СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 7](#_Toc420419586)

[ВВЕДЕНИЕ 8](#_Toc420419587)

[1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ 9](#_Toc420419588)

[1.1 Архитектура мобильного приложение для Android 9](#_Toc420419589)

[1.2 Архитектура клиент-сервер 10](#_Toc420419590)

[1.3 Система управления базами данных 12](#_Toc420419591)

[1.4 СУБД SQLite 14](#_Toc420419592)

[2 ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 17](#_Toc420419593)

[2.1 Анализ предметной области 17](#_Toc420419594)

[2.2 Постановка задачи 18](#_Toc420419595)

[2.3 Обзор существующих аналогов 19](#_Toc420419596)

[2.4 Определение границ первой версии программного комплекса 20](#_Toc420419597)

[2.5 Обзор технологий, используемых при реализации программного комплекса 21](#_Toc420419598)

[3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 23](#_Toc420419599)

[3.1 Анализ требований 23](#_Toc420419600)

[3.2 Разработка архитектуры программного комплекса 24](#_Toc420419601)

[3.2.1 Проектирование структуры БД 26](#_Toc420419602)

[3.2.2 Проектирование уровня доступа к данным 27](#_Toc420419603)

[3.2.3 Уровень бизнес-логики 27](#_Toc420419604)

[4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 29](#_Toc420419605)

[4.1 Реализация базы данных 29](#_Toc420419606)

[4.2 Реализация уровня доступа к данным 31](#_Toc420419607)

[4.3 Реализация уровня бизнес-логики 32](#_Toc420419608)

[4.4 Реализация уровня представления 34](#_Toc420419609)

[5 ТЕСТИРОВАНИЕ И ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ 39](#_Toc420419610)

[5.1 Подготовка устройств для тестирования приложения 39](#_Toc420419611)

[5.2 Тестирование приложения 41](#_Toc420419612)

[5.3 Публикация приложения в магазин Google Play 43](#_Toc420419613)

[6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ФИТНЕС ПРИЛОЖЕНИЯ ДЛЯ УСТРОЙСТВ ПОД УПРАВЛЕНИЕМ ОПЕРАЦИОННОЙ СИСТЕМЫ АНДРОИД 44](#_Toc420419614)

[6.1 Характеристика фитнес приложения для устройств под управлением операционной системы Андроид 44](#_Toc420419615)

[6.2 Расчёт стоимостной оценки затрат 44](#_Toc420419616)

[6.3 Расчёт стоимостной оценки результата 45](#_Toc420419617)

[6.4 Расчёт экономии основных видов ресурсов в связи с использованием нового ПС 47](#_Toc420419618)

[6.5 Расчёт показателей эффективности использования программного продукта 50](#_Toc420419619)

[7 ЭРГОНОМИЧЕСКАЯ ОЦЕНКА ПРОГРАММНОГО ПРОДУКТА ФИТНЕС ПРИЛОЖЕНИЯ ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ АНДРОИД 54](#_Toc420419620)

[ЗАКЛЮЧЕНИЕ 58](#_Toc420419621)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 59](#_Toc420419622)

[ПРИЛОЖЕНИЕ А 60](#_Toc420419623)

[ПРИЛОЖЕНИЕ Б 78](#_Toc420419624)

## ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяют следующие обозначения и сокращения:

API – Application Programming Interfaces

BLL – Business Logic Layer

DAL – Data Access Layer

SDK – Software Development Kit

UML – Unified Modeling Language

JSON – JavaScript Object Notation

БГУИР – Белорусский Государственный Университет

Информатики и Радиоэлектроники

БД – база данных

ПО – программное обеспечение

ПП – программный продукт

СУБД – система управления базами данных

## ВВЕДЕНИЕ

Мы живем в мире, где высокотехнологичные устройства, такие как смартфоны и планшеты стали обыденным и повсеместным явлением. Сейчас сложно представить человека, который бы не знал о том, что такое смартфон или планшет, а их небольшая цена позволили многим гражданам обзавестись этими «компьютерами на ладони». И действительно, мощность современных устройств в разы превышает мощность компьютеров, модели которых были популярны 20-30 лет назад. Вкупе со встроенными датчиками и приборами, большим экраном, батареей и подключением к Интернет, смартфоны и планшеты способны решать огромное количество задач, помогаю пользователю не только быть на связи, но и работать с информацией быстро и удобно.

