Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ТПУ)

Институт кибернетики
Направление 09.04.01 «Информатика и вычислительная техника»
Кафедра автоматики и компьютерных систем (АИКС)

КУРСОВОЙ ПРОЕКТ
ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»
на тему «Time Tracker»
Пояснительная записка

Студент гр. 8ВМ71 _____ Ю. Ю. Ибетуллов
Студент гр. 8ВМ71 _____ И. В. Пономарев
Студент гр. 8ВМ71 _____ О. А. Рачкован
Студент гр. 8ВМ71 _____ С. А. Черных
«___» _____ 2017 г.

Руководитель:
Инж. кафедры ПИ _____ Д. Н. Лайком
«___» _____ 2017 г.

Оглавление

Введение

Жизнь в современном мире можно очень ёмко охарактеризовать одним словом спешка - высокий ритм жизни, информационная перегруженность, большое количество неотложных дел на решение которых все никак не найдется времени в плотном графике жителя 21 века. Именно поэтому управление временем для современного человека – центральное понятие любой системы личной эффективности и продуктивности.

В жёстком мире капитализма, где время — это деньги и нет ничего личного, работодатель не хочет платить за недоработку, а сотрудник скрупулёзно считает каждый предназначенный ему доллар необходимо оценивать свою ежедневную эффективность.

То, какое количество самого ценного ресурса – времени и на что конкретно тратит его человек, можно определить лишь систематически отслеживая ежедневную активность.

Для оценки личной эффективности, эффективности персонала, для получения полной картины того, как и куда тратится время, широко применяется программное обеспечение классов time tracking software - программное обеспечение автоматизированного учета времени.

Тайм-трекер позволяет вести простой учет времени для личного использования, так и для работы сотрудников. Он может быть полезным для тех, кто работает по гибкому графику или удаленно из дома, а также для распределенных команд, работающих в разных часовых поясах.

1 Требования к программе

1.1 Назначение программы и область применения

Программный комплекс Time Tracker TPU предназначен для того, чтобы авторизированные пользователи могли осуществлять учет и контроль своего личного времени. Программный комплекс состоит их серверной и клиентской части. Пользователь, взаимодействуя с клиентским приложением, имеет возможность создавать задачи, запускать таймер выполнения задачи и останавливать. Также пользователь может группировать задачи по проектам.

Областью применения программного комплекса является как личное использование с целью учета времени, так и использование в корпоративной среде (учет рабочего времени сотрудников).

Перечень функциональных требований к системе:

– система должна предоставлять пользователю возможность входа в систему, если пользователь зарегистрирован;

– если пользователь не зарегистрирован в система, система должна зарегистрировать его при входе;

– система должна предоставлять пользователю возможность авторизации через Google аккаунт;

– система должна предоставлять пользователю возможность учета времени: создание, запуск, остановка задачи;

– система должна предоставлять пользователю возможность редактирования, удаления задач, проектов.

Перечень нефункциональных требований:

– исходный код системы должен располагаться в открытом репозитории GitHub;

– серверная часть системы должна быть реализована на языке программирования Java с использованием новейших технологий построения веб-серверов.

1.2 Варианты использования

Варианты использования, или прецеденты – это способ записи требований (Рисунок 1.1).

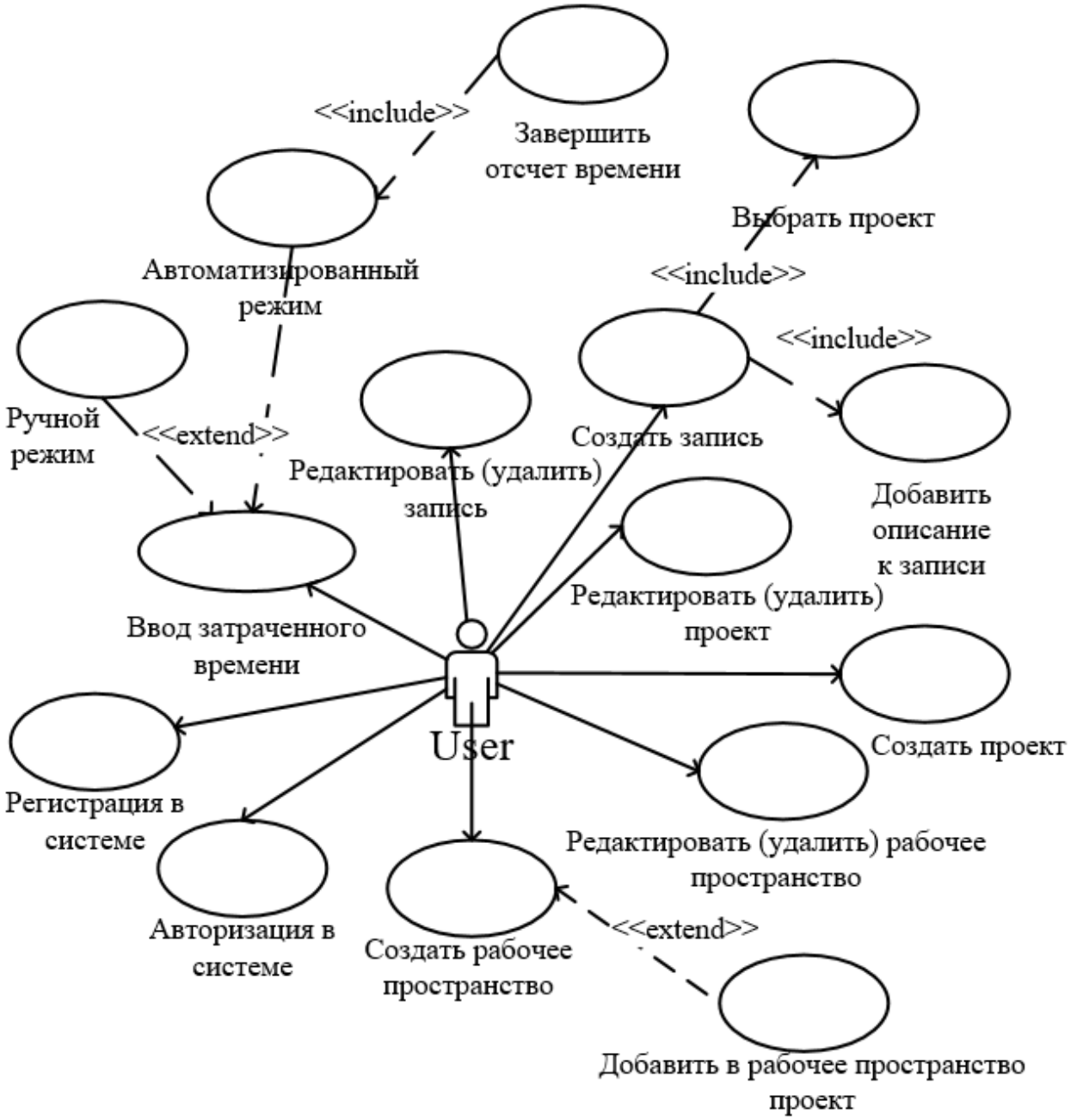

Рисунок 1.1 – Диаграмма вариантов использования

Данная диаграмма вариантов использования (Рисунок 1.1) отображает доступную функциональность серверного приложения. Среди которых есть как простые операции создание\обновление\удаление существующих сущностей, так и сложные: авторизация, отсчет времени В настоящий момент вся функциональность, описанная на диаграмме, кроме регистрации в системе,

реализована. Подробное описание вариантов использования приведены ниже в нотации UML[1] (Таблица 1.2 - Таблица 1.6).

Таблица 1.1 - Варианты использования

Регистрация в системе	-
Авторизация в системе	Авторизация в системе использую учетные данные google аккаунта.
Создание рабочего пространства	Создание сущности типа Workspace
Редактирование (удаление) рабочего пространства	Обновление полей существующего объекта или удаление оного.
Создание проекта	Создание сущности типа Project
Редактирование (удаление) проекта	Обновление полей существующего объекта или удаление оного.
Создание задания	Создание сущности типа Task
Редактирование (удаление) записи	Обновление полей существующего объекта или удаление оного.
Ввод рабочего времени	Время вводится либо в ручном режиме, либо автоматически при создании задания.

Таблица 1.2 - Прецедент: Регистрация в системе

ID: 1
Краткое описание: Пользователь проходит регистрацию, вводит свои данные, чтобы в дальнейшем он мог принимать участие в проектах.
Актёры: Регистрацию должны пройти все актеры.
Предусловия: Пользователь не зарегистрирован.

Основной поток: 1. Прецедент начинается, когда пользователь переходит на форму регистрации. 2. Пользователь вводит личные данные. 3. Введенные данные проверяются на корректность. 4. Если введенные данные корректны, то регистрация завершается, иначе пользователю указываются ошибки, которые необходимо исправить.
Постусловия: Пользователь может быть авторизован.
Альтернативные потоки: Нет.

Таблица 1.3 - Прецедент: Авторизация в системе

ID: 2
Краткое описание: Пользователь вводит свои данные, чтобы войти в систему и использовать функционал приложения.
Актёры: Авторизацию должны пройти все актеры.
Предусловия: Пользователь не авторизован системой.
Основной поток: 1. Прецедент начинается, когда пользователь запускает приложение. 2. Пользователь вводит данные для входа в систему. 3. Введенные данные проверяются сервером. 4. Если введенные данные корректны, то пользователь авторизован и получает доступ к функциям приложения, иначе пользователю указывается ошибка.
Постусловия:

Пользователь авторизован.
Альтернативный поток: 1. Прецедент начинается, когда пользователь запускает приложение. 2. Пользователь выбирает авторизацию через google аккаунт. 3. Введенные данные проверяются сервером. 4. Если введенные данные корректны, то пользователь авторизован и получает доступ к функциям приложения, иначе пользователю указывается ошибка.

Таблица 1.4 - Прецедент : Создание рабочего пространства

ID: 3
Краткое описание: Система позволяет авторизированному пользователю, создать рабочее пространство в котором он может объединить свои проекты.
Главные актеры: Авторизованный пользователь (User).
Второстепенные актеры: Нет.
Предусловие: 1. Пользователь вошел в систему.
Основной поток: 1. Пользователь выбрал опцию «Создать рабочее пространство». 2. Система выводит форму (окно), в которой необходимо указать название рабочего пространства. 3. Пользователь вводит текст. 4. Пользователь нажимает кнопку применить. 5. Система создает рабочее пространство.
Постусловие: В системе создано рабочее пространство с указанным именем.
Альтернативный поток: Нет.

Таблица 1.5 - Прецедент: Создание проекта

ID: 4
Краткое описание: Пользователь имеет возможность создать проект и добавить его в существующее рабочее пространство
Главные актеры: Пользователь.
Второстепенные актеры: Нет.
Предусловие: 1. Пользователь вошел в систему. 2. Создано хотя бы одно рабочее пространство.
Основной поток: 6. Прецедент начинается, когда пользователь выбирает пункт «Создание проекта» 7. Пользователь вводит название и описание проекта 8. Пользователь выбирает рабочее пространство и выбирает опцию «Сохранить» 9. Система создает проект с указанным именем и добавляет его в выбранное рабочее пространство
Постусловие: Создан новый проект Пользователя и добавлен в выбранное рабочее пространство
Альтернативный поток: Нет.

Таблица 1.6 - Прецедент : Создание записи

ID: 5
Краткое описание: Система позволяет авторизированному пользователю создать запись (задачу) для ведения учета затраченного на эту запись (задачу) времени..
Главные актеры: Авторизованный пользователь (User).
Второстепенные актеры: Нет.
Предусловие: 1. Пользователь вошел в систему.
Основной поток: 1. Пользователь выбрал опцию «Создать запись». 2. Система выводит форму (окно), в которой необходимо указать описание и выбрать проект. 3. Пользователь вводит текст описания. 4. Пользователь выбирает проект. 5. Пользователь нажимает кнопку сохранить. 6. Система создает запись с указанным описанием в выбранном проекте.
Постусловие: В системе создана запись с указанным описанием в выбранном пользователем проекте.
Альтернативный поток: Нет

Таблица 1.7 - Прецедент : Ввод затраченного времени

ID: 6
Краткое описание: Система позволяет авторизированному пользователю вести учет затраченного времени.
Главные актеры: Авторизованный пользователь (User).
Второстепенные актеры: Время.
Предусловие: 1. Пользователь вошел в систему. 2. Пользователь выбрал запись (задачу).
Основной поток: 1. Пользователь выбрал опцию «Ввод затраченного времени». 2. Система выводит форму (окно), в которой необходимо указать способ ввода затраченного времени: ручной режим или автоматизированный режим. 3. Пользователь выбирает ручной режим. 4. Пользователь задает начало и конец отсчета. 5. Система добавляет к выбранной записи значение затраченного времени.
Постусловие: В выбранную запись (задачу) добавлено значение затраченного времени.

Альтернативный поток:

1. Пользователь выбрал опцию «Ввод затраченного времени».
2. Система выводит форму (окно), в которой необходимо указать способ ввода затраченного времени: ручной режим или автоматизированный режим.
3. Пользователь выбирает автоматизированный режим.
4. Пользователь нажимает на кнопку «Начать отсчет».
5. Через некоторый промежуток времени пользователь нажимает на кнопку «Закончить отсчет».
6. Система добавляет к выбранной записи значение затраченного времени.

