проекти, резултатът като физически размер на данните беше много малък, което за мобилно устройство е добре. В това приложение обаче има случаи, когато размерът на данните от Bugzilla сървъра е прекалено голям, за да бъде прочетен целият в паметта на телефона и след това чак да се вземе тази информация от резултата, която е нужна. Ето защо се наложи да конструирам архитектурата по такъв начин, че паметта на устройството да не се претоварва. Тук взех предвид две неща. Първо, за връзка с Bugzilla сървъра аз използвам "HttpsURLConnection", от която мога да взема при четене на данни входен поток ("InputStream") чрез метода [https.getInputStream\(\)](#). По този начин няма нужда да прочитам цялата информация. Вторият фактор е, че форматът на резултатите, които връща Bugzilla сървъра, когато се използва REST програмният интерфейс, е "JSON". Взимайки под внимание тези две неща, просто трябваше да намеря такъв "JSON" парсър, който да може да чете данни от входен поток, тоест да поддържа "Streaming API". И след доста прегледани такива парсъри, се оказа, че "Jackson" е най – добрият такъв. Но за да мога да използвам този вид четене на данни от входен поток, потокът трябва да бъде отворен, следователно четенето на данни трябва да стане в doInBackground метода, тъй като единствено там можем да си позволим да оставим този поток отворен. Но четенето на данните трябва да бъде специфично за всеки подклас на BugzillaHttpRequest, тъй като веднъж ще се прочита конфигурацията за акаунта, друг път листа с бъгове и данните от сървъра трябва да се обработят по различен начин. Това би било лесно в "onPostExecute", който да бъде предефиниран във всеки подклас, но "onPostExecute" се изпълнява в UI нишката, където ако четем от входен поток приложението ще блокира. Ето защо четенето трябва да става в "doInBackground",

```
@Override

    protected Void doInBackground(String... params) {

        try {

            openConnection(params[0]);

            https.setRequestProperty("Accept", "application/json");

            if (https.getResponseCode() == HttpURLConnection.HTTP_OK) {

                readJsonStream(https.getInputStream());

            } else {

                return null;

            }

        } catch (Exception e) {

            e.printStackTrace();

            return null;

        } finally {

            if (https != null) {

                https.disconnect();

            }

        }

        return null;

    }
```

Фиг. 3.2. Осъществяване на връзка със сървъра и прочитане на върнатата информация

но по такъв начин, че да е специфично за всеки подклас. И за да стане това, просто трябваше да създам един абстрактен метод **abstract void** processStream(JsonParser jParser), който се извиква от readJsonStream(https.getInputStream()) (фиг. 3.3.) и се имплементира от всеки подклас, което прави възможно специфичното парсване на данни.

```
private void readJsonStream(InputStream in) throws IOException, JsonParseException {  
  
    JsonFactory jFactory = new JsonFactory();  
  
    JsonParser jParser = null;  
  
    try {  
  
        jParser = jFactory.createJsonParser(in);  
  
        if (jParser != null) {  
  
            processStream(jParser);  
  
        }  
    } finally {  
  
        if (jParser != null) {  
  
            jParser.close();  
  
        }  
    }  
  
    error = false;  
  
}
```

Фиг. 3.3. Създаване на парсър от входния поток за прочитане на данните

abstract void processStream(JsonParser jParser) приема един аргумент JsonParser jParser, който се конструира от метода, описан във фиг. 3.3. и служи за парсване на данните (виж т. 3.2.7. Парсване на данни). Тук обаче идва друг проблем. "OnPostExecute" винаги ще се изпълни след като doInBackground приключи, независимо дали четенето на данни е било успешно или не. Ето защо в главния клас BugzillaHttpRequest съществува поле **boolean** error = **true**, което указва, дали е възникнала грешка при четенето на данните или не. Тази променлива в началото е "true" и чак ако четенето на данните е било успешно, тя става "false". След това в "onPostExecute" се проверява, дали има грешка или не и ако няма грешка програмата продължава своя нормален ход на работа.

3.2.6.1.1. AccountConfiguration заявка

Тази заявка служи за две неща. Първо, заявката се изпълнява при опит да се добави акаунт, или да се редактира вече съществуващ такъв. Тоест чрез нея се прави проверка, дали на дадения адрес на сървъра има реално съществуващ Bugzilla сървър и дали останалите данни на акаунта като имейл и парола са вярни. И второ, ако всичко е наред със сървъра и данните на акаунта, то необходимата информация за конфигурацията на дадения Bugzilla сървър се изтегля и се създава нов обект AccountConfiguration. Тази заявка може да бъде изпълнена от две места – или от LoginDialog – а, когато искаме да добавим нов акаунт или да редактираме стар, или от Main класа, когато искаме да опресним конфигурацията. При успешно свързване към сървъра, данните се прочитат и ако всичко е наред, информацията за акаунта и конфигурацията се записват чрез класовете AccountPersistor и AccountConfigurationPersistor, след което LoginDialog – ът се затваря и се връщаме на главния екран на приложението. Ако обаче е възникнала грешка при четенето на данните или при свързването към сървъра (тоест `error = true`) следва, че опитът да добавим или редактираме акаунт е неуспешен. Ето защо извикваме метода `((BugzillaActivity)context).returnToPreviousActivity()`, който ако сме направили заявката от LoginDialog – а прави текущ акаунт записаният като текущ (ако има такъв) преди изпълнението на заявката.