С развитием мобильных аппаратов и переходом их на стадию смартфонов, на рынке мобильных экосистем появились два лидера. Это компания Apple с операционной системой iOs, и компания Google с операционной системой Android. Под управлением этих операционных систем работает более чем 2.5млрд устройств[1], что составляет долю рынка в 90%[1]. Создав экосистемы на базе своих операционных систем, эти компании позволили большому количеству разработчиков легко создавать и распространять свои приложения для мобильных устройств через магазины приложений. В частности, для магазина Google Play, где пользователи Android могут приобрести или скачать бесплатные приложения, количество доступных программ стало больше, чем 1млн. [2].

В наше время смартфон может выступать в роли интеллектуального помощника для занятий спортом. Разнообразные спортивные приложения предлагаю свои программы тренировок, сохраняют статистику выполненных упражнений пользователем, мотивируют и поощряют его за выполненные упражнения, напоминают о времени занятия спортом и т.д. Все эти функции помогают построить тренировочный процесс более правильно, эффективно и интересно.

В рамках этого дипломного проекта будет рассмотрена разработка фитнес приложения, которое предлагает пользователям наборы тренировок, сохраняет статистику выполнения подходов на сервере, напоминает о времени тренировок.

## 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

* 1. Single Page Application

Single Page Application – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все javascript-файлы (модули, виджиты, контролы и т.д.) , а также файлы CSS вместе с загрузкой самой страницы. Если приложение достаточно сложное и содержит богатый функционал, то количество файлов со скриптами может достигать нескольких сотен, а то и тысяч. А загрузка всех скриптов не означает, что при загрузке сайта будут загружены сразу все сотни и тысячи файлов со скриптами. Для решения проблемы загрузки большого количества скриптов в SPA призван API под названием AMD.

Application Programming Interface (API) - это интерфейс программирования, набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.

AMD реализует возможность загрузки скриптов по требованию. То есть, если для “главной станицы” одностраничного портала потребовалось 3 скрипта, они будут загружены стразу перед стартом программы. А если пользователь кликнул на другую страницу одностраничного порталато принцип AMD загрузит модуль (скрипт + разметка) только перед тем как перейти на эту страницу. В целом, SPA обладает несколькими весьма весомыми плюсами.

Первым плюсом стоит отметить тот факт, что приложения на SPA отлично работают на устройствах как стационарных, так и мобильных. Персональные компьютеры, планшеты, смартфоны и простые современные телефоны могут беспрепятственно работать с сайтами построенных по принципу SPA. Это значит, что создав одно приложение, получается гораздо большая аудитория пользователей, нежели при использовании стандартного подхода.

**Второй плюс – богатый пользовательский интерфейс**, так называемый User Experience. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией (в некоторых случаях).

**Третий плюс – SPA существенно сокращает так называемые “хождения по кругу”, то есть загрузку одного и того же контента снова и снова.** Если ваш сайт использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона. Кэширование данных на данном этапе развития достигло высочайших результатов, но если нечего кэшировать, то и время, и ресурсы на это не тратятся.

* 1. JavaScript

JavaScript — прототипно-ориентированный сценарный язык программирования. Является диалектом языка ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

JavaScript является объектно-ориентированным языком, но используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

– функции как объекты первого класса;

– автоматическое приведение типов;

– автоматическая сборка мусора;

– анонимные функции.