2 Анализ

2.1 Классы анализа

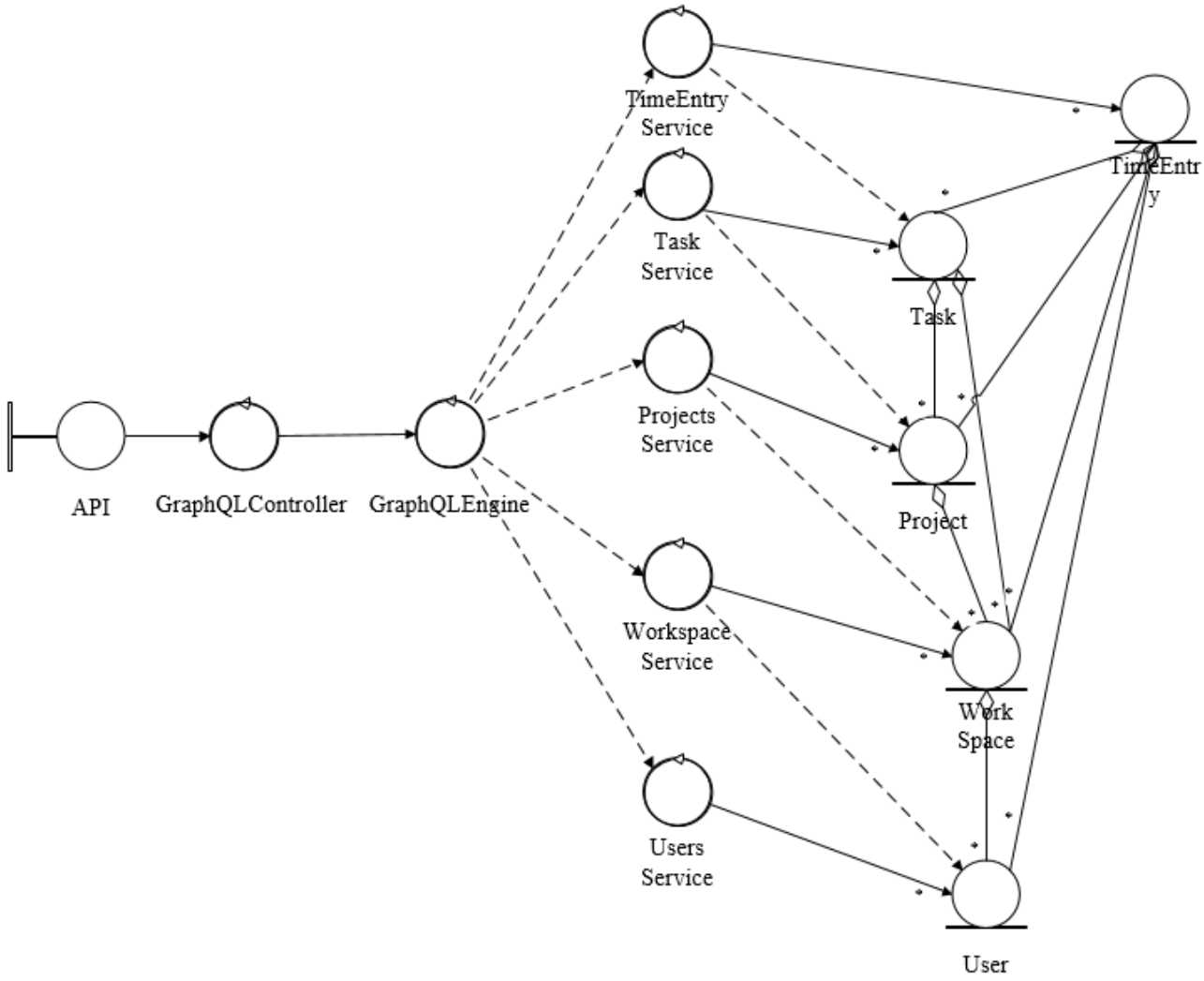

Рисунок 2.1 – Диаграмма классов анализа

Диаграмма классов анализа содержит набор объектов модели приложения, которыми управляют соответствующие сервисы.

Все запросы, которые могут быть обработаны серверным приложением отправляются на GraphQLController. В нем из тела http запроса извлекаются отправленные данные и формируется обращение к механизмам GraphQL технологии [2].

GraphQLEngine (GraphQL) – это библиотека позволяющая использовать специальный синтаксис в http запросах, который описывает как запрашивать данные, и, в основном, используется клиентом для загрузки данных с сервера [3].

Аналогично чтению данных, GraphQL может использоваться и для записи новой информации на серверное приложение. Функции, отвечающие за это, называются: мутации. Все типы, мутации и функции чтения данных указываются в схемах.

GraphQLEngine скрывает в себе реализации парсинга схем, согласования функций описанных в схемах с реальными обработчиками серверного приложения и сами обработчики.

Workspace – сущность имеет смысл рабочего места и объединяет в себе связанные по смыслу объекты типа Project. Имеет имя.

Project – сущность объединяющая в себе связанные по смыслу задачи. Содержит только имя и ссылки на сущности типа Task.

Task – сущность отражающая некоторую деятельность. Может содержать ряд временных промежутков, описание и имя.

TimeEntry – сущность по работе со временем. Содержит время старта, окончания и длительность.

Workspace Service – отвечает за все функции необходимые при работе с типом Workspace. А также управляет временем жизни объектов этого типа.

Project Service – отвечает за все функции необходимые при работе с типом Project. А также управляет временем жизни объектов этого типа.

Task Service – отвечает за все функции необходимые при работе с типом Task. А также управляет временем жизни объектов этого типа.

TimeEntry Service – отвечает за все функции необходимые при работе с типом TimeEntry. А также управляет временем жизни объектов этого типа.

User Service – отвечает за все функции необходимые при работе с типом User. А также управляет временем жизни объектов этого типа.

2.2 Диаграммы деятельности

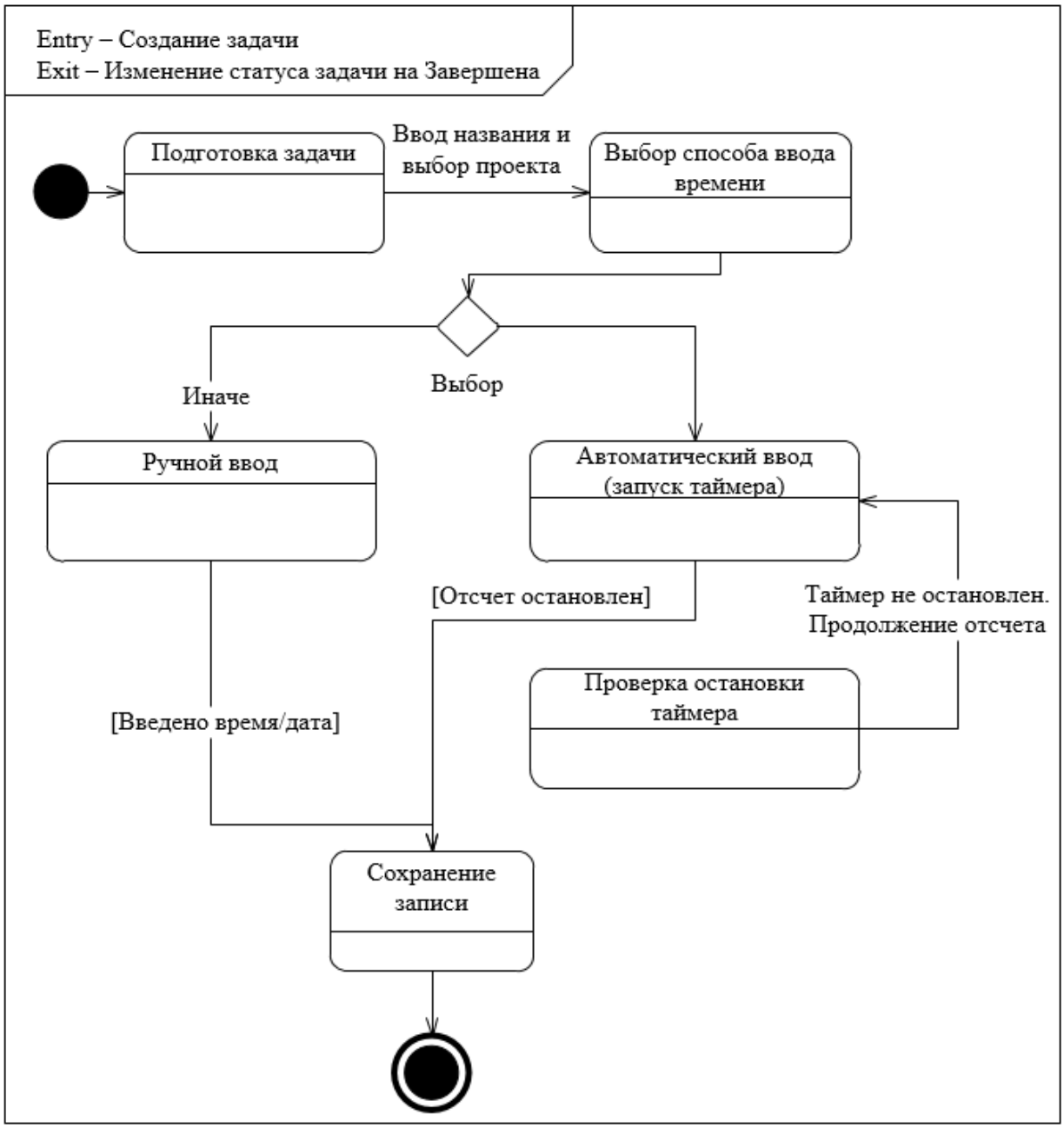

Рисунок 2.2 – Диаграмма деятельности для «Создания задачи»