3.2.6.1.2. SearchRequest

Тази заявка служи за изтегляне на бъгове по определени критерии. Трябва да се отбележи, че при нея не се прочита цялата информация за намерените бъгове, а само тази част от информацията, която се показва в BugList, като по този начин се оптимизира скоростта на четене и се икономисва от памет. Заявката

има метод `String computeRequest(Bundle extras)`, който служи за изчисляване на нужните параметри на самата "GET" заявка, които се определят в зависимост от това по какъв начин е била стартирана, дали чрез бутона "My bugs" или през `SearchDialog` – а. Аргументът `Bundle extras` включва данните за заявката, които после обаче трябва да бъдат подредени по определен начин. След като заявката приключи ако всичко е наред, се извиква "callback" методът `((BugList) context).processData(bugs)`, чийто аргумент представлява прочетените бъгове.

3.2.6.1.3. BugDetailsRequest

Тази заявка сваля всичката допълнителна информация за бъга, която се показва в `BugComprehensiveView`. Когато заявката приключи своята работа, ако не е възникнала грешка се извиква "callback" методът `((BugComprehensiveView) context).processResponse()` на класа `BugComprehensiveView`, където свалените данни се показват на потребителя. Тук трябва да се отбележи, че ако потребителят на приложението е създател на бъга и ако потребителят е въвел своите имейл и парола в данните на акаунта, то Bugzilla сървърът ще добави към информацията на бъга още едно поле "update_token", което по – късно трябва да се добави към данните, които се изпращат към сървъра, при опит да се редактира този бъг. По този начин Bugzilla се уверява, че този, който редактира бъга има правото да го прави.

3.2.6.2. BugzillaHttpPostPutRequest

Този клас служи за изпълнението на "POST" или на "PUT" заявки към Bugzilla сървъра. Класът обединява двата типа заявки, тъй като разликата между двата вида е единствено в един ред код – `https.setRequestMethod(typeOfRequest)`, където `typeOfRequest` представлява вида на заявката: "PUT" или "POST".

Този клас има едно много важно поле. Това е променливата `static boolean isInProgress = false`, която показва, дали в момента има изпълняваща се заявка, като в началото тя е `false`, тъй като няма изпълняващи се заявки. Идеята на тази променлива трябва да се обясни с пример. Да кажем, че потребителят направи някакви промени по някой от параметрите на бъг, който е създал преди това, и иска да запази промените. За да може обаче тези промени да се запазят на Bugzilla сървъра, на него трябва да му се изпрати заедно с променените параметри и поле `"update_token"`, което преди това е получено чрез заявката за самия бъг. По този начин сървърът знае, че даденият потребител има правото да прави промени по бъга. Въпросът е в това, че когато се направят промени по бъга, за него се генерира автоматично нов `"update_token"`. Ето защо веднага след като бъгът със стария `"update_token"` е изпратен, трябва да се свали новогенерирания `"update_token"`, тоест веднага след като се изпълни по – долу описаната заявка `"BugPutRequest"`, трябва да се изпълни и `"BugTokenRequest"` и чак след това потребителят да имат правото да прави нови промени по бъга. Но потребителят може да не изчаква и втората заявка и веднага да се опита да изпрати други промени по бъга, които обаче няма да могат да бъдат запазени на сървъра, тъй като тези нови промени ще бъдат изпратени със стария `"update_token"`, а не с новогенерирания, който се намира на сървъра и все още не е свален. Ето защо преди да се изпрати заявката за промяна на бъга, `isInProgress` става `"true"` и става `"false"`, чак когато новият `"update_token"` е свален.

Основната логика на класа `"BugzillaHttpPostPutRequest"` е, че той има поле `ObjectMapper`, който се грижи за това да направи `"JSON"` стринг от даден обект чрез метода `mapper.writeValueAsString(objectToSend)`, като самият обект, който искаме да трансформираме се предава на конструктора на класа.


```
@Override
```

```
protected Void doInBackground(String... params) {

    OutputStream output = null;

    partOfUrl = params[0];

    try {

        if(dataToSend == null) {

            dataToSend = mapper.writeValueAsString(objectToSend);

        }

        openConnection(partOfUrl);

        https.setDoOutput(true);

        https.setFixedLengthStreamingMode(dataToSend.length());

        https.setRequestMethod(typeOfRequest);

        https.setRequestProperty("Content-Type", "application/json");

        output = https.getOutputStream();

        output.write(dataToSend.getBytes());

        output.flush();

        output.close();

        int status = ((HttpsURLConnection) https).getResponseCode();

        if (status == HttpURLConnection.HTTP_CREATED

            || status == HttpURLConnection.HTTP_ACCEPTED) {

            error = false;

        }

    } catch(JsonGenerationException jGE) {

        showEndOfRequestMessage();

    } catch(JsonMappingException jME) {

        showEndOfRequestMessage();

    } catch (Exception e) {

        e.printStackTrace();

    } finally {
```

```
        if (https != null) {  
            https.disconnect();  
        }  
    }  
    return null;  
}
```

Фиг. 3.4. Осъществяване на връзка към сървъра и изпращане на информация към него

След това се прави самата заявка към сървъра и ако отговорът, който ни се върне означава, че всичко е наред, то променливата `error` става равна на `false`, тоест всичко е наред със заявката. Независимо обаче от стойността на `error` винаги се отива в "onPostExecute", където се прави проверка, дали заявката е била успешна. Ако не е била, то се автоматично се изпраща нова заявка към сървъра, като това се прави с цел, ако в момента на първата заявка сървърът не е имал възможност да я приеме, то при втората може да успее. Общият брой на опитите, които се правят за изпращане на информацията са три на брой. Ако и след последния опит няма положителен отговор от страна на сървъра, то се показва съобщение, че качването на информацията не е било успешно.

```
        if (error && sendTimeout != 0) {  
            sendTimeout--;  
            isInProgress = false;  
            new BugzillaHttpPostPutRequest(context, showMessage, typeOfRequest,  
                                           sendTimeout, dataToSend).execute(partOfUrl);  
        } else {  
            showEndOfRequestMessage();  
        }
```

Фиг. 3.5. Проверка за резултата от заявката

3.2.6.2.1. BugPutRequest

С тази заявка се прави промяна на параметрите на разглеждания бър. След като заявката е приключила, ако всичко е наред се изпраща нова заявка BugTokenRequest, която сваля новия "update_token" на бър, тъй като той се е променил. След като тя приключи, ако е била успешна се изпраща нова заявка за коментарите на бър, тъй като в случай на промяна на резолюцията на бър на "DUPLICATE", Bugzilla сървърът автоматично добавя един нов коментар към бър и съответно коментарите за бър трябва да се обновят. След като и тази заявка приключи, се извиква методът ((BugComprehensiveView) [context](#)).notifyDataChange(), който казва на BugComprehensiveView да се обнови, тъй като има променена информация за бър.