В языке отсутствуют такие полезные вещи, как:

– модульная система: JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;

– стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, базовых типов для бинарных данных;

– стандартные интерфейсы к веб-серверам и базам данных;

– система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

* 1. Система управления базами данных

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Можно выделить следующую классификацию СУБД

По модели данных:

– иерархические;

– сетевые;

– реляционные;

– объектно-ориентированные;

– объектно-реляционные;

По степени распределённости:

– локальные СУБД (все части локальной СУБД размещаются на одном компьютере);

– распределённые СУБД (части СУБД могут размещаться на двух и более компьютерах).

По способу доступа к БД:

– Файл-серверные. В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера. Недостатками являются потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД. На данный момент файл-серверная технология считается устаревшей. Примеры: Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

– Клиент-серверные. Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Примеры: Oracle, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Caché, ЛИНТЕР.

– Встраиваемые. Встраиваемая СУБД — СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети. Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL, либо через специальные программные интерфейсы. Примеры: OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, Microsoft SQL Server Compact, ЛИНТЕР.

Ниже перечислены основные функции СУБД:

– управление данными во внешней памяти (на дисках);

– управление данными в оперативной памяти с использованием дискового кэша;

– журнализация изменений, резервное копирование и восстановление базы данных после сбоев;

– поддержка языков БД (язык определения данных, язык манипулирования данными);

Обычно современная СУБД содержит следующие компоненты:

– ядро, которое отвечает за управление данными во внешней и оперативной памяти, и журнализацию;

– процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;

– подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;

– сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

* 1. Архитектура Module-Widget-Service

При работе с JavaScript, разработчик зачастую сталкивается с проблемой огромного числа зависимостей компонентов друг от друга. Архитектура Module-Widget-Service является одним из вариантов решения данной проблемы.

В основе подхода лежат две идеи:

– JavaScript виджеты — небольшие модули, каждый из которых «владеет» определённой частью веб-странички (т.е. всё управление этой частью странички происходит исключительно через методы этого модуля, а не через прямую модификацию DOM). Виджет отвечает исключительно за функциональность, но не за внешний вид; поэтому прямая модификация части DOM, которым «владеет» виджет, снаружи виджета допускается — но только для дизайнерских задач (для архитектуры и общей сложности приложения нет принципиальной разницы между коррекцией внешнего вида через CSS или jQuery).

– Глобальный диспетчер событий. Взаимодействие между виджетами осуществляется путём посылки сообщений глобальному диспетчеру, а уже он принимает решение что с этим сообщением делать — создать/удалить виджеты, вызвать методы других виджетов, выполнить дизайнерский код и другое. В отличие от динамического подхода к обработке событий (когда обработчики конкретного события добавляются/удаляются в процессе работы) статический диспетчер упрощает понимание и отладку кода. Безусловно, есть задачи, для которых нужны именно динамические обработчики событий, но в большинстве случаев это избыточное усложнение, поэтому всё, что можно, делается статическими обработчиками.

* 1. Среда реализации

*Visual Studio* — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения).

WebStorm — интегрированная среда разработки на JavaScript, CSS & HTML от компании JetBrains, разработанная на основе платформы IntelliJ IDEA. WebStorm обеспечивает автодополнение, анализ кода на лету, навигацию по коду, рефакторинг, отладку, и интеграцию с системами управления версиями. Важным преимуществом интегрированной среды разработки WebStorm является работа с проектами (в том числе, рефакторинг кода JavaScript, находящегося в разных файлах и папках проекта, а также вложенного в HTML). Поддерживается множественная вложенность (когда в документ на HTML вложен скрипт на Javascript, в который вложен другой код HTML, внутри которого вложен Javascript) — то есть в таких конструкциях поддерживается корректный рефакторинг.