3 Проектирование

3.1 Проектные классы

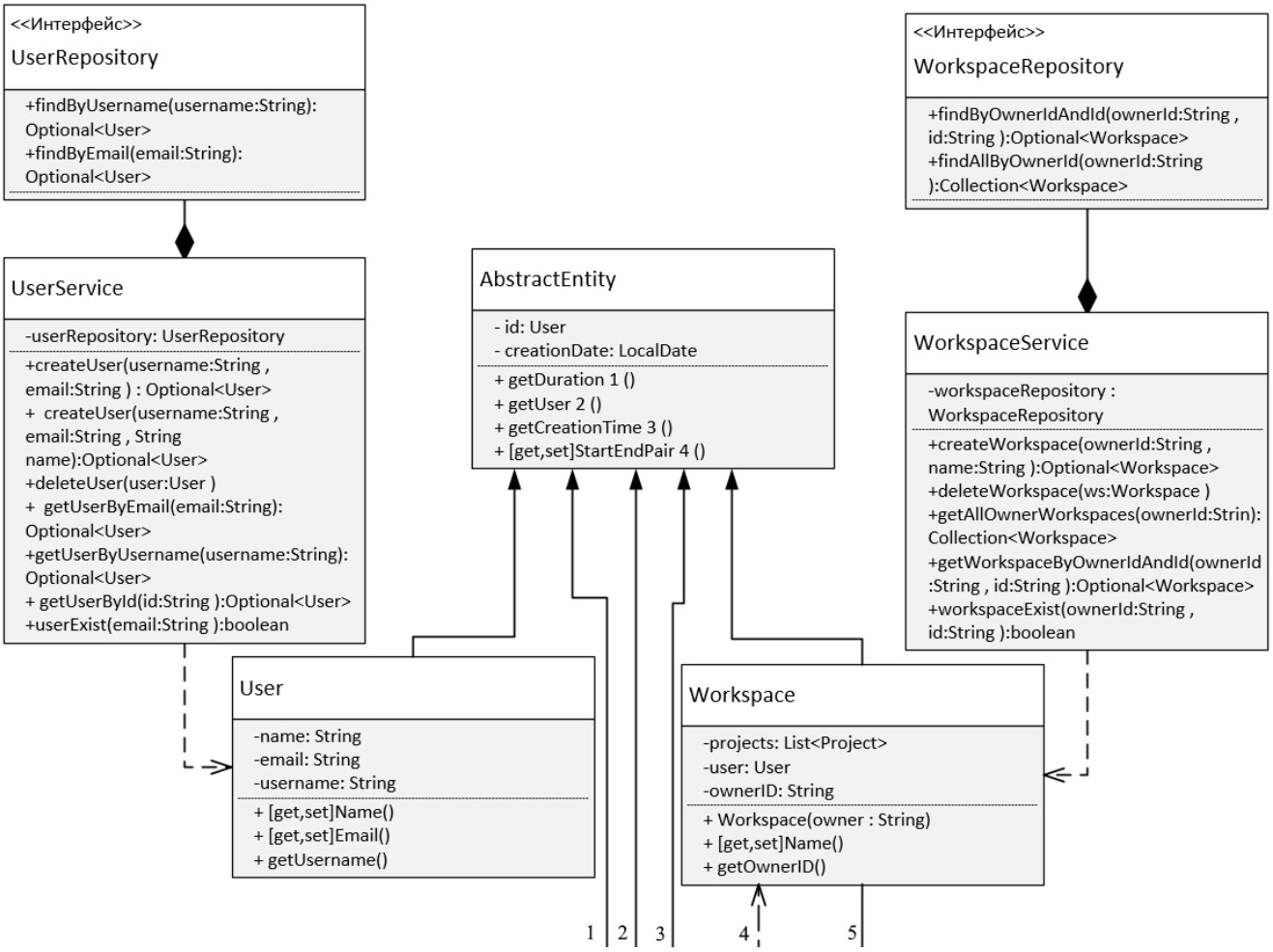

Рисунок 3.1 - Диаграмма проектных классов. Часть 1

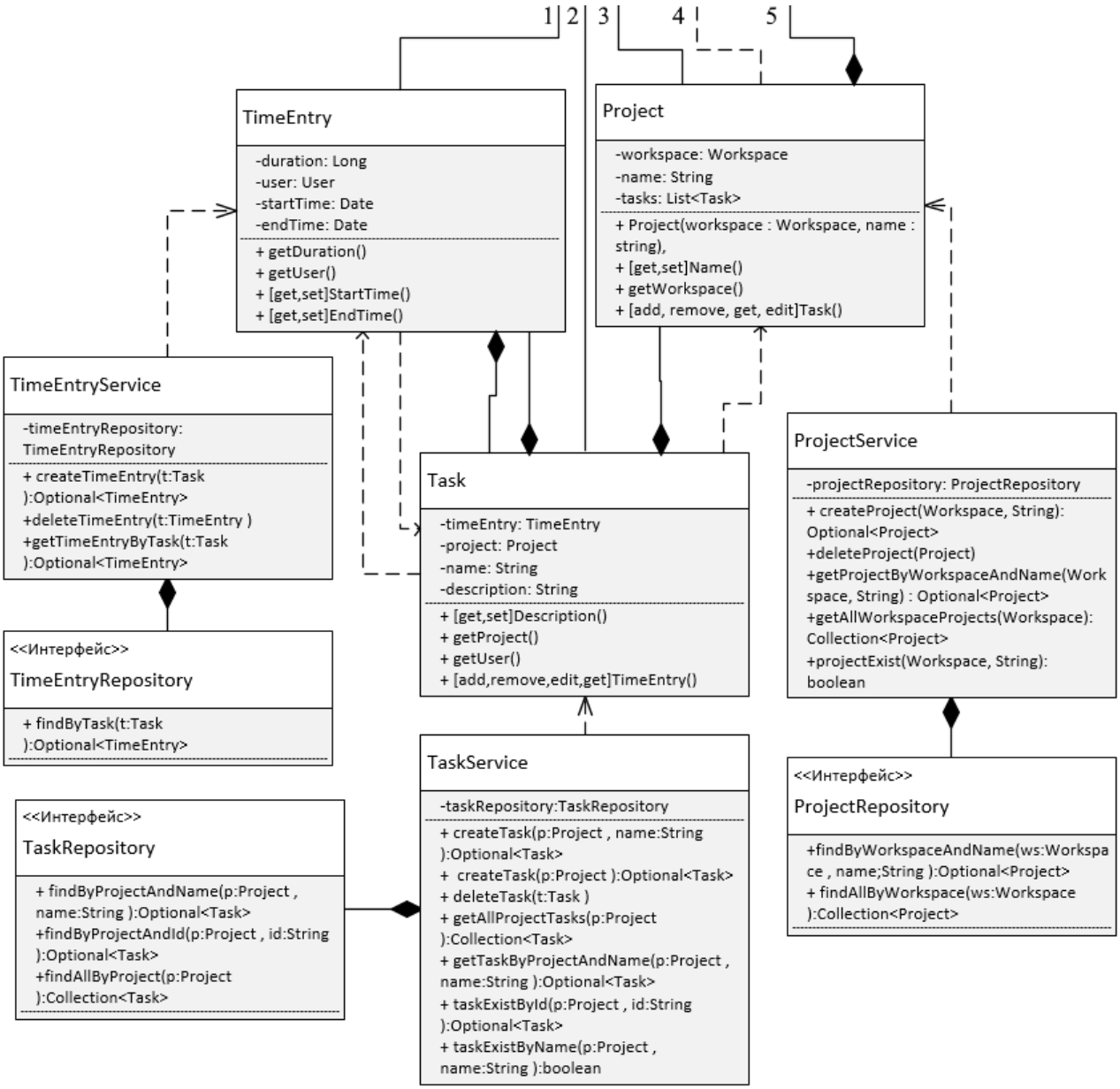

Рисунок 3.2 - Диаграмма проектных классов. Часть 2

3.2 Диаграммы последовательностей для операций проектных классов

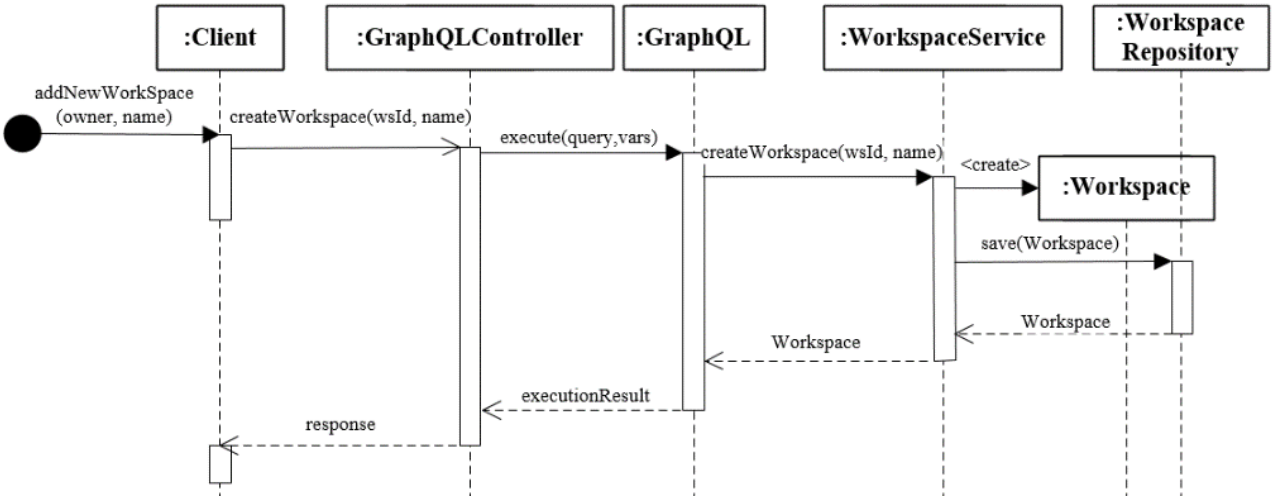

Рисунок 3.3 – Диаграмма последовательности создания рабочего пространства

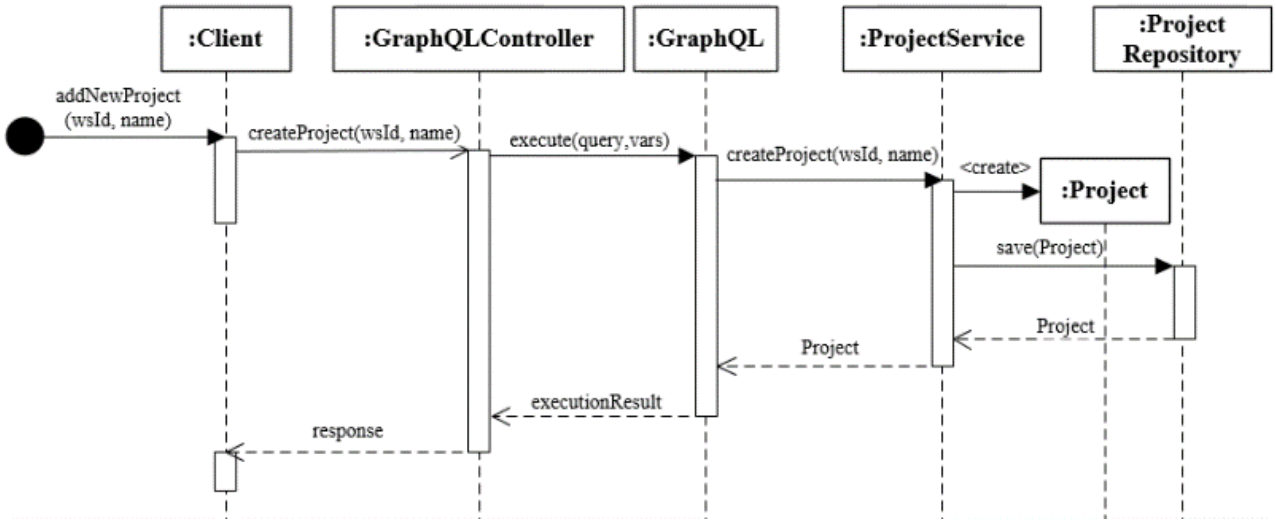

Рисунок 3.4 – Диаграмма последовательности создания проекта

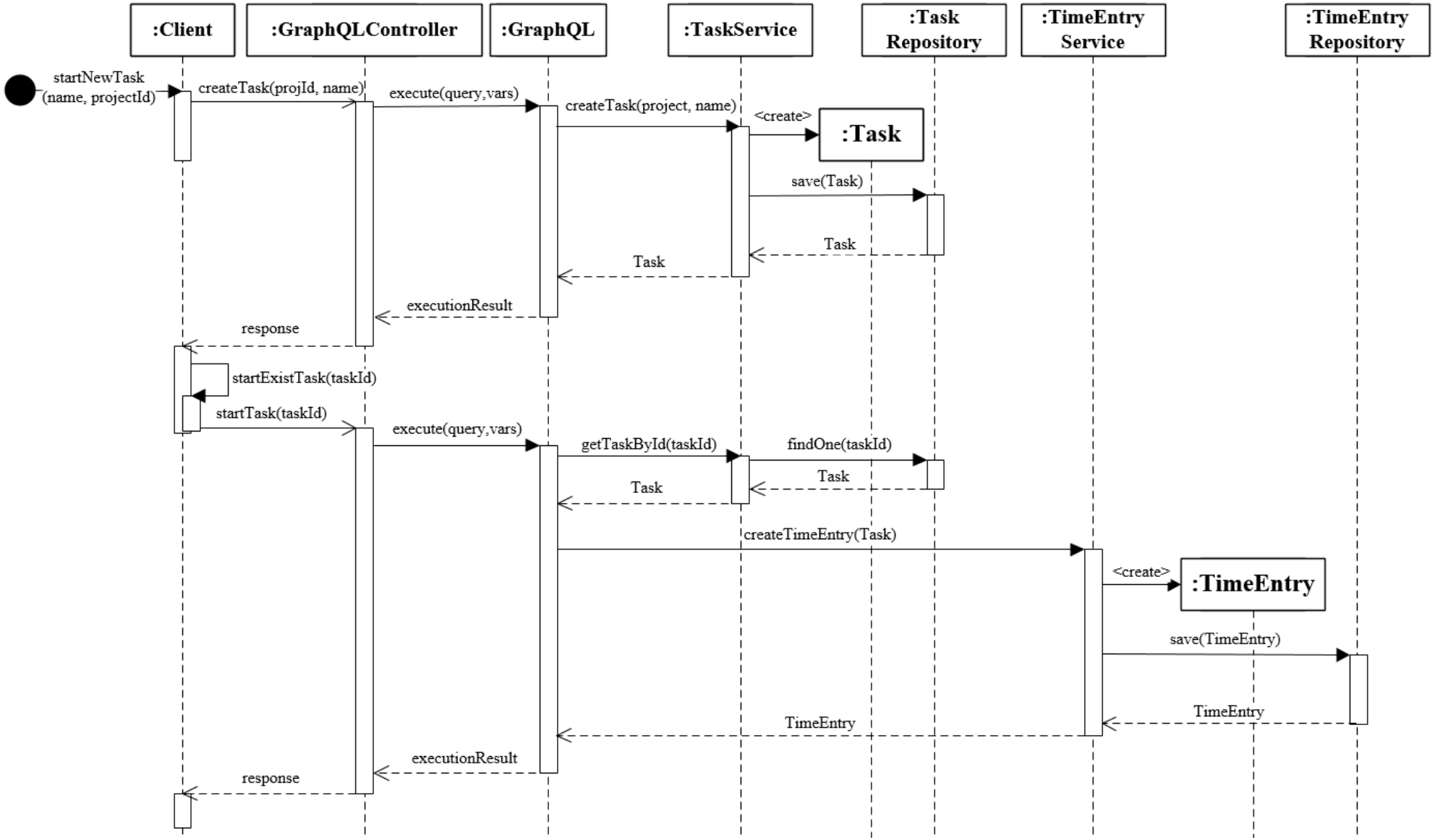

Рисунок 3.5 - Диаграмма последовательности создания и запуска задачи

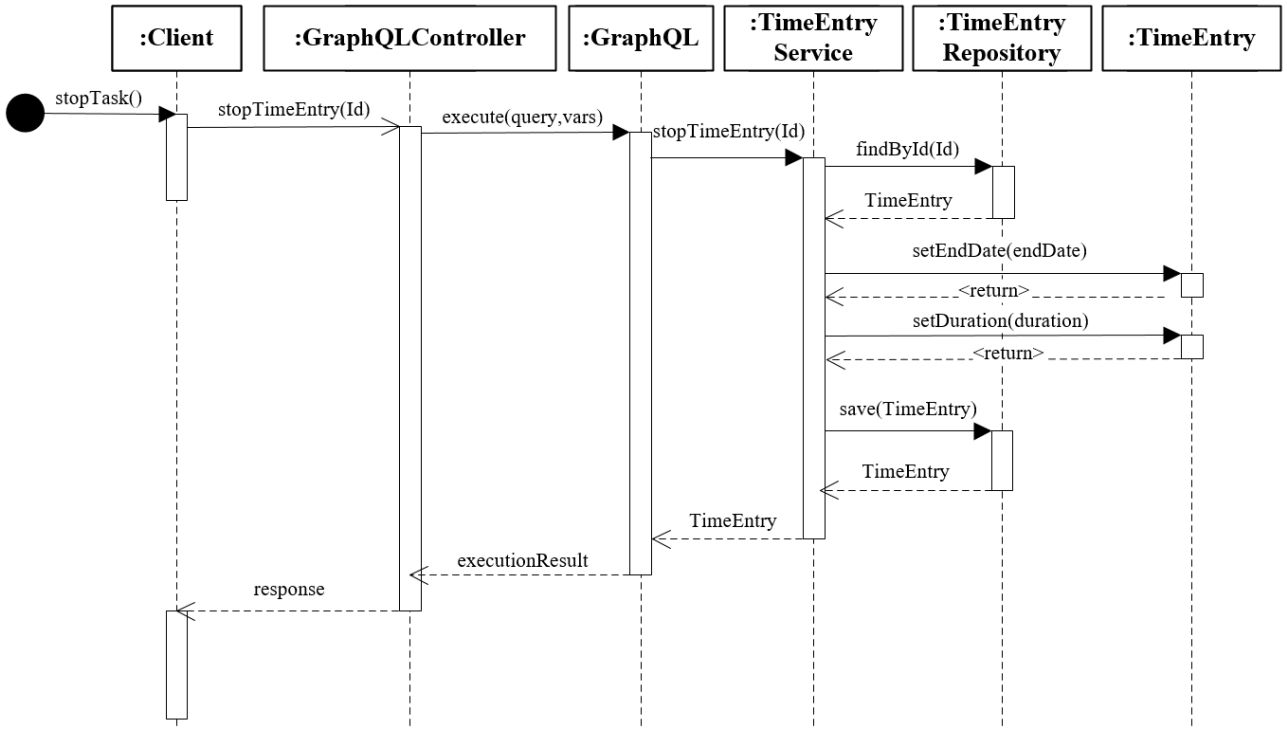

Рисунок 3.6 – Диаграмма последовательности завершения задачи

4 Реализация

4.1 Тестирование системы

Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Другими словами, можно сказать, что тестирование - это одна из техник контроля качества, включающая в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов.

Существует два основных вида тестирования: модульное и интеграционное. Модульное тестирование - это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу "белого ящика", то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода [4].

Интеграционное тестирование - это тестирование части системы, состоящей из двух и более модулей. Основная задача интеграционного тестирования - поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями.

Основная идея модульного тестирования состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. А интеграционное тестирование представляет собой одну из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования.

Для Android приложения были написаны тесты, которые проверяется корректность взаимодействия с локальной базой данных Realm. В ОС Android используется стандартная база данных SQLite, взаимодействие с которой выполняется через Java интерфейс SQLiteDatabase и SQLiteOpenHelper. Но имплементировать стандартные механизмы по управлению БД в Android непросто и неудобно. Для упрощения процесса используются различные библиотеки и ORM “надстройки”, в том числе и Realm. Realm - кроссплатформенная, мобильная база данных, использующая собственный движок, обеспечивающий высокую скорость работы и простоту в применении. Практика показывает, что в большинстве случаев Realm значительно превосходит в скорости не только SQLite, но и другие альтернативные ORM для Android, такие как ORMLite и Greendao. Также стоит отметить простоту интеграцию и поддержку Realm в Android [5].

Написанные тесты проверяют корректность выполнения основных операций: добавление, обновление и удаление, с сущностью «Задача». Написаны модульные тесты, тестирующие такие методы как “addTask”, “getTask”, “getTasksOfProject”, “startTask”, “finishTask”, “removeTask”.

Тест “addTask” создает сущность, вносит ее в локальную БД, запоминая ее идентификатор. Затем тест извлекает эту сущность по ее идентификатору и сравнивает ожидаемый и полученный объекты.

Тест “ getTasksOfProject” предварительно создает сущность “Проект” и несколько сущностей “Задач”. Затем из БД извлекаются все задачи указанного проекта, и результат проверяется на корректность.