3.2.6.2.2. BugPostRequest

С тази заявка се създава нов бър. Интересното при нея е, че още методът onPreExecute е предефиниран, като в него се създава нов Intent, който да върне потребителя на главното меню още преди изпращането на информацията, така че той да може да продължи своята работа.

3.2.6.2.3. CommentPostRequest

С тази заявка се добавя нов коментар към дадения бър, като ако след изпълнението на заявката всичко е наред трябва да се изпрати нова заявка BugTokenRequest, която сваля новия "update_token" на бър, тъй като той се е променил. След което се извиква методът ((BugComprehensiveView) [context](#)).notifyDataChange(), който казва на BugComprehensiveView да се обнови, тъй като има променена информация за бър.

3.2.7. Парсване на данни

До последните дни за парсване в приложението беше използвана библиотеката "Gson". Същата библиотека поддържа и много удобен начин даден обект директно да бъде преобразуван в JSON. Проблемът бе в това, че в класа "Bug" има някои полета, които не са примитиви, а обекти от други класове, като например класът "Product". Този клас има няколко полета като име на продукт и име на компонент. При качването на нов бъг обаче или при редактирането на някой такъв, форматът на JSON, който се изпраща изисква полетата на класа Product всъщност да са полета на Bug, за да може сървърът да приеме заявката. Тоест трябва полетата на класа Product да станат част от полетата на класа Bug чрез разграждането на обекта Product на неговите полета. При "Gson" обаче това е невъзможно и се налагаше за всички полета на класа Product директно в кода да бъде направено копие на тях в класа Bug, което значеше да имам две еднакви полета в два различни класа, което като дизайн на приложението не е добре. Ето защо се наложи да потърся алтернативно решение за библиотека за "JSON". След много проучвания открих такава на име "Jackson", която поддържа точно функционалността, която ми бе необходима и единственото, което бе нужно да направя, за да разградя класа Product на неговите полета бе да сложа една анотация над полето Product @JsonUnwrapped. "Streaming" програмният интерфейс на тази библиотека за сметка на това обаче, бе много по – трудно реализируем, отколкото при предишната. Въпреки това, според различни статистики на "JSON" парсъри "Jackson" е най – бързата и най – малко изискваща памет възможна библиотека, което значи, че смяната от "Gson" към "Jackson" си е заслужавала, тъй като при едно мобилно приложение тези два фактора са от изключително значение.

Както се споменава в точка 3.2.6.1. парсването става в метода, **abstract void** processStream(JsonParser jParser), където се извикват нужните

статични методи на класа "ParseStreamJson". Като аргумент на горния метод се подава JsonParser jParser, тъй като този клас поддържа парсването на данни от поток. Класът ParseStreamJson съдържа методи, които са характерни за различните типове заявки, като например за заявката AccountConfigurationRequest е нужен методът от ParseStreamJson, който да ни върне нов обект AccountConfiguration, а именно ParseStreamJson.readAccountConfiguration(jParser). Идеята на JsonParser -а е, че данните са разделени на така наречените "token" - и, като за един такъв "token" се счита всичко освен кавичките. Основният цикъл, който се използва за четене на данните във всеки един от методите е показан на фиг. 3.6.

```
while (jParser.nextToken() != JsonToken.END_OBJECT) {  
    ...  
}
```

Фиг. 3.6. Проверка за резултата от заявката

За JsonToken.*END_OBJECT* се счита затварящата къдрава скоба ("}").

3.3. Основни методи и класове, които са заложиени в Android операционната система

За работата на приложението се използват няколко основни термина от Android, които са общи.

3.3.1. Интерфейс

Всички екрани за едно Android приложение представляват отделни Activity класове. Но тези класове казват единствено какво трябва да става при работа с приложението, а не дефинират начина, по който изглеждат екраните. За интерфейса на приложението се грижат файловете в директорията "/res/layout". Всеки файл в тази

папка представлява един "XML" документ, който следва определени правила, и който определя начина, по който ще изглежда даденият екран. За да стане това, в началото на метода `onCreate(Bundle bundle)` на всяко `Activity`, се извиква методът `setContentView(int resId)`, като аргументът на който се подава името на "XML" файла, който искаме да използваме. На всеки елемент ("TextView", "ImageView" и други) от "XML" файла, към който искаме да имаме достъп от това `Activity`, трябва да сложим таг за идентификационен номер ("android:id"), като по този номер в `Activity` – то, можем да намерим определения елемент, което ни дава възможност да извършваме различни манипулации с него.

3.3.2. Преминаване между екрани

Като такива за връзка между тях се използват така наречените `Intent` – и, чрез които става преминаване от един `Activity` клас към друг, тоест от един екран на друг. Конструкторът на един `Intent` приема два аргумента – контекста на текущото `Activity`, което е текущият екран и втори аргумент класът, наследяващ `Activity`, към който искаме да се насочим, тоест вторият екран.

```
Intent intent = new Intent(Context currentActivity, Class<?> destinationActivity);
```

За да стартираме преминаването на текущото `Activity` се извиква МЕТОДЪТ `startActivity(intent)`.