## 2 ПОСТАНОВКА ЗАДАЧИ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Анализ предметной области

Нынешний темп жизни человека, в совокупности с сидячей в своём большинстве работе программиста, предоставляет слишком мало свободного времени для ведения здорового образа жизни. К сожалению, сегодня обычные занятия спортом пользуются слишком малой популярностью. Однако, эти занятия можно совместить с чем-то необычным, вернув таким образом интерес к занятиям спортом. Так появилась идея создания городских игр. Эти игры сочетают в себе как неплохую физическую подготовку, связанную с постоянным бегом по улицам города, так и интеллектуальную разминку, включающую в себя загадки, шифры и другие головоломки. В последнее время такие игры начали приобретать всё более массовый характер, а их специфика выходить за рамки города. Теперь такие игры несут в себе, в дополнение к уже имеющимся плюсам, образовательный и культурный характер. Так как во время решения очередной загадки команда зачастую знакомиться с некоторыми интересными историческими фактами и посещает различные культурные места города.

Обычная концепция игры имеет следующую схему. Команда получает у координаторов загадку, указывающую на определенное место в городе. Прибыв на место, команда обнаруживает тайник, содержащий новую загадку с новым ориентиром. Однако при такой концепции использование компьютерных технологий сводится к минимуму. В последние годы, особенно для студентов инженерных специальностей, компьютерные технологии стали неотъемлемой частью досуга и учебы. Так возникла идея новой концепции городской игры, которая будет тесно взаимодействовать с компьютерными технологиями

Программный продукт, разрабатываемый в рамках дипломного проектирования, предназначен для сопровождения интеллектуально-спортивной игры, обеспечения её лучшей интерактивности и качества проведения. Цель приложения обеспечить участникам игры доступ в систему для получения заданий и ввода ответов во время процесса игры. В ходе разработки приложения были проанализированы слабые и сильные места схожих систем, выделены те важные аспекты, которые необходимы пользователям, и все это было реализовано в данном проекте.

2.2 Постановка задачи

В рамках дипломного проекта поставлена цель разработать веб-сайт, который предоставляет пользователям базовый функционал для участия в игре. Разрабатываемая система должна соответствовать общей концепции игры и являться удобным дополнением в процессе прохождения городской игры.

Важной особенностью разрабатываемой системы является персонализация, под которой понимается возможность заходить под своим аккаунтом на разных устройствах, и которая будет поддерживать синхронизацию прогресс игры на разных устройствах для одного и того же пользователя.

Система должна обладать понятным и удобным интерфейсом, который позволяет пользователю, впервые зашедшему в игру, без особых усилий выполнить необходимые действия. Сайт должен одинаково хорошо отображаться на всех популярных на данный период мобильных платформах, с различной конфигурацией программного обеспечения, аппаратного обеспечения и размеров экрана. Также необходимым является и автоматическое растягивание элементов интерфейса, чтобы выглядеть гармонично на устройствах с различными размерами экрана и плотностью пикселей.

Не менее важным является и скорость работы системы. Так как сама игра задает достаточно быстрый темп, основывающийся на соревновательном элементе, система должна соответствовать этим критериям.

Для достижения поставленной цели необходимо решить следующие задачи:

- С помощью языка программирования C# и фреймфорка ASP.NET WebApi, реализовать серверную часть сайта.

- С помощью языка программирования JavaScript и фреймфорка Module-Widget-Service, реализовать клиентскую часть сайта.

- С помощью языка программирования SQL и библиотеки NHibernate реализовать часть хранения данных вебхостинга.

2.3 Обзор существующих аналогов

Мобильное приложение, разрабатываемое в рамках дипломного проекта, предоставляет возможность пользователям заниматься споротом с помощью программ популярных упражнений, не выходя из дома. На сегодняшний день существует огромное количество мобильных проектов, предоставляющих аналогичные возможности своей аудитории. Основная особенность разрабатываемого приложения, это обучение и контроль за правильностью выполнения подходов, а также набор тренировок, для достижения эффекта от занятия спортом.