Тесты “startTask” и “finishTask” проверяет изменение состояние задачи, добавление и обновление сущность “StatisticsTask” (“TimeEntry”).

Тест “removeTask” проверяет удаление задачи и связанные с ними сущности “TimeEntry”.

Тестирование БД Realm относится к инструментальным unit-тестам, а это означает, что тестирование только с помощью JVM невозможно, нужен эмулятор

или реальное устройство Android. Инструментальные unit-тесты находятся в папке “androidTest” и запуск теста можно запустить с помощью команды “./gradlew connectedCheck“ или интерфейса среды разработки, предварительно настроив эмулятор или реальное устройство [6].

Результат тестирование представлен на рисунке 1. Тесты завершились успешно, что означает, что ожидаемые результаты совпали с действительными.

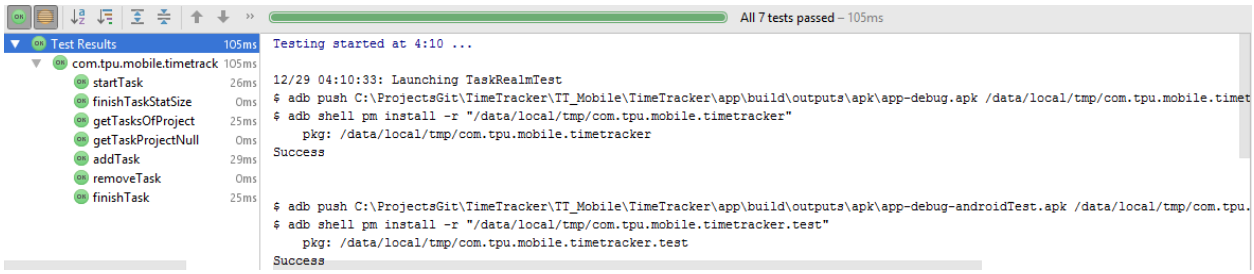

Рисунок 4.1 -Результаты тестирования

предварительно создает сущность “Проект” и несколько сущностей “Задач”. Затем из БД извлекаются все задачи указанного проекта, и результат проверяется на корректность.

4.2 Непрерывная интеграция

Непрерывная интеграция – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. Слово continuous в термине Continuous Integration означает «непрерывный / непрекращающийся». Это означает, что в идеале сборка проекта должна идти буквально все время. Каждое изменение в системе контроля версий должно интегрироваться без пропусков или задержек. Непрерывная интеграция является одним из основных приемов экстремального программирования.

В качестве систем непрерывной интеграции выбраны веб-сервисы Travis CI, AppVeyor.

4.2.1 Travis CI

Travis CI – распределенный веб-сервис для сборки и тестирования программного обеспечения, использующий GitHub [7] в качестве хостинга исходного кода. Данный сервис бесплатен для публичных проектов. В курсовом проекте Travis CI используется для непрерывной интеграции проекта серверного ПО, а также проекта клиентского ПО, разработанного под Android платформу [8].

Первым шагом является настройки системы является вход в учетную запись Travis CI через аккаунт на GitHub [9]. Для этого необходимо на странице https://travis-ci.org нажать на кнопку “Sign In With Github” (Рисунок 4.2).

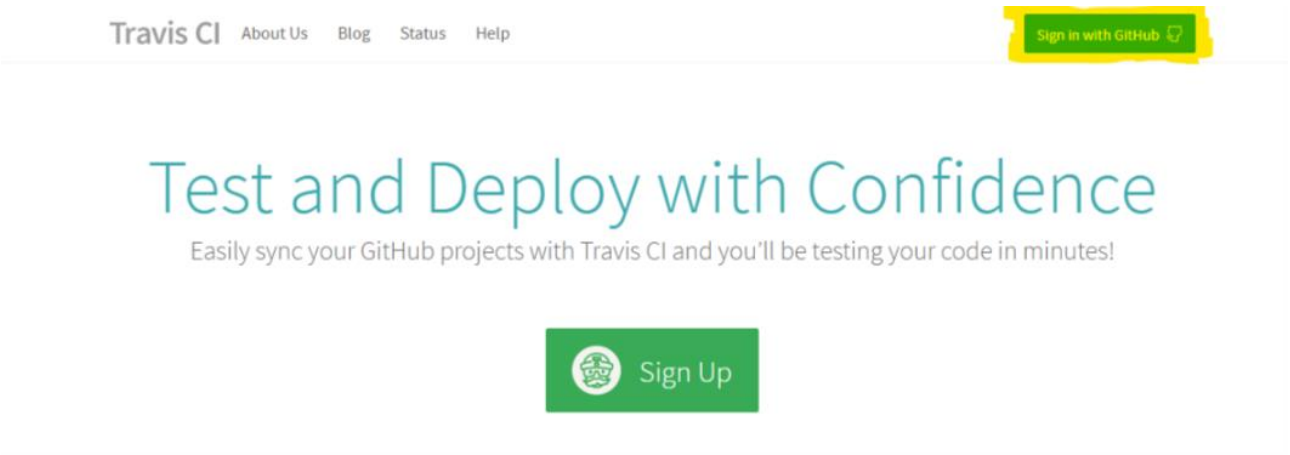

Рисунок 4.2- Главная страница Travis CI.

После входа на сервис необходимо активировать синхронизацию репозитория проекта на GitHub с сервисом. Для это необходимо зайти в настройки профиля и привести необходимые переключатели во включенное положение как показано на Рисунок 4.3.

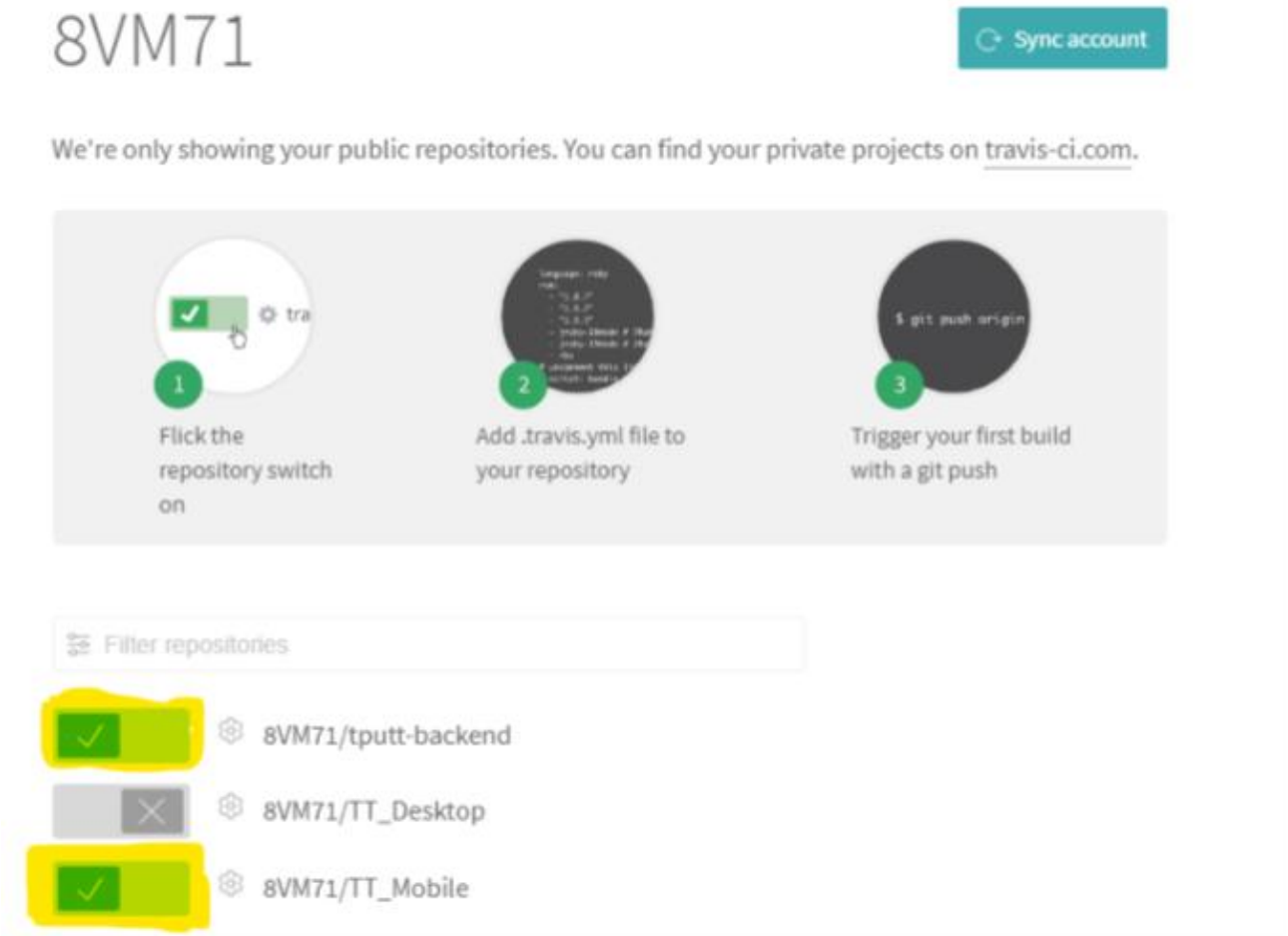

Рисунок 4.3 – Настройка доступа Trais CI к репозиторию

Следующим шагом является добавление в корневую директорию проекта файла .travis.yml. В этом файле содержится скрипт, в котором определены настройки окружения сборки проекта, скрипты, выполняемые перед сборки, скрипты сборки и скрипты, выполняемые по окончанию сборки. Всю подробную информацию о настройке данного файла можно найти на сайте Travis CI.

Содержание файла .travis.yml для проекта серверного ПО представлено ниже:

```
sudo: required

services:
  - docker

language: java

jdk:
  - oraclejdk8

before_install:
  - chmod +x gradlew

install: true

cache:
  directories:
    - $HOME/.gradle/caches/
    - $HOME/.gradle/wrapper/

script:
  - ./gradlew clean
  - ./gradlew test --rerun-tasks
  - ./gradlew assemble

after_success:
  - sh .travis/deploy_dockerhub.sh
  - if [ -e ./gradlew ]; then ./gradlew jacocoTestReport;else gradle
jacocoTestReport;fi
  - bash <(curl -s https://codecov.io/bash)
```

В данном файле указывается, что проект написан на языке Java. На основе этой информации Travis автоматически подготовит необходимое окружение для сборки, характерное для сборки большинства проектов на этом языке. В секции script указывается скрипт сборки приложения. Во время сборки приложения автоматически соберутся и запустятся unit-тесты и если некоторые из них не пройдут, то сборка будет считаться неудачной. После удачной сборки проекта выполняется скрипт сборки Docker образа, содержащем ПО и отправка этого образа в Docker репозиторий. После выполняется отправка отчета о

покрытии кода тестами на сервис CodeCov, где анализируется процент покрытия кода тестами.

Ниже представлен рисунок успешно выполненной сборки серверного ПО (Рисунок 4.7).

Содержимое и описание файла .travis.yml для приложения Android представлено далее:

В файле ".travis.yml" указывается язык разработки “Android” и версия Java. В глобальных переменных присутствует переменная “ADB_INSTALL_TIMEOUT”, отвечающая за время ожидания отклика эмулятора, и “secure”, который содержит токен разработчика. Токен обязательно должен быть зашифрованным. Для возможности развертки собранного приложения на гитхабе необходимо создать токен с полным контролем над проектом (Рисунок 4.4). Зашифровать токен можно с помощью команды «travis encrypt GH_TOKEN=token_here”.

```
sudo: required
language: android
jdk: oraclejdk8
env:
  global:
  - ADB_INSTALL_TIMEOUT=10
  - secure: "$GH_TOKEN”
```

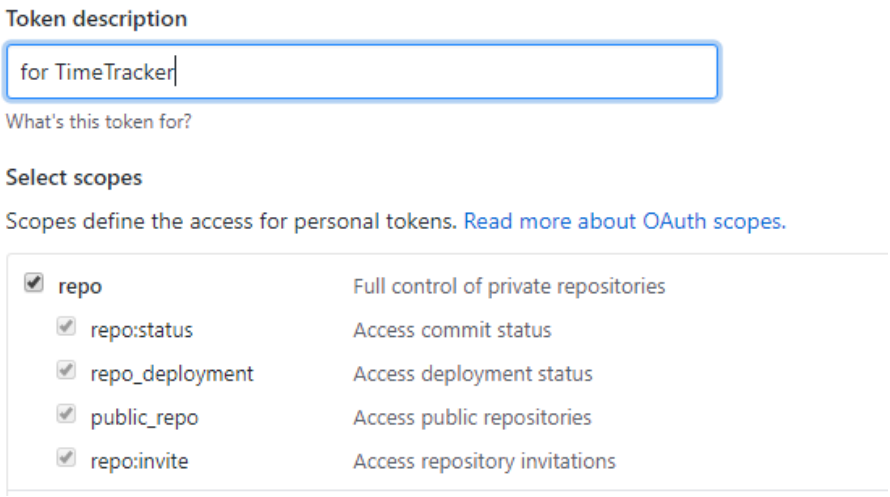

Рисунок 4.4 - Создание токена

Следующим этапом в написании файла конфигурации является указание необходимых используемых компонентов: версия SDK, версия эмулятора,

различные библиотеки и лицензии. Можно указать несколько версий эмуляторов, это позволяет протестировать приложение на разных версиях Android, что особенно актуально для инструментальных тестов. В нашем случаи они и тестируются. Использование несколько эмуляторов дает гарантию, что приложение будет протестировано, но заметно увеличивает время сборки.

```
android:
  components:
  - tools
  - platform-tools
  - tools
  - build-tools-25.0.3
  - android-25
  - android-22
  - extra-android-m2repository
  - extra-android-support
  - extra-google-m2repository
  - extra-google-android-support
  - extra-google-google_play_services
  - sys-img-armeabi-v7a-android-22
  licenses:
  - android-sdk-preview-license-.+
  - android-sdk-license-.+
  - google-gdk-license-.+
```

Далее представлены команды запуска эмулятора и установка разрешения выполнения файлов сборки приложения и скрипта для развертывания.

```
install:
  #Starts an emulator for instrumental tests
  - echo no | android create avd --force -n test -t android-22 --
abi armeabi-v7a -c 32M
  - emulator -avd test -no-audio -no-window &
  - android-wait-for-emulator
  - adb shell input keyevent 82 &
  - chmod +x ./TimeTracker/gradlew
  - chmod +x ./autobuild/tags.sh
```

Предварительно проект очищается, проходят инструментальные тесты, а затем проект собирается в файл “.apk”.

```
script:
  #Deletes and clean the build directory
  - ./gradlew clean
  - ./gradlew connectedCheck
  - ./gradlew assembleDebug
```

Перед релизом происходит выполнение скрипта “tags.sh”, который создает “tag” и выполняет команду “push” в ветку проекта.

```
before_deploy:
  - ./autobuild/tags.sh
```

Последний участок кода предназначен для размещения созданного установочного файла в “release” репозитория. Установлена возможность перезаписи файла и распространяется на все ветки репозитория.

```
deploy:
  provider: releases
  api_key:
    secure: “..”
  file: TimeTracker/app/build/outputs/apk/TimeTracker.apk
  overwrite: true
  skip_cleanup: true
  on:
    all_branches: true
```

Рисунок 4.5 иллюстрирует результат работы данного конфигурационного файла. Первые попытки сборки оказались неудачными, но изучение логов (Рисунок 4.6) указало на ошибку, которая была связана с конфигурационным файлом, предназначенным для авторизации google – пользователей. После обновления данного файла, сборка прошла успешно.