Четвърта Глава

Ръководство на потребителя

4.1. Изисквания и инсталация на приложението

За работата на приложението е нужно наличието на Android устройство - смартфон или таблет с версия на операционната система 2.2 и нагоре. При наличие на такова следва софтуерният продукт да бъде свален от онлайн пазара за Android приложения, където е дистрибутирано, а именно Google Play. Имаме две опции за изтегляне на приложението - директно от устройството или през интернет сайта на Google Play. И двата начина изискват потребителя да има активен Google акаунт.

4.1.1. Инсталация на приложението през устройството:

1. Отивате в "Настройки" ("Settings") -> "Акаунти и синхронизация" ("Accounts & sync").

2. Проверявате, дали вече имате съществуващ Google акаунт. Ако вече имате такъв преминете на стъпка десет (10), ако не изберете бутона "Добави акаунт" ("Add account").

3. Изберете "Google"

4. Изберете "Следващ" ("Next")

5. Тук имате две възможности - да създадете Google акаунт, ако нямате такъв ("Създаване" / "Create"), или да изберете да се впишете, ако имате Google акаунт ("Вписване" / "Sign in"). Ако все още нямате акаунт изберете първата опция и се регистрирайте, следвайки инструкциите. След регистрацията преминете направо на стъпка десет (10). При наличието на вече съществуващ акаунт изберете втората опция

6. Тук трябва да въведете вашето потребителско име и парола, които сте използвали при създаването на акаунта. След като сте готови изберете "Вписване" ("Sign in")

Внимание! За да може вашето устройство да се свърже с Google сървърите и да направи потвърждение на вашият акаунт е нужно устройството да има налична интернет връзка. Ако устройството в момента на потвърждение няма такава, то ще се опита да се свърже, след което ще изведе съобщение, че нямате връзка ("You don't have a network connection."). За да включите "Wi-Fi" и да успеете да осъществите такава, изберете бутона по - долу ("Connect to Wi-Fi"). Изберете налична мрежа и натиснете бутона "Назад" ("Back"), след което е нужно отново да въведете своята парола и да натиснете "Вписване" ("Sign in")

7. След осъществяване на връзката може да избере кои типове данни да се синхронизират между устройството и вашият акаунт. За добавяна на опция просто избере празното квадратче до Вашия избор.

Когато сте готови натиснете "Приключи" ("Finish").

8. Показва се съобщение, че Google акаунтът Ви вече е свързан с устройството ("Your Google account is now linked to this phone"). Когато сте готови натиснете "Приключи настройките" ("Finish setup").

9. Сега вече сте готови да изтеглите приложението. Върнете се в началния екран на Вашето устройство.

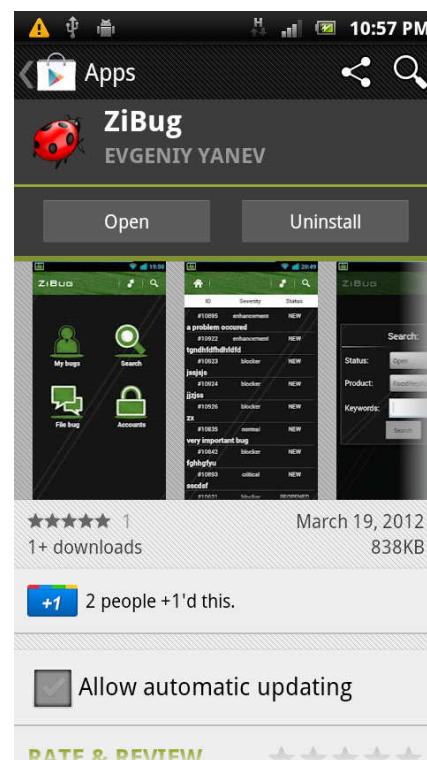
10. Избирате приложението "Play Market", след което в горния десен ъгъл изберете лупата. Отваря се поле за писане, където трябва посредством клавиатурата да въведете името на приложението - "ZiBug" (без кавичките), като името може да бъде въведено и с малки букви. След като въведете името, натиснете отново лупата (или тази на самата клавиатура, или тази в горния десен ъгъл).

11. Избирате това приложение, под име "ZiBug"

12. Под името на приложението от дясната страна има бутон, на който пише "Инсталирай" ("Install"). Изберете го.

13. Прочетете какви са изискванията ("Permissions") на приложението. Ако не одобрявате някое от изискванията, можете да се откажете от свалянето на приложението, но при съгласие за сваляне се счита, че одобрявате всички изброени изисквания. Когато сте прочели изискванията и сте решили, че ще свалите приложението, натиснете бутона "Приемам & свалям" ("Accept & download").

14. Под името на приложението ще се покаже прозорец, който Ви уведомява за прогреса на сваляне. След като приложението се свали, то автоматично се инсталира. Когато всичко е готово, под името на приложението ще се появят два бутона - "Отвори" ("Open") и "Изтрий" ("Uninstall"). Ако в определен момент решите, че не желаете повече този продукт на устройството си можете да се върнете тук и да изберете "Изтрий" ("Uninstall").



4.1.2. Инсталация на приложението през браузър:

1. Отворете браузъра, който използвате.
2. В полето за въвеждане на адреса на сайта, който искате да посетите, въведете "https://play.google.com/store" (без кавичките).
3. След като страницата зареди, проверете какво пише на бутона в горния десен ъгъл. Ако пише "Вписване" ("Sign in") изберете бутона.
4. Тук имате две възможности – ако все още нямате Google акаунт изберете бутона "Регистриране" ("Sign up") и създайте своя

акаунт, след което отново отидете на адрес "https://play.google.com/store" и преминете към точка пет (5). Ако вече имате Google акаунт, просто въведете Вашето потребителско име и парола и ще бъдете автоматично прехвърлени към предходната страница.