Среди мобильных приложений, которые предоставляют своим посетителям возможность заниматься спортом с помощью программ упражнений, можно выделить группу приложений Runtastic. Программы Runtastic представляют собой отдельные приложения для каждого популярного упражнения. На данный момент представлены программы для следующих упражнений: пресс, подтягивания, отжимания, приседания. Хоть каждая программа и является отдельным приложением и устанавливается пользователем отдельно от остальных, благодаря возможностям операционной системы Андроид в каждом из приложений есть доступ к статистике и запуску других приложений Runtastic.

Скачав приложение Runtastic, пользователь может выбрать уровень выполняемых приложений по сложности. Простые уровни включают в себя 10 дневные тренировки, средний и сложный уровни обычно длятся 20 дней и включают в себя подходы с большим количеством выполняемых упражнений. Один из минусов для пользователей является то, что средний и сложный уровень не доступны в бесплатной версии программы, и пользователь должен приобрести платную версию приложений. Такая же неприятная особенность касается и рейтинга людей, занимающихся спортом с помощью программы Runtastic. Пользователи, которые приобрели платную версию программы, могут видеть свои достижения на фоне достижений других пользователей в таблице результатов. Для пользователей с бесплатной версией этот функционал ограничен.

Также программа Runtastic сохраняет и синхронизируют статистику пользователя, позволяет просматривать пользователю количество выполненных подходов за все время использования приложений. Вся эта информация представлена в удобном виде с помощью графиков, имеется возможность фильтровать графики для просмотра интересующей информации.

Еще одной особенностью программ Runtastic является система виртуальных наград, которыми награждаются пользователи за достижения тех или иных показателей. В частности, награды выдаются за рекорды в количестве выполненных упражнений, за окончания уровня. Все это призвано еще больше мотивировать пользователей выполнять упражнения.

Минус занятия спортом с приложениями Runtastic, что на первый взгляд кажется не так, является очень глубокая привязанность к смартфону при занятии спортом. Это значит, что при подтягивании нужно держать аппарат в кармане, и он с помощью датчиков изменения ускорения определяет выполненных подход. Такая же ситуация и с другими упражнениями. Проблема в том, что под управлением операционной системы Android существует большое множество аппаратов, и каждый из них имеет разные по точности датчики, а значит программа не всегда может определить, выполнено или нет упражнение. Это мешает удобному использованию приложений.

Таким образом, исходя из представленного выше описания, выделим основные отличия от имеющихся аналогов:

– весь функционал доступен бесплатно;

– реализация основных элементов направлена на быструю и удобную работу с приложением, выполненные упражнения будут отмечаться пользователем нажатием соответствующей кнопки на экране интерфейса;

– Все упражнения объединены в одно приложение.

2.4 Определение границ первой версии программного комплекса

Первая версия приложения будет включать в себя набор упражнений и тренировок, реализацию возможности персонализации и синхронизации данных пользователя, возможности просмотра статистики и таблицы достижений пользователей.

Тренировки будут включать в себя 7 упражнений. Это упражнения на пресс, отжимания, подтягивания, приседания, отжимания обратным хватом, упражнения на брусьях. Для этих упражнений будут доступны тренировки, каждый этап которых состоит из 5 подходов. Каждый подход — это определенное количество выполняемых упражнений. Пользователь выбирает, смог ли он выполнить подход или нет. Если все 5 подходов выполнены, приложение отмечает этап выполненным и повышает сложность следующего этапа, иначе пользователь будет пытаться выполнить подход в следующий раз. Седьмая тренировка будет состоять из одного подхода. Это будет бег и в качестве задания для пользователя будет предложено пробежать определенное количество метров. Если пользователь выполняет задание, в следующий раз сложность повышается. Пользователь имеет возможность выбрать те упражнения, которые ему интересны – и сформировать тренировки на каждый день.