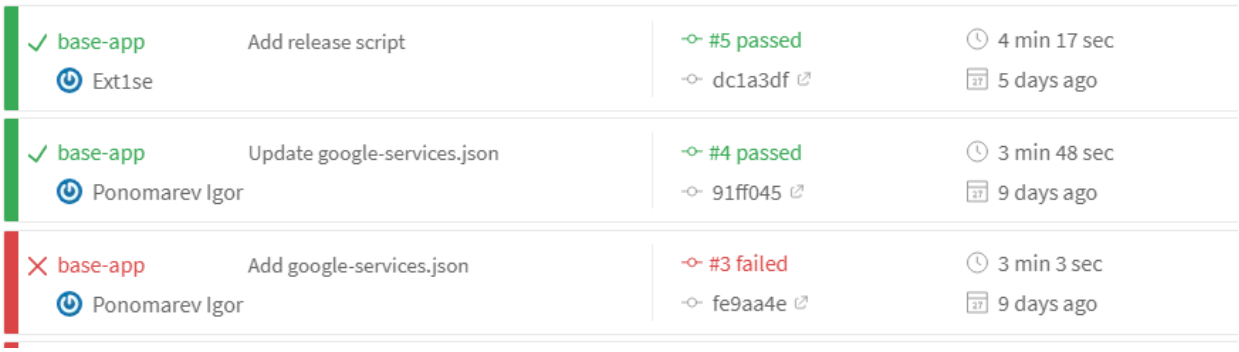

Рисунок 4.5. История сборок проекта

```
FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app:processDebugGoogleServices'.
> File google-services.json is missing. The Google Services Plugin cannot function without it.
   Searched Location:
  /home/travis/build/8VM71/TT_Mobile/TimeTracker/app/src/debug/google-services.json
```

Рисунок 4.6. Лог ошибки

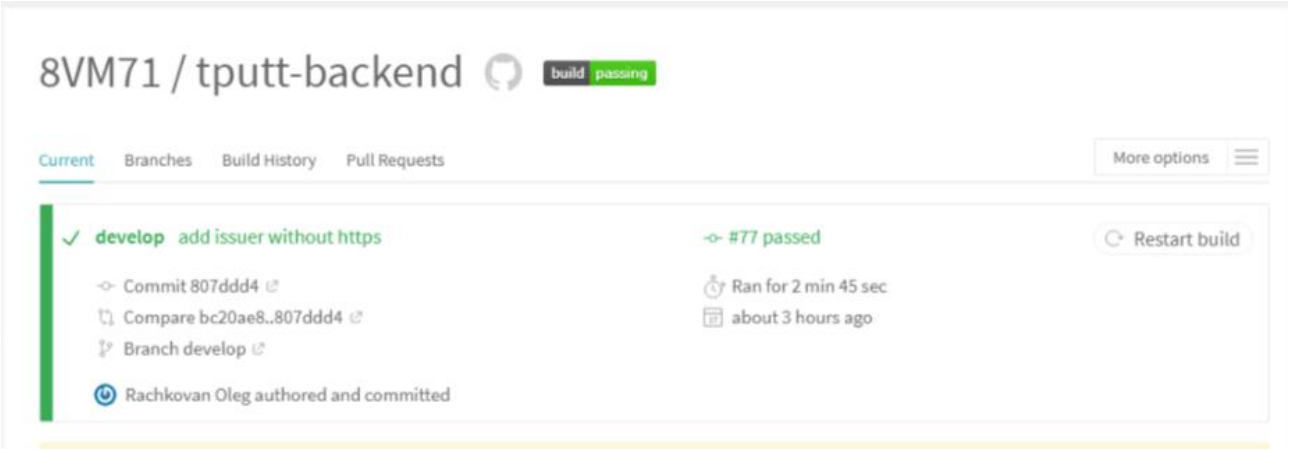

Рисунок 4.7 – Сборка успешно выполнена

Результат неудачной сборки (Рисунок 4.8)

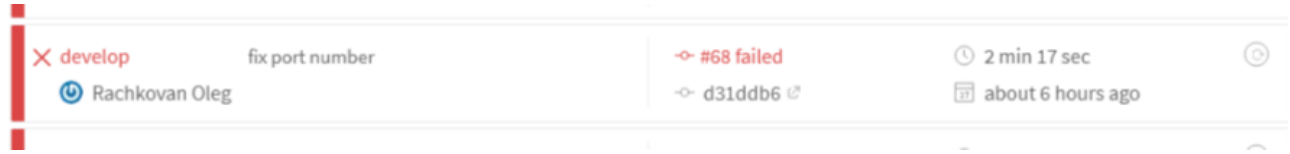

Рисунок 4.8 – Сборка окончена со статусом “failed”

4.2.2 AppVeyor

Для настольного приложения в качестве системы непрерывной интеграции была выбрана система AppVeyor [10]. Данный сервис специализируется на работе с Windows приложениями. Также он бесплатен для открытых проектов GitHub.

Для настройки сначала необходимо войти в учетную запись AppVeyor через аккаунт GitHub (Рисунок 4.9-Рисунок 4.10).

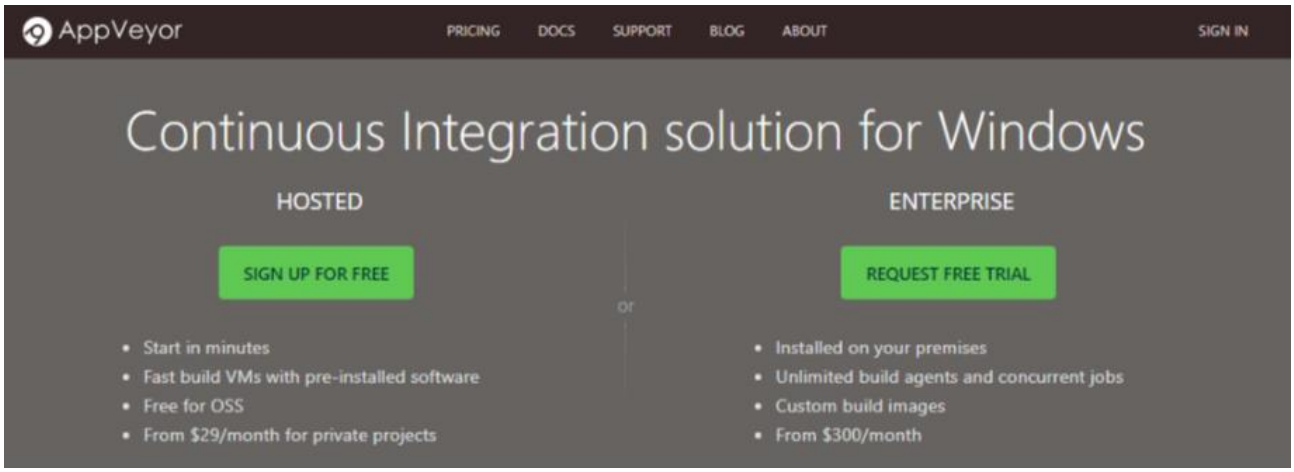

Рисунок 4.9 – Главная страница AppVeyor

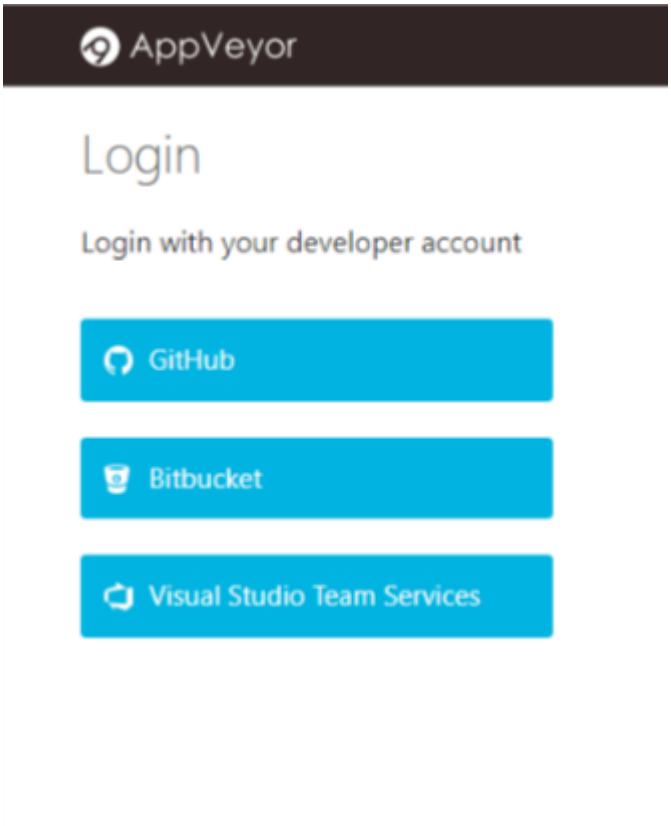

Рисунок 4.10 – Форма входа

После на вкладке Projects выбрать пункт New Project и выбрать необходимый репозиторий проекта (Рисунок 4.11).

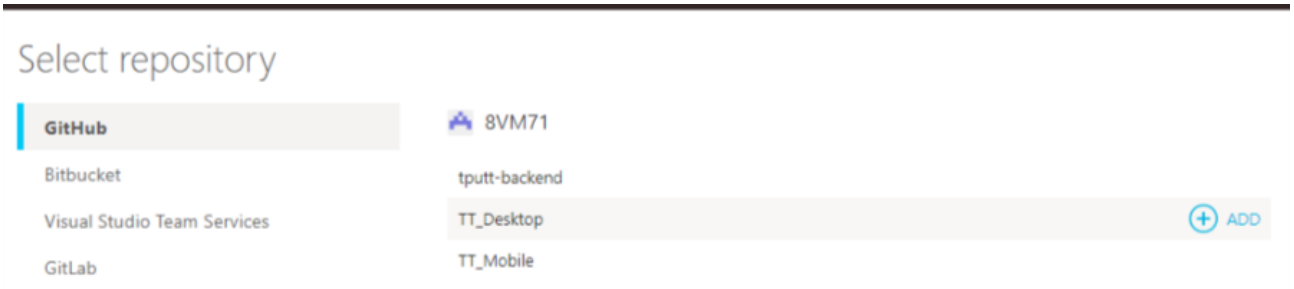

Рисунок 4.11 – Добавление проекта

После добавления проекта в сервис AppVeyor нужно в корневую директорию проекта добавить файл appveyor.yml, где описать конфигурацию процесса сборки.

Код файла для проекта настольного клиента приведен ниже:

```
version: 0.0.0.{build}
image: Visual Studio 2015

branches:
  only:
  - develop

skip_tags: true

environment:
  QTDIR: C:\Qt\5.9.2\msvc2015
  VSPATH: C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC
  TOOLSDIR: C:\Qt\Tools\QtCreator
  INNOSETUPPATH: C:\Program Files (x86)\Inno Setup 5\
  INSTALLER_NAME:
"%APPVEYOR_PROJECT_NAME%%APPVEYOR_BUILD_VERSION%"
  PR_NAME: "%APPVEYOR_PROJECT_NAME%"
  BUILD_V: "%APPVEYOR_BUILD_VERSION%"
  BUILD_F: "%APPVEYOR_BUILD_FOLDER%"

before_build:
  -                                                                    set
PATH=%PATH%;%QTDIR%\bin;%VSPATH%\bin;%TOOLSDIR%\bin;%INNOSETUPPATH%
  - call "%VSPATH%\vcvarsall.bat"
  - mkdir build

build_script:
  - cd build
  - qmake ../tputt.pro -spec win32-msvc
  - jom

after_build:
  -  windeployqt  --release  %APPVEYOR_BUILD_FOLDER%\bin\release  --
qmldir %APPVEYOR_BUILD_FOLDER%
  -             xcopy             C:\OpenSSL-Win32\bin\libeay32.dll
%APPVEYOR_BUILD_FOLDER%\bin\release /y
  -             xcopy             C:\OpenSSL-Win32\bin\ssleay32.dll
%APPVEYOR_BUILD_FOLDER%\bin\release /y
  -       iscc.exe       /Qp       /DAPPVEYOR_PROJECT_NAME=%PR_NAME%
/DAPPVEYOR_BUILD_VERSION=%BUILD_V%      /DINSTALLER_NAME=%INSTALLER_NAME%
/DAPPVEYOR_BUILD_FOLDER=%BUILD_F%
"%APPVEYOR_BUILD_FOLDER%\.appveyor\installer.iss"
  -         7z         a         -y         ..\bin\installs\tttpu.zip
%APPVEYOR_BUILD_FOLDER%\bin\release\*

artifacts:
  - path: bin/installs/tttpu.zip
    name: archive
    type: zip
  - path: bin/installs/%INSTALLER_NAME%.exe
```

```
    name: installer

deploy:
  - provider: GitHub
    name: production
    release: $(appveyor_project_name)-v$(appveyor_build_version)
    description: 'Auto release on master'
    artifact: installer
    prerelease: true
    draft: true
    auth_token:
      secure: <code>
    on:
      branch: develop
```

В секции *environment* определяются переменные среды, необходимые для процесса сборки. В секции *build_script* определен скрипт сборки приложения. После успешной сборки приложения выполняется секция *after_build,* в которой определен скрипт копирования всех необходимых зависимостей, динамически подключаемых библиотек, сборки установочного пакета, а так zip-архива. Секция *artifacts* определяет артефакты сборки, их имена и пути до них (Рисунок 4.14). В секции *deploy* описана конфигурация развертывания установочного пакета на GitHub Releases.

Результат успешной сборки приложения представлена на Рисунок 4.12.