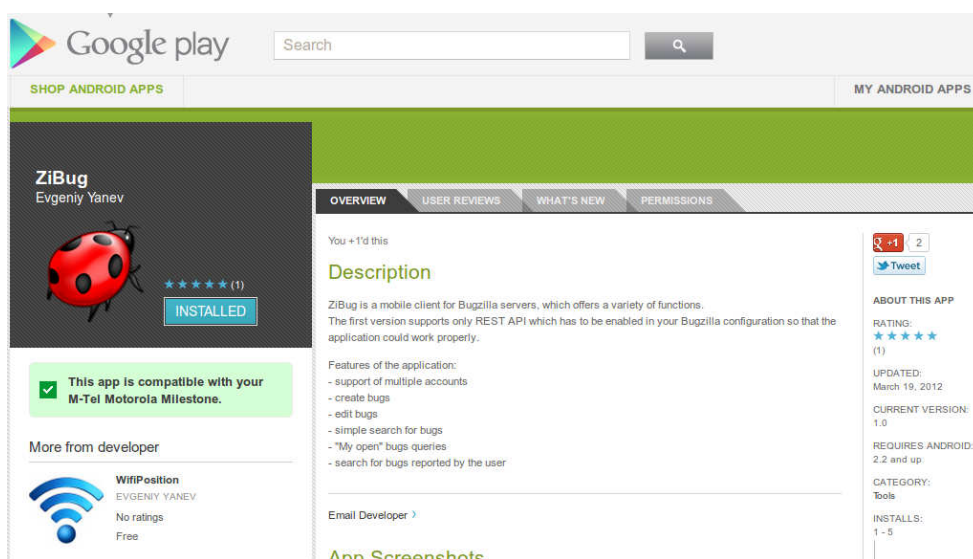
5. След като сте готови, в горната дясна част на страницата има лупа и поле до нея, в което пише "Търсене" ("Search"). Натиснете на полето и въведете името на приложението - "ZiBug" (без кавичките), като името може да бъде и с малки букви. След това натиснете или "Enter" на клавиатурата или лупата.

6. Намерете приложението под търсеното име - "ZiBug". От дясната страна на изображението, което характеризира приложението, има бутон "Инсталиране" ("Install"). Изберете го

7. От падащото меню до надписа, който пише "Изпрати до..." ("Send to...") изберете устройството, на което искате да бъде инсталирано приложението.

8. Под разделителната ивица са изписани всички изисквания на приложението. Ако не одобрявате някое от изискванията, можете да откажете да свалите приложението и да натиснете бутона "Изход" ("Cancel") в долния ляв ъгъл, но при съгласие за сваляне се счита, че одобрявате всички изброени изисквания. Когато сте прочели изискванията и

сте решили, че ще свалите приложението натиснете бутона "Инсталиране" ("Install").



Внимание! Приложението няма да може да се свали, ако устройството, на което се опитвате да го инсталирате, няма интернет достъп. При липса на такъв приложението ще се свали първият път, когато на устройството се осигури достъп до интернет.

Поздравления! Вече имате инсталирано приложението на Вашето устройство.

4.2. Работа с приложението

Работата с приложението може условно да бъде разделена на четири части – мениджмънт на акаунти, търсене и редакция на бъгове и създаване на нов бъг.

Внимание! За работата на приложението, е нужна свързаност на устройството към интернет. В случай, че нямате такава и се опитате да предприемете действие, за което е нужно устройството да комуникира с Bugzilla сървъра, ще се покаже съобщение, че в момента няма налична интернет връзка ("No internet connection right now. Please try again later").

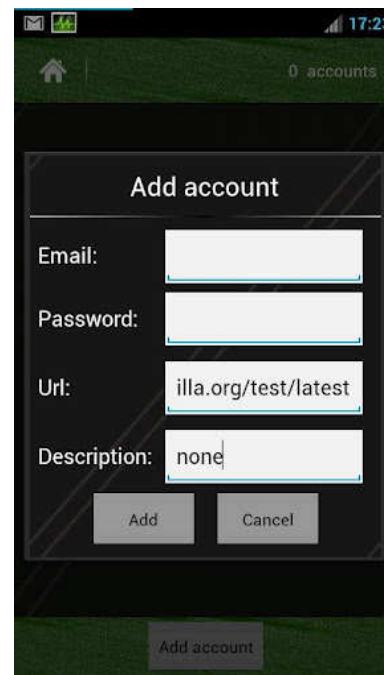
4.2.1. Мениджмънт на акаунти

4.2.1.1. Добавяне на акаунт

При първоначалното стартиране на приложението потребителят се намира на главния екран на приложението. Тук са позиционирани четири главни бутона - "Мойте бъгове" ("My bugs"), "Търсене" ("Search"), "Качване на бъг", ("File bug") и "Акаунти" ("Accounts"). При опит да се избере някоя от опциите ще се покаже кратко съобщение, че не е избран акаунт ("No account selected"). Това съобщение няма да се покаже единствено при избиране на опцията "Акаунти" ("Accounts"), като до нея може да се достъпи по два начина - или чрез бутона, който е под формата на катинар и под него

има надпис "Accounts", или чрез бутона, който се намира в горната част на екрана, непосредствено от лявата страна на лупата.