Каждый пользователь может заниматься спортом с помощью приложения, зайдя под своим аккаунтом или нет. Если пользователь выбирает регистрацию и вход в приложение через АПИ социальной сети Facebook, тогда для пользователя заводится аккаунт, а если он уже существует, то на устройство пользователя синхронизируется информация об его текущих результатах и тренировках. Если пользователь не хочет входить под своим аккаунтом, то он может без помех заниматься спортом, но в таком случае при удалении приложения или установки его на другое устройство он потеряет статистику и набор текущих тренировок.

Статистика пользователей сохраняется как локально на устройстве, так и синхронизируется с удаленным сервером при условии, что пользователь вошел под своим аккаунтом. Статистика включает в себя информацию об дате и количестве выполненных тех или иных упражнениях. Также, если пользователь зашел под своим аккаунтом, ему доступна таблица результатов других пользователей, где он может увидеть, какое место по отношению к другим пользователям, занимающимся по нашему приложению, он занимает. Таблица достижений призвана мотивировать пользователя заниматься интенсивнее для продвижения выше в таблице.

2.5 Обзор технологий, используемых при реализации программного комплекса

ASP.NET Web APIпозволяет с легкостью создавать службы HTTP для широкого диапазона клиентов, включая браузеры и мобильные устройства. Веб-API ASP.NET идеально подходит для разработки приложений RESTful на платформе .NET Framework.

Microsoft SQL Server— система управления реляционными базами данных (СУРБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

NHibernate — ORM-решение для платформы Microsoft .NET, портированное с Java. Это бесплатная библиотека с открытым кодом, распространяется под лицензией GNU Lesser General Public License.

NHibernate позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов. NHibernate 2.0.1, выпущенная в конце сентября 2008 года предоставляет бо́льшую часть возможностей Hibernate 3.

RequireJS *—* является реализацией AMD (Asynchronous Module Definition), API для объявления модулей и их асинхронной загрузки «на лету», когда они понадобятся. Это разработка Джеймса Бёрка (James Burke) и она достигла версии 1.0 после двух лет разработки. RequireJS помогает организовать код с помощью модулей и будет управлять асинхронной и параллельной загрузкой ваших файлов. Так как скрипты загружаются только при необходимости и параллельно, это сильно уменьшает время загрузки страницы. На базе RequireJS хорошо строится архитектура Module-Widget-Service.

LESS *—* это препроцессор CSS, позволяющие писать ясный CSS с использованием программных конструкций вместо статических правил. LESS вдохновлен Sass. Sass — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей. Sass был спроектирован с целью как упростить, так и расширить CSS, в первых версиях фигурные скобки были исключены из синтаксиса (этот синтаксис называется sass). LESS разработан с целью быть как можно ближе к CSS, поэтому у них идентичный синтаксис. В результате существующие CSS можно использовать в качестве LESS кода. Новые версии Sass также включают CSS-подобный синтаксис, который называется SCSS (Sassy CSS).

## 3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Проектирование является важным этапом разработки программного обеспечения. Любой проект, связанный с созданием программного продукта, требует предварительного проектирования, построения структуры и планирования сроков разработки. Лишь после предварительного утверждения плана разработки и выбора используемых технологий и структуры программного продукта начинается его реализация.

3.1 Анализ требований

В разрабатываемом программном продукте будет одна роль: пользователь. Пользователь – это человек, который зашел на сайт и зарегистрировался в системе.

Ниже представлены основные функции пользователя:

– вход в систему с помощью Facebook;

– управление наборами упражнений;

– просмотр таблицы результатов пользователей;

– сохранение информации о выполненных подходах;

– просмотр статистики выполненных упражнений;

Также надо отметить, что функция управления наборами упражнений включает в себя некоторых набор дополнительных функций. Пользователь может добавлять упражнения в список своих тренировок, устанавливать их сложность и удалять приложения из списка. Также функции сохранения информации о выполненных подходах и просмотр статистики включают в себя соответственно функции публикации