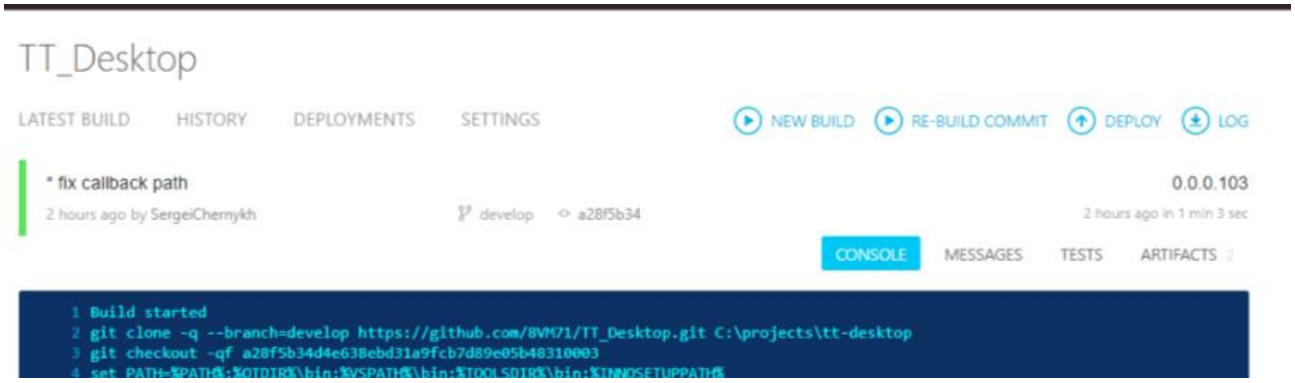

Рисунок 4.12 – Сборка успешно завершена

Результат неудачной сборки (Рисунок 4.13)

Рисунок 4.13 – Процесс сборки завершился неудачно

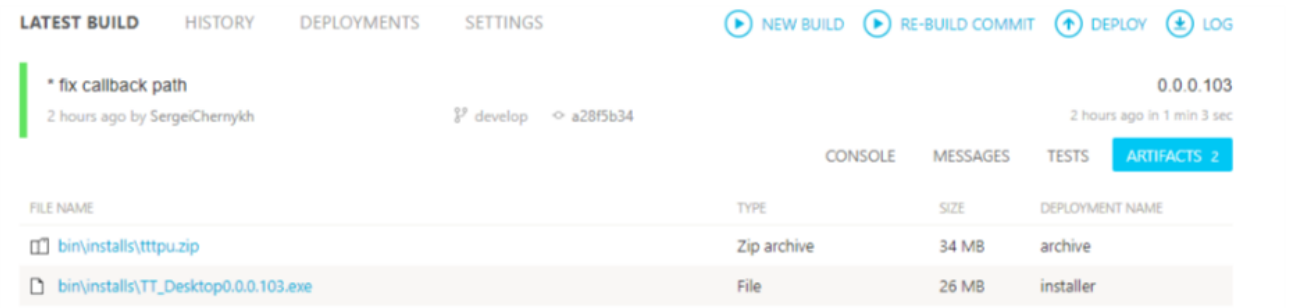

Рисунок 4.14 – Артефакты собранного приложения

4.3 Развертывание

Серверное программное обеспечение развертывается посредством облачного веб-сервиса Docker Cloud [11] на виртуальном сервере, предоставленном компанией Amazon.

Процесс настройки Docker Cloud начинается с создания ноды на вкладке Nodes [12]. При создании указываются ключи взаимодействия с сервисами Amazon [13]. Docker Cloud автоматически создает экземпляр виртуального веб-сервера. Созданный экземпляр (нода) отображается на вкладке Nodes (Рисунок 4.15).

Рисунок 4.15 – Созданный экземпляр виртуального веб-сервера

Далее на вкладке Services необходимо создать сервис (Рисунок 4.16), которой создает docker контейнер на основе полученного из docker регистра образ. Созданный контейнер (Рисунок 4.17) разворачивается на ноде. При каждом обновлении образа происходит переразвертывание контейнера.

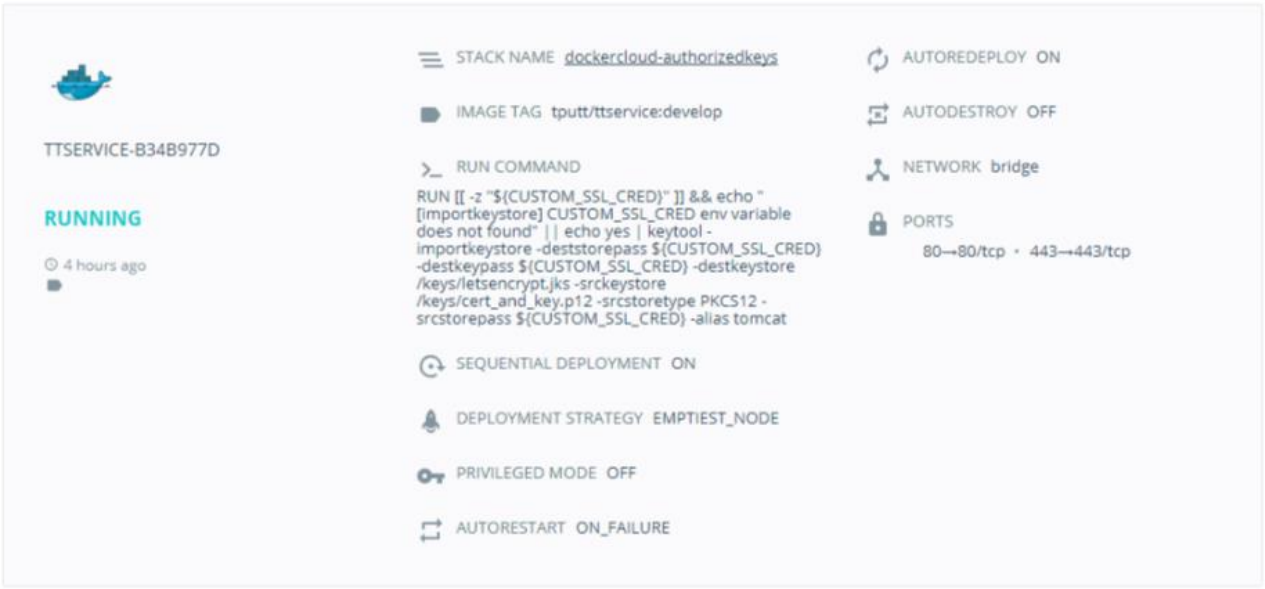

Рисунок 4.16 – Созданный сервис

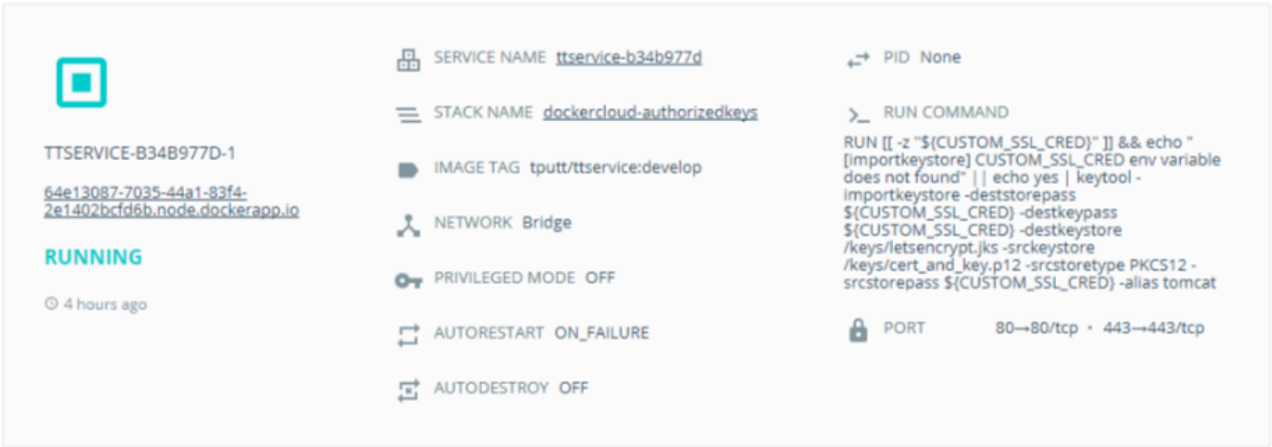

Рисунок 4.17 – Созданный контейнер

4.4 Взаимодействие с сервером

Для выполнения запросов к серверу используется GraphQL - синтаксис, который описывает, как запрашивать данные, и, в основном, используется клиентом для загрузки данных с сервера. GraphQL имеет три основные характеристики:

- Позволяет клиенту точно указать, какие данные ему нужны;
- Облегчает агрегацию данных из нескольких источников;
- Использует систему типов для описания данных.

Для Android-клиента используются специальная библиотека Apollo-Android, которая добавляет удобство в написании запросов. Но для того, чтобы полноценно использовать данную библиотеку необходимо предварительно настроить ее. Для этого нужно GraphQL схему и преобразовать в JSON формат. Получить GraphQL схему и преобразовать ее можно с помощью команды “apollo-codegen download-schema schema.graphql --output schema.json”. После того, как схема будет преобразована нужно создать файл “.graphql”, в который заносятся необходимые запросы.

Запросы GraphQL делятся на два типа: запросы – чтение и мутации, которые вносят изменения в серверную БД. Далее будут рассмотрены примеры подобных запросов.

Запрос на получение всех задач пользователя представлен ниже. Входной атрибут – идентификатор пользователя, используется для того, чтобы получить все задачи, которые находятся внутри проектов, а те – в рабочем пространстве. При успешном выполнении запроса возвратятся задачи: их название, дата создания и сущности TimeEntry, которые отвечают за временные показатели.

```
query getTasks($ownerId: String!)
{
  workspaces(ownerId: $ownerId){
    id, projects{
      id, tasks{
        id, name, description, crdate, timeEntries
        {
          id, crdate, duration, endDate, startDate
        }
```

```
        }
      }
    }
}

type Query {
    workspaces(ownerId: String!) : [Workspace]!
}
```

Пример мутации для создания рабочего пространства представлен ниже. Возвращаемым значением является идентификатор сущности, созданной на сервере, чтобы клиент внес его в локальную БД.

```
mutation createWorkspace($workspace: WorkspaceInput!) {
  createWorkspace(workspace: $workspace)
}

type Mutation {
    createWorkspace(workspace: WorkspaceInput!): String!
}
```

После того как все примеры запросов буду написано, библиотека Apollo соберет все указанные запросы, которые можно будет использовать как объекты и классы. Пример выполнения запроса на языке Java выглядит следующим образом.

```
final WorkspaceInput workspace = WorkspaceInput.builder()
  .ownerId(ownerID)
  .name(name)
  .description(description)
  .build();

Rx2Apollo.from(client.mutate(new CreateWorkspace(workspace)))
        .observeOn(AndroidSchedulers.mainThread())
        .subscribeWith(new
DisposableObserver<Response<CreateWorkspace.Data>>() {
            @Override
            public void onNext(Response<CreateWorkspace.Data>
dataResponse) {
                if (dataResponse.errors().isEmpty())
                    workspaceController.createWorkspace(
                            dataResponse.data().createWorkspace(),
                            workspace);
            }
            @Override
            public void onError(Throwable e) {
                progressBar.setVisibility(View.GONE);
            }
            @Override
```

```
        public void onComplete() {
        }
    });
```

Создается объект “workspace”, его поля заполняются необходимыми данными. А после клиент Apollo вызывает метод «mutate», аргументом которой является рассмотренная выше мутация с атрибутом “workspace”. После того как запрос выполнится, проверяются входные данные. Если ошибок нет, то извлекаем созданный сервером идентификатор, а на стороне клиента создаем и вносим объект в локальную БД.

5 Документация

5.1 Назначение

Программное обеспечение “TimeTracker” предназначено для людей, желающих отслеживать и контролировать свое личное время. А именно, программное обеспечение “Time Tracker” служит для учета рабочего времени выполнения заданий в собственных проектах и целях.

Программа позволяет создавать рабочее пространство, проекты и задачи, отслеживать их выполнение и получать статистику.

5.2 Условия выполнения программы

Для запуска мобильного приложения необходимо устройства на базе ОС Android с версией не меньше 4.4.4 KitKat и доступом в интернет. Вход в приложение осуществляется с помощью google – авторизации, если у пользователя нет google-аккаунта, ему предлагается зарегистрировать его [14].

5.3 Выполнение программы

При открытии Android приложения пользователя приветствует экран авторизации (Рисунок 5.1), которую он должен пройти, чтобы получить доступ к основному функционалу.

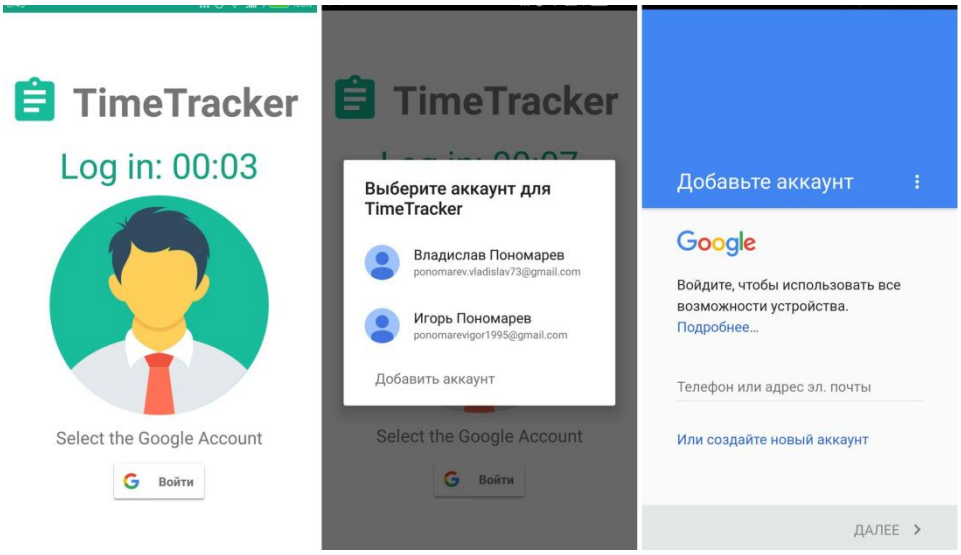

Рисунок 5.1. Экран авторизации

Если пользователь успешно прошел авторизацию, он переходит на основной экран. Используя меню (Рисунок 5.2), пользователь может перемещаться между вкладками “workspaces”, “projects”, “tasks” или выйти из приложения.

Если пользователь авторизуется впервые, то для него создаются проект и рабочее пространство по умолчанию.