При избиране на един от двата метода се прехвърляме на екрана за създаване, изтриване или редактиране на акаунти. В горния десен ъгъл на този екрана се намира поле, което индексира броят на наличните акаунти, като в началото там бива изписано "0 акаунта" ("0 accounts"), тоест няма никакви добавени акаунти. Под този текст имаме лист от акаунти, като в началото той ще е празен. Ако потребителят иска да добави нов акаунт, трябва да избере бутона в долната част на екрана ("Добави акаунт" / "Add account"). При натискането му се показва



диалогов прозорец с четири полета - "Имейл" ("Email"), "Парола" ("Password"), "УРЛ", ("URL") и "Описание" ("Description"). Полетата "URL" и "Description" са задължителни, за да можем да добавим акаунт, докато другите две "Email" и "Password" са опционални, като се попълват само при наличието на съществуващ акаунт за сървъра на Bugzilla, чийто адрес се попълва в полето "URL". Тук трябва да се вметне, че ако тези две полета се оставят празни, потребителят, използващ приложението, ще има единствено възможността да търси бъгове, но не би могъл да редактира бъгове, нито да създава нови такива. В полетата "Email" и "Password" се въвеждат съответно имейла и паролата на Bugzilla акаунта, който се използва за идентификация с Bugzilla сървъра. В полето "URL" се въвежда адресът на сървъра, а в полето "Description" кратко описание на акаунта, чрез което той ще може лесно да се разпознава. В случай, че потребителят промени решението си и иска да се откаже от добавянето на акаунт, има възможност за връщане назад или чрез натискане на бутона за връщане на Вашето Android устройство, или

чрез натискане на бутона "Отменяне" ("Cancel"). Ако обаче потребителят иска да добави акаунт, след като е въвел всички необходими данни, трябва да натисне бутона "Добави" ("Add"). Устройството ще се опита да направи връзка с Bugzilla сървъра и да сваля неговата конфигурация. Ако възникне някакъв проблем по време на свалянето на необходимите данни, то ще се покаже кратко съобщение, уведомяващо , че в момента устройството не може да се свърже към сървъра ("Unable to contact server")*. Ако обаче всичко е наред, приложението ще се прехвърли на главния екран.

4.2.1.2. Редактиране на акаунт

Ако по време на работа, потребителят реши, че трябва да промени някои от настройките на акаунта, това може много лесно да стане. За да направи това, първо, трябва да се отиде на екрана за мениджмънт на акаунти. След това, от листа с наличните акаунти, трябва да се намери този акаунт, който потребителят желае да промени, и да задържи пръста си на него. Ще се покаже меню с две възможности - "Редактиране" ("Edit") и "Изтриване" ("Delete"). Избира се първата опция. Ще се покаже диалогов прозорец, попълнен с наличните данни за акаунта. Тук можете да промените данните или да добавите нови. След като сте готови с избора си, натиснете бутона "Редактирай" ("Edit"). Устройството ще се опита да се свърже с Bugzilla сървъра, за да провери валидността на въведените данни. При възникване на грешка ще се покаже съобщение, че връзката със сървъра не е възможна ("Unable to contact server")*. Ако обаче всичко е наред, ще бъдете върнати на главния екран.

*Този вид грешка може да се дължи на две причини. Първо, в момента, когато се опитвате да свалите данните, е възможно

сървърът, с който се опитва да комуникира устройството да не е в изправност. Втората причина за проблема е, че сте въвели грешни данни за акаунта, като адрес на сървъра , имейл или парола. Проверете данните си още веднъж и в случай, че отново се покаже същото съобщение, изчакайте известен период от време преди отново да опитате, тъй като вероятно проблемът е от страната на сървъра.

4.2.1.3. Изтриване на акаунт

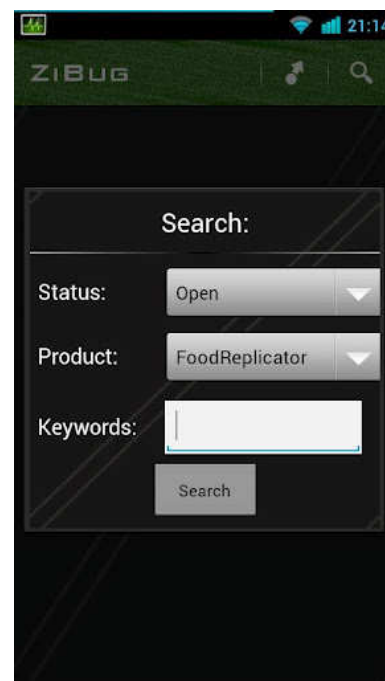
Ако по време на работа, решите, че искате да премахнете някой от Вашите акаунти, можете много лесно да го направите. За да направите това, първо трябва да отидете на екрана за мениджмънт на акаунтите. След това, намерете от листа с наличните акаунти този акаунт, който желаете да промените и задръжте пръста си на него. Ще се покаже меню с две възможности - "Редактиране" ("Edit") и "Изтриване" ("Delete"). Изберете втората опция. Ще се покаже диалогов прозорец, който Ви пита, дали искате да изтриете избрания акаунт ("Do you want to delete this account?"). Ако наистина искате да изтриете този акаунт, изберете бутона "Да" ("Yes") и тогава акаунтът ще бъде изтрит и премахнат от списъка с наличните акаунти. Ако изберете "Не" ("No"), ще се върнете на екрана за мениджмънт на акаунти, без да бъде направена никаква промяна.

4.2.2. Търсене и редакция на бъгове

Търсенето на бъгове може да бъде разделено на два типа – просто търсене на бъгове и търсене на бъгове, свързани с потребителя.

4.2.2.1. Просто търсене на бъгове

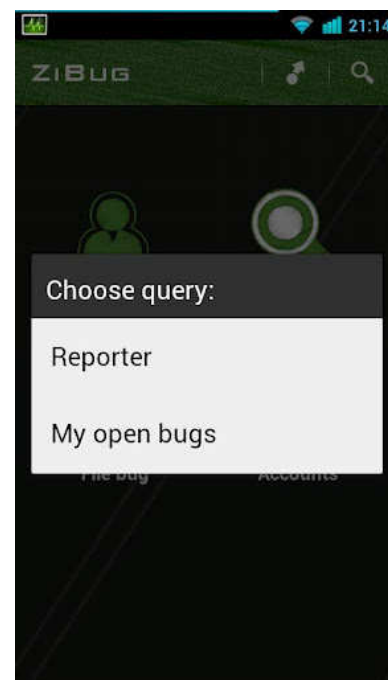
За да извършите този тип търсене, първо трябва да имате съществуващ акаунт. Този тип търсене може да бъде извършено, както от началния екран на приложението, избирайки бутона във формата на лупа, под който пише "Търсене" ("Search"), така и от всеки друг екран, в чийто горен, десен ъгъл има поставена лупа, която също може да бъде избрана. Който и метод да използвате, ще бъде показан диалогов прозорец, със заглавие "Търсене" ("Search"). В този диалогов прозорец са изредени критериите, по които можете да направите Вашето търсене. Първата опция е статусът на бъга. Той може да бъде "Отворен" ("Open"), "Затворен" ("Closed"), както и "Всички" ("All"), който комбинира, първите два избора. Следващата опция показва продукта на бъга. Тук Ви се изреждат всички съществуващи продукти за Вашата Bugzilla конфигурация. Третата опция е ключова дума, по която да търсите. След като сте попълнили избраните от Вас критерии натиснете бутона "Търсене" ("Search").



4.2.2.2. Търсене на бъгове, свързани с потребителя

За да извършите този тип търсене, трябва както да имате съществуващ акаунт, така и да сте въвели потребителско име и парола за този акаунт. Този тип търсене може да бъде извършено само от началния екран на приложението, избирайки бутона "Моите бъгове" ("My bugs"). След избирането на този бутон се появява диалогов прозорец с две опции. Първата опция е "Докладчик" ("Reporter"), при избирането на която приложението ще търси за тези бъгове, за които потребителят е докладвал.

Втората опция се нарича "Моите отворени бъгове" ("My open bugs") и при нейното избиране приложението ще търси за онези бъгове, на които потребителят е или създател или пълномощник ("assignee"), като още едно условие при търсенето е, че ще бъдат изведени само онези бъгове, чийто статус е отворен, тоест още не са разрешени.

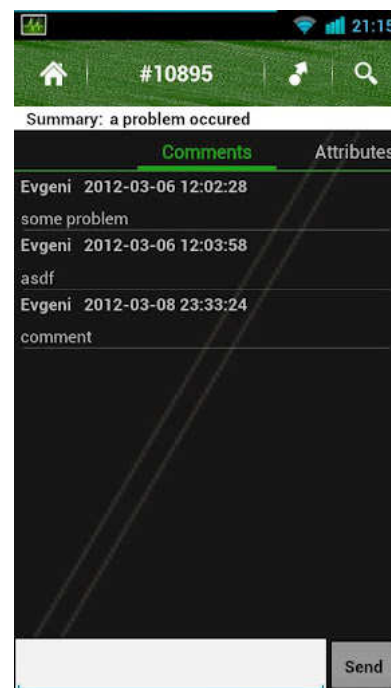


И при двата вида търсене, след като съответният метод на търсене е избран, ще се покаже диалогов прозорец, на който пише "Търсене" ("Searching"). След като диалоговият прозорец изчезне, ако няма намерени бъгове ще се покаже надпис "Нула бъгове намерени" ("Zarro Boogs found"). Ако обаче има намерени бъгове, ще се зареди лист с намерените бъгове. Всяка клетка от листа е един намерен бъг, като за всеки бъг се показва само най – важната информация за него – неговият идентификационен номер ("ID"), сериозността на бъга ("Severity"), неговият статус ("Status") и неговото обобщение ("Summary"). При избиране на някой бъг, ще се покаже диалогов прозорец, на който пише "Извличане" ("Fetching"), тоест се извлича по – детайлна информация за бъга. След като диалоговият прозорец изчезне, ще се покаже екрана даващ по – детайлна информация за бъга. Най – отгоре на екрана в средата е показан идентификационният номер на бъга. Отдолу е изписано обобщението на бъга ("Summary"). Останалата долна част на екрана е разделена на табове, като предвижването между тях може да стане или чрез плъзгането наляво и надясно на пръста по екрана, или чрез избиране заглавието на желания таб.

4.2.2.2.1. Коментари ("Comments")

Първият таб показва лист с коментарите за бъга ("Comments"). Всяка клетка от листа е един коментар, като за всеки коментар е показан неговият автор, датата, когато е качил своя коментар и самият текст на коментара.

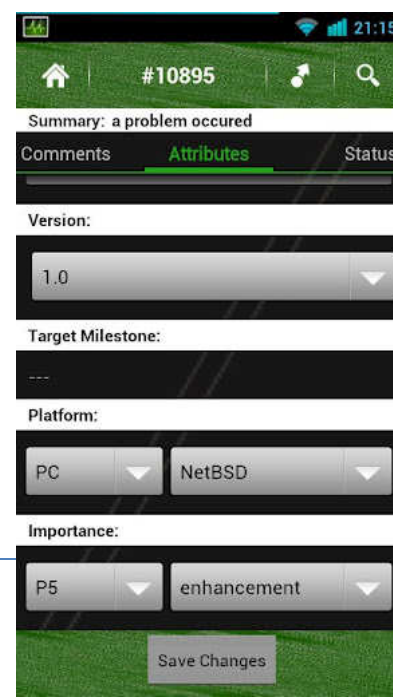
Ако потребителят е въвел в данните на акаунта си своето потребителско име и парола, в най – долната част на този таб ще има възможност да добави нов коментар, като текста на коментара бъде попълнен в текстовото и се натисне бутона "Изпрати" ("Send").