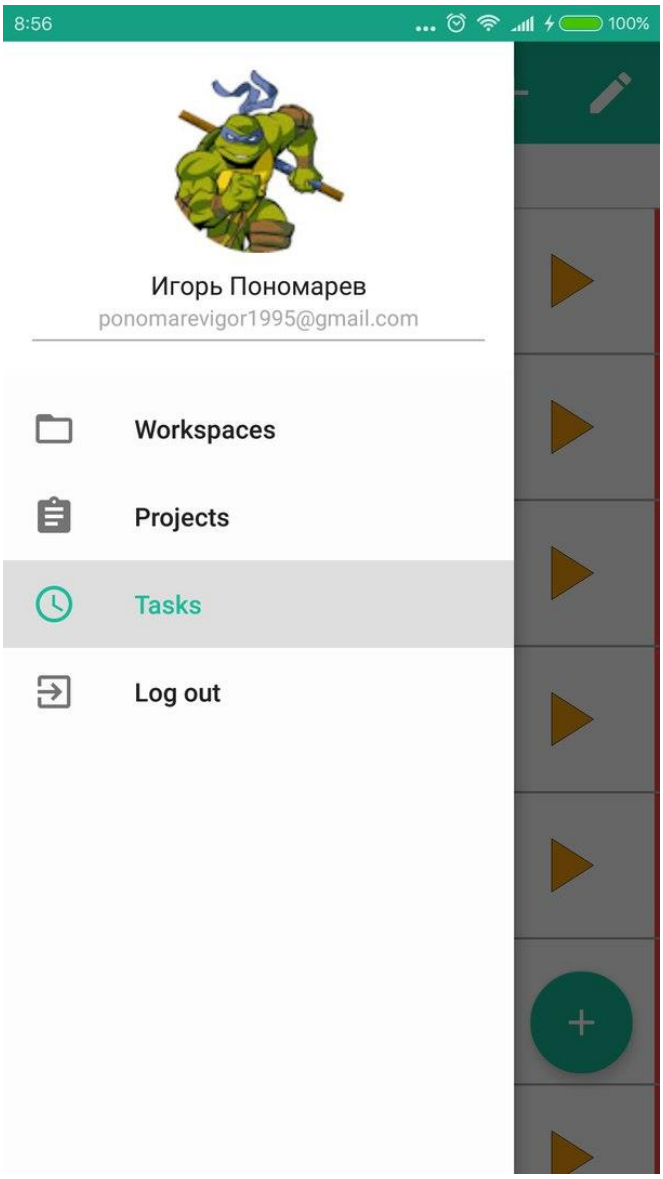

Рисунок 5.2 - Меню – навигация по приложению

По умолчанию пользователь переходит на экран с задачами, с которыми он может взаимодействовать. Приложение выводит весь список задач со всех проектов, но используя верхнее меню навигации, пользователь может выбрать интересующий его проект, и будут загружены задачи только этого проекта

(Рисунок 5.3). Задачи загружаются с сервера и хранятся в локальной БД. Нажав на кнопку в нижнем углу экрана, выводится диалоговое окно, в котором пользователю предлагается ввести имя и описание новой задачей. Задача будет добавлена в проект, который был выбран пользователем или в проект “No project” по умолчанию (Рисунок 5.4).

Пользователь может удалить задачу, зажав ее и переместив влево (Рисунок 5.5), и перейти на экран редактирования и статистики задачи обычным кликом на задачу. Также в верхнем меню предоставляется возможность перейти на экран создания нового проекта или отредактировать выбранный.

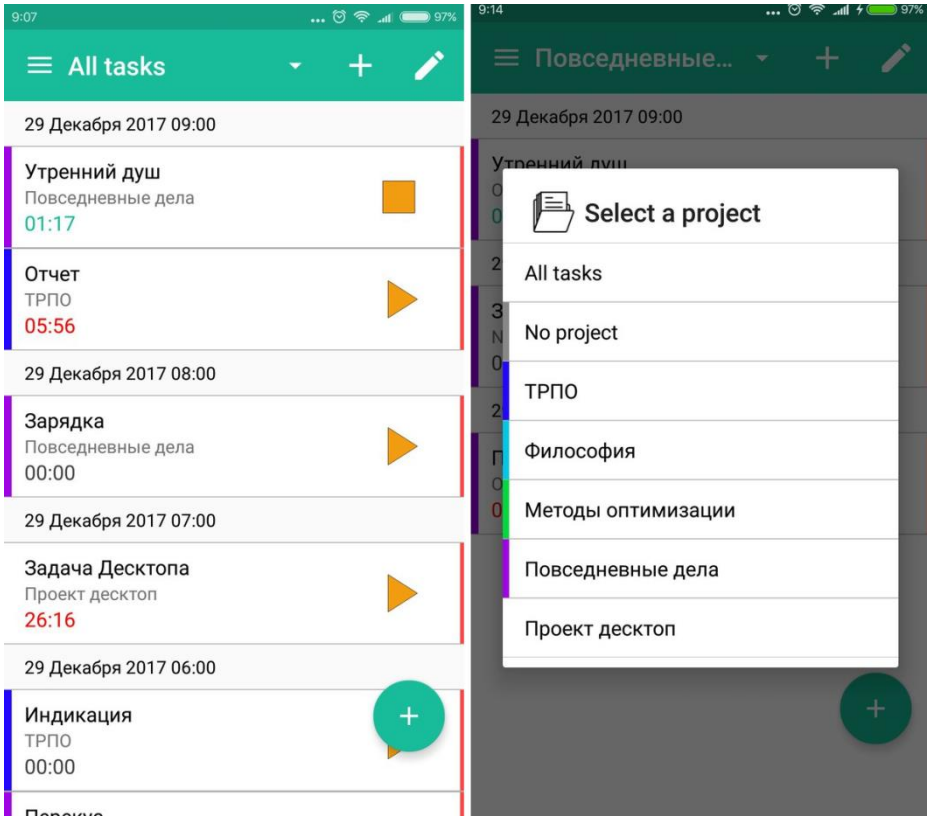

Рисунок 5.3. Вывод списка задач и диалоговое окно с выбором проекта

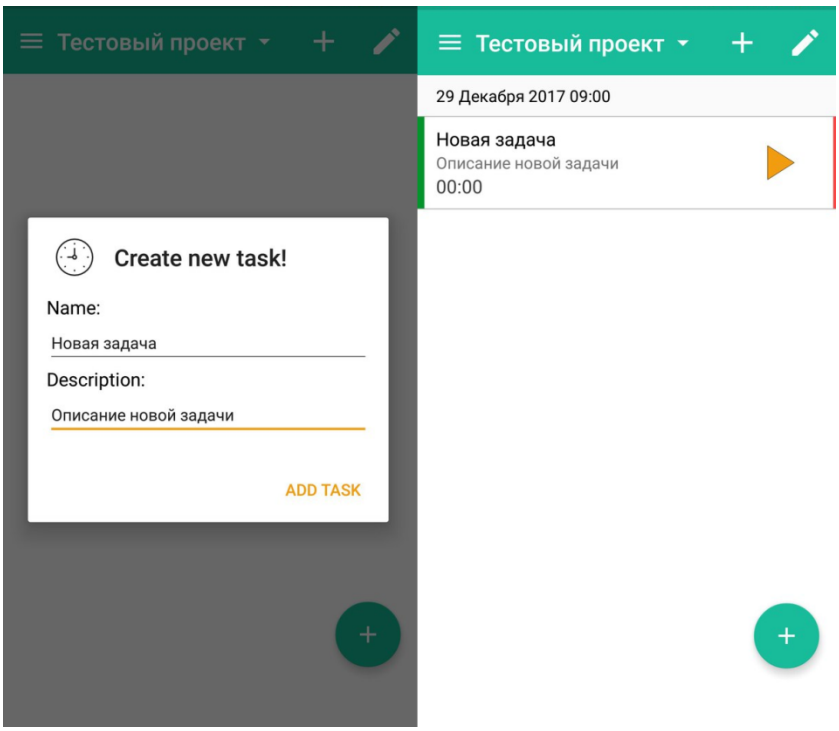

Рисунок 5.4. Интерфейс создание новой задачи

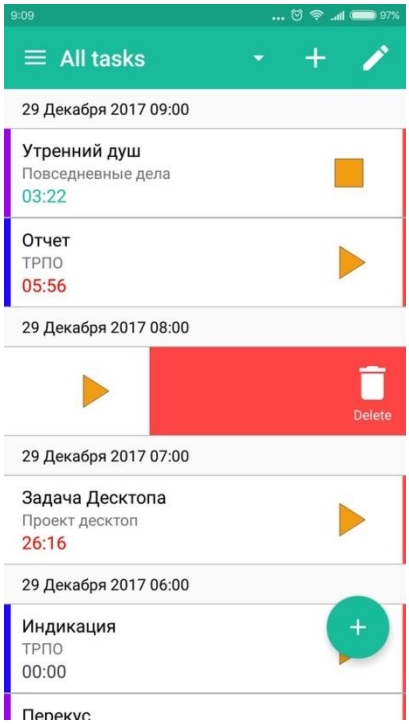

Рисунок 5.5. Удаление задачи

С помощью активной кнопки задачи, пользователь может начинать или завершать задачу. В зависимости от состояния, задача имеет отличительный признак. Каждый раз, когда пользователь начинает задачу, создается сущность TimeEntry, которая добавляется к задаче. При завершении задачи эта сущность становится доступной для редактирования.

Рисунок 5.6 демонстрирует все возможные состояния задачи. Первая задача активная, идет отсчет ее времени, и, завершив ее, окончательно сформируется TimeEntry. Вторая задача уже завершена, но ее можно восстановить, и для нее создаётся новая TimeEntry. Последняя задача только создана и даже не начиналась выполняться. Каждое смена состояния задачи сопровождается отправлением запроса серверу об этом. Если сервер корректно отработал запрос и вернул положительный результат, то задача меняет свое состояние.

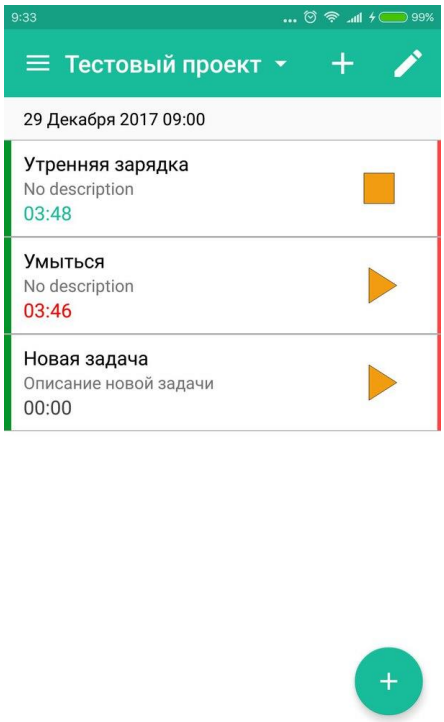

Рисунок 5.6. Интерфейс состояния задач

Как уже было описано выше пользователь, кликнув на задачу, может перейти на экран ее редактирования (Рисунок 5.7). Этот экран имеет две страницы: первая позволяет отредактировать данные задачи, а вторая выводит статистику задачи (ее TimeEntry) с возможностью отредактировать каждую из них (Рисунок 5.8). Пользователь может изменить имя, описание и проект задачи, а для TimeEntry начало, конец и продолжительность выполнения.

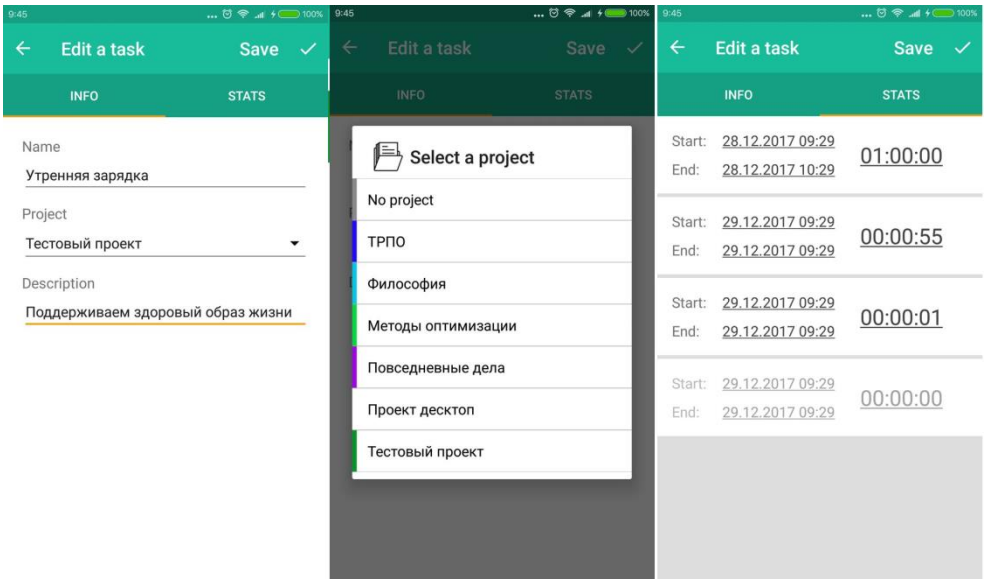

Рисунок 5.7. Интерфейс редактирование задачи

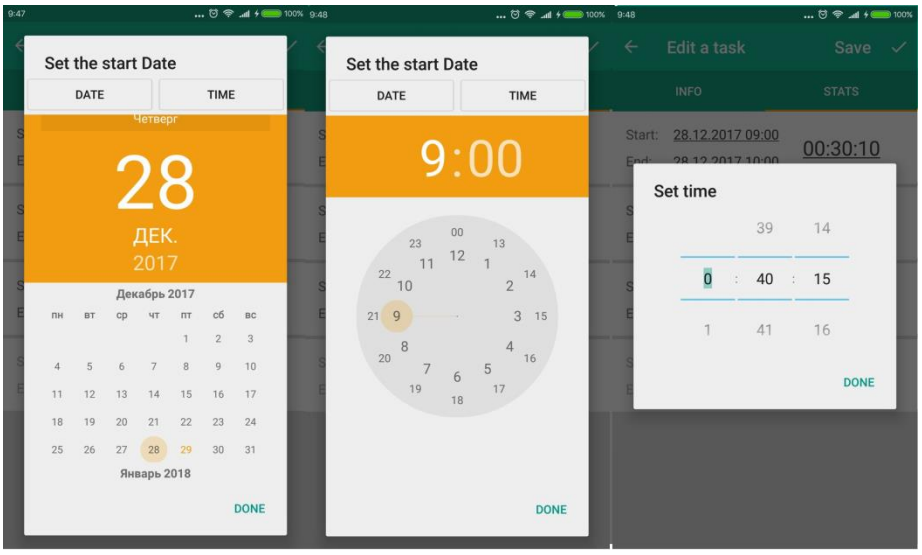

Рисунок 5.8. Пример редактирования TimeEntry

Закончив редактирование, пользователь нажимает кнопку “Save” и запрос отправляется серверу. Если пользователь передумал вносить изменения, он может нажать возврат, и он вернется на экран задач.