Следващите два таба имат два възможни изгледа - когато потребителят има правото да редактира бъга и когато няма такова право. Разликата между двата е, че при първия изглед, когато потребителят е въвел в данните на акаунта си своето потребителско име и парола и същевременно е създадел на бъга потребителят ще има правото да редактира настройките на съответния този бъг, които ще му бъдат показани под формата на падащи менюта. При втория изглед потребителят няма да има такова право, а настройките на бъга ще му бъдат показани като текст.

4.2.2.2.2. Атрибути ("Attributes")


Този таб дава информация за различни настройки на бъга. На първо място се показват



продукта на бѳга ("Product"), компонентът на бѳга ("Component") и версията на бѳга. "Target Milestone" е следващото поле. След тях следва информация за платформата ("Platform") на бѳга, тоест операционната система и хардуера. Най – последна е информацията за това, с каква степен на важност е този бѳг ("Importance"), като тази информация се разделя на приоритет и суровост на бѳга. В случай, че потребителят има правото да редактира бѳга, най – отдолу на този таб ще има бутон "Запази промените" ("Save Changes"), като при неговото избиране всички направени промени ще бъдат запазени на Bugzilla сървъра.

4.2.2.2.3. Статус ("Status")

Този таб дава информация за статуса на бѳга. На първо място се показва самият статус на бѳга ("Status"). Ако този статусът на бѳга индикира, че бѳгът е разрешен по някакъв начин, следващата опция ще показва решението на бѳга ("Resolution"). Ако решението на бѳга е дубликат ("DUPLICATE"), то ще бъде изписан идентификационният номер на бѳга дубликат. След това се изписва потребителят, който е отговорен за разрешаването на бѳга ("Assigned to"), следван от потребителя, който е докладвал за този бѳг ("Reported by").



The screenshot shows the 'Status' tab for bug #10895. The status is 'RESOLVED' and the resolution is 'DUPLICATE'. The duplicate of is '154'. The assigned to is 'tara@bluemartini.com'. A 'Save Changes' button is at the bottom.

В случай, че потребителят на приложението има правото да редактира бѳга, най – отдолу на този таб ще има бутон "Запази промените" ("Save Changes"), като при неговото избиране всички направени промени ще бъдат запазени на Bugzilla сървъра.

4.2.2.2.4. Прикачени файлове ("Attachments")

Този таб показва лист с прикачените файлове към бъга. Всяка клетка от листа е един прикачен файл, като за всеки такъв е изведена информация за името на прикачения файл, времето на добавяна на файла, типа на файла и потребителят, който е качил самия файл. При продължително натискане на един от прикачените файлове се показва прозорец, който пита, дали прикаченият файл да бъде свален ("Do you want to download attachment?"). При избиране на опцията "Да" ("Yes") прикаченият файл ще бъде свален и записан на паметта на телефона в папка "ZiBug", като за края на свалянето потребителят ще бъде уведомен чрез нотификация, чрез която същият може директно да отвори файла.

4.2.3. Създаване на бъг

За да може потребителят да създаде нов бъг, първо трябва да има съществуващ акаунт и да е въвел потребителско име и парола за този акаунт. След това от началния екран на приложението трябва да избере бутона "Регистрирай бъг" ("File bug"). При избирането му ще се отвори нов екран, на който е показан лист с наличните продукти за дадения акаунт. Избирайки един продукт, ще се покажат като подлист компонентите за дадения продукт. При избирането на даден компонент ще се отвори нов екран, показващ цялата информация за бъга, който ще бъде качен. Това е докладчикът на бъга ("Reporter"), продуктът на бъга ("Product"), компонентът на продукта ("Component"), версията на продукта ("Version"), която при наличие на повече от една ще може да се избира, информация за платформата за бъга ("Platform"), важността на бъга ("Importance"), кратко обобщение на бъга ("Summary") и описание за бъга ("Description"). Полетата, отбелязани със звездичка ("*") пред наименование са задължителни. Най – отдолу на екрана има бутон "Предаване на бъга" ("Submit bug"), при натискането на който бъгът

ще бъде пратен към Bugzilla сървъра и потребителят ще бъде върнат на началния екран. При успешно създаване на бърг ще се покаже съобщение, че бъргът е качен ("Bug uploaded"), ако обаче има грешка при създаването на бърга ще се покаже съобщение, че качването на бърга се е провалило ("Bug failed to upload").

Заключение

Разработеното приложение покрива всички изисквания на даденото задание, като неговата функционалност е разширена и има добавени различни възможности за редакция на бъгове, създаване на нови такива и търсене на бъгове по различни критерии. Софтуерният продукт е пряко приложим в работата на различни организации, като дори вече е достъпен в пазара за Android приложения – Google Play.

Бъдещите версии на програмата предвиждат работа с “WebService” интерфейса за комуникация с Bugzilla сървърите, като по този начин възможностите на крайния потребител ще се увеличат.

Използвана литература

1. Ayushman Jain, Introduction to Android development Using Eclipse and Android widgets, 16.11.2010
2. Features, <http://www.bugzilla.org/features/#searchpage>
3. Barzan 'Tony' Antal, Bugzilla: Open Source Bug-Tracking System, 24.12.2008
4. Lars Vogel, Android Development Tutorial, 06.03.2012
5. Nicolas Zozol, RESTful Web Services, 29.05.2008
6. Roger L. Costello, Building Web Services the REST Way
7. REST (representational state transfer),
<http://searchsoa.techtarget.com/definition/REST>, май 2002
8. Dave Marshall, Remote Procedure Calls (RPC)
9. Bugzilla:REST API, https://wiki.mozilla.org/Bugzilla:REST_API
10. Bugzilla::WebService,
www.bugzilla.org/docs/4.0/en/html/api/Bugzilla/WebService.html
11. Dave Winer, XML-RPC Specification, 30.06.2003