Если пользователь находится на экране задачи, приложение выводит весь список проект со всех рабочих пространств, но используя верхнее меню навигации, пользователь может выбрать интересующее его рабочее пространство, и будут загружены проекты только этого рабочего пространства (Рисунок 5.9). Нажав на кнопку в нижнем углу экрана, пользователь переходит на экран создания проекта, в котором пользователю предлагается ввести имя, описание и выбрать цвет (Рисунок 5.10). Проект будет добавлен в рабочее пространство, который был выбран пользователем или в “No workspace” по умолчанию.

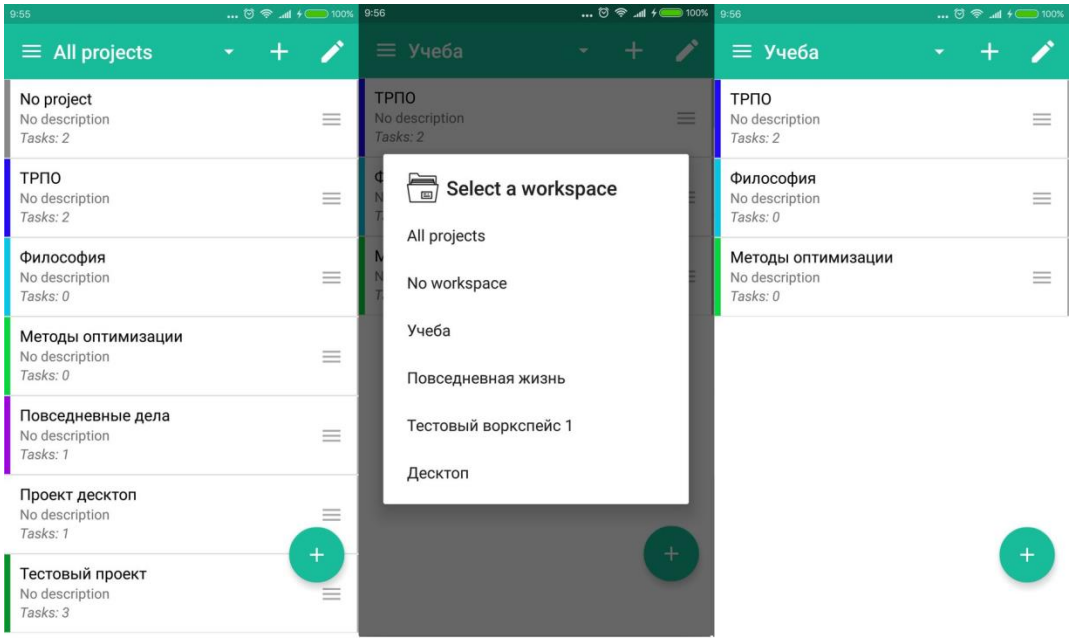

Рисунок 5.9. Интерфейс вывода проектов

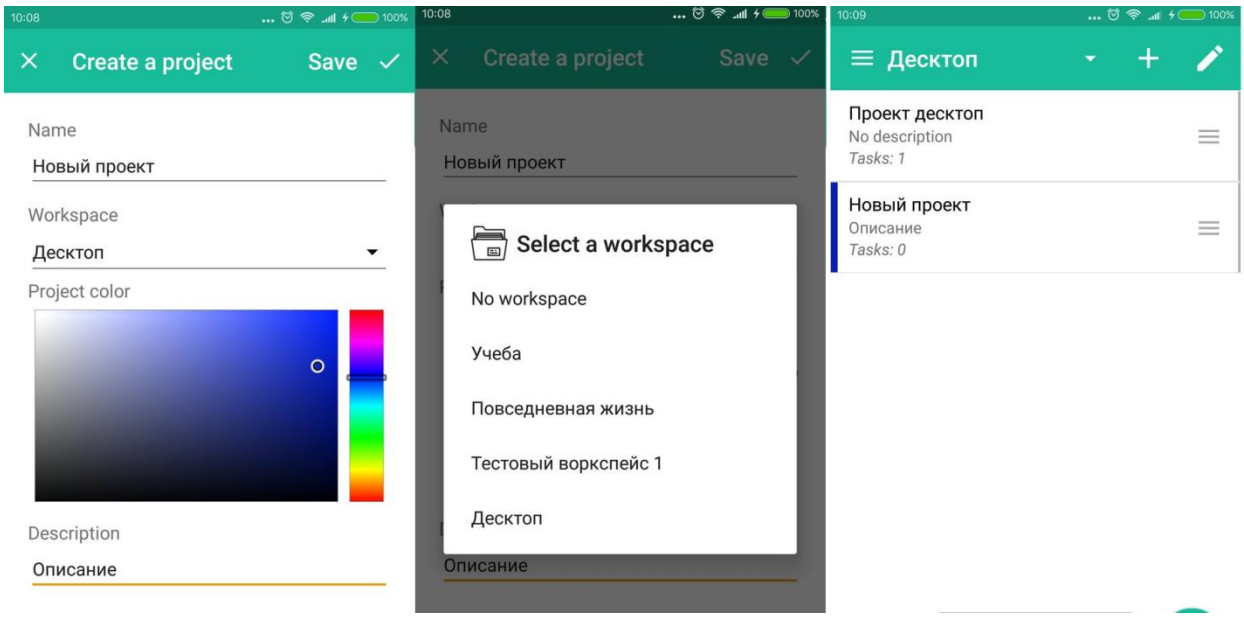

Рисунок 5.10. Создание проекта.

Кликнув на проект, пользователю предоставляется меню возможных действий с ним: переход к задачам проектам (Рисунок 5.11), получение информации о проекте (Рисунок 5.12), редактирование и удаление. Информация о проекте включает информацию о самом проекте и статистики выполнения задач.

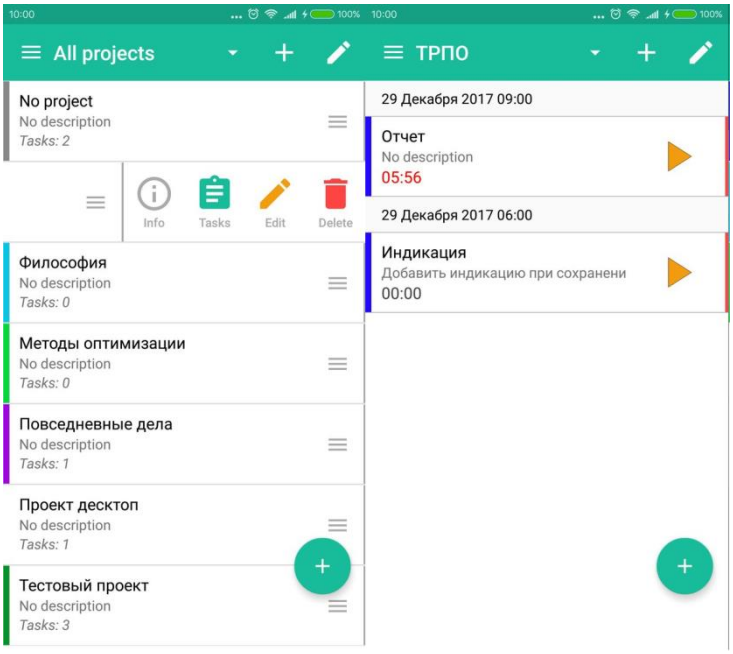

Рисунок 5.11. Переход к задачам проекта

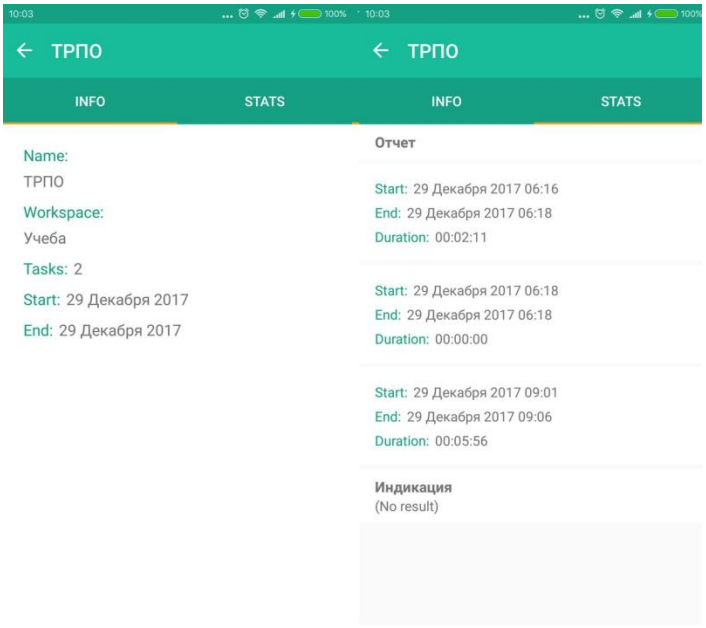

Рисунок 5.12. Информация о проекте

Если пользователь находится на экране рабочего пространство, приложение выводит весь список данной сущности. Нажав на кнопку в нижнем углу экрана, выводится диалоговое окно, в котором пользователю предлагается ввести имя и описание нового рабочего пространства. Кликнув на элемент, пользователю предоставляется возможный функционал с данной сущностью (Рисунок 5.13).

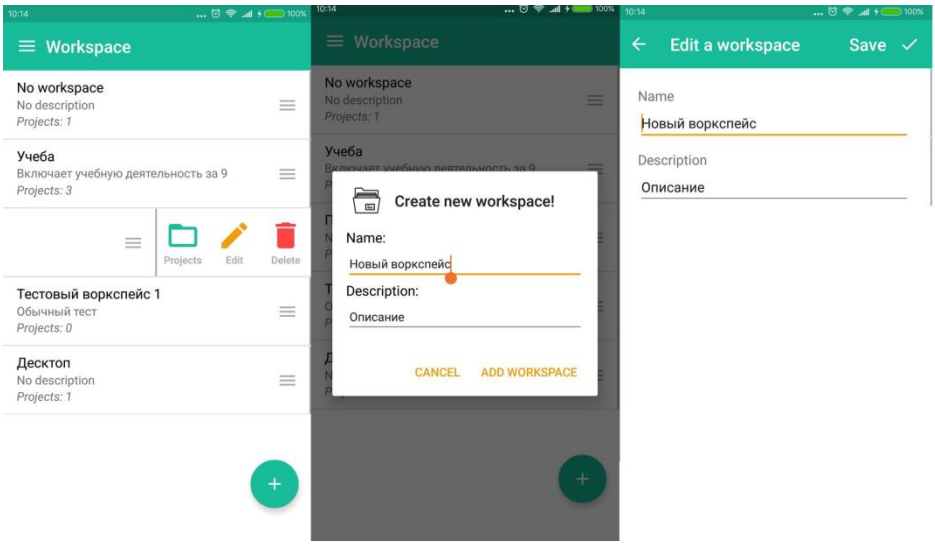

Рисунок 5.13. Вывод (слева), создание (в центре), редактирование (справа) рабочего пространства.

Заключение

Результатом выполнения курсового проекта “TimeTracker” является два клиента: приложения для операционных систем Windows, Android и сервер, обслуживающий клиентов. Разработанное программное обеспечение служит для учета и контроля рабочего времени выполнения заданий в собственных проектах и целях.

Разработка включала в себя несколько этапов, каждый из которых сопровождался UML диаграммами и их описанием. На этапе анализа были выявлены требования к структуре и поведению приложения. Были выявлены варианты использования и их спецификации, классы анализа, диаграммы последовательности для вариантов использования и конечные автоматы классов анализа. На этапе проектирования система практически не претерпела изменений, что свидетельствует о качественном выполнении работ на этапе анализа.

Был сформирован окончательный вид проектных классов, диаграммы последовательностей и конечные автоматы для операций проектных классов.

ПО использует две системы непрерывной интеграции Travis CI и AppVeyor. Сборки выполняются успешно, создается релизное приложение.

Настольное приложение реализовано на языке C++ с использованием фреймворка Qt в среде разработки QtCreator, а мобильное приложение на языке Java в AnroidStudio. Серверное приложение реализовано на языке Java с использованием фреймворка Spring в IntelliJ IDEA. Развернуто серверное ПО на выделенном виртуальном веб-сервере компании Amazon.

Авторизация пользователей на клиентах реализована при помощи аутентификации через Google аккаунт.

Для клиентов в качестве оформления и дизайна использовался Material Design от Google.

Функционал готового приложения включает в себя:

Авторизация пользователя.

Создание рабочего пространства, проекта, задач.

Редактирование сущностей.

Синхронизация с сервером.

Запуск и останов таймера ведения задач.

Приложение сопровождено соответствующей документацией. Итогом курсового проекта программный комплекс, полностью удовлетворяющий заявленным требованиям.

Список использованных источников

1 Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектноориентирован ный анализ и проектирование, 2е издание. – Пер. с англ. – СПб: Символ Плюс, 2007. – 624 с., ил;

2 A query language for your API [Электронный ресурс] // Режим доступа: http://graphql.org/ (дата обращения 29.12.2017);

3 Spring Framework 5.0 // Режим доступа: http://spring.io (дата обращения 29.12.2017);

4 Основы тестирования программного обеспечения [Электронный ресурс] // Режим доступа: https://www.intuit.ru/studies/courses/48/48/lecture/1432 (дата обращения 29.12.2017);

5 Realm — кроссплатформенная мобильная база данных [Электронный ресурс] // Режим доступа: https://jetruby.com/ru/blog/realm-mobilnaya-baza-dannyh/ (дата обращения 29.12.2017);

6 Gradle Build Tool Guide [Электронный ресурс] // Режим доступа: http://gradle.org /docs/ (дата обращения 29.12.2017);

7 Built for developers [Электронный ресурс] // Режим доступа: http://github.com (дата обращения 29.12.2017);

8 Google Developers Docs [Электронный ресурс] // Режим доступа: http://developers.google.com (дата обращения 29.12.2017);

9 Travis CI Guide [Электронный ресурс] // Режим доступа: http://docs.travis-ci.com (дата обращения 29.12.2017);

10 Continuous Integration and Deployment service for Windows [Электронный ресурс] // Режим доступа: http://appveyor.com/docs (дата обращения 29.12.2017);

11 Docker Cloud^ The official cloud service for continuously delivering Docker applications. [Электронный ресурс] // Режим доступа: http://cloud.docker.com (дата обращения 29.12.2017);

12 Docker Documentation [Электронный ресурс] // Режим доступа: http://docs.docker.com (дата обращения 29.12.2017);

13 Amazon Web Services [Электронный ресурс] // Режим доступа: http://aws.amazon.com (дата обращения 29.12.2017);

14 Android Developers Guide [Электронный ресурс] // Режим доступа: http://developer.android.com (дата обращения 29.12.2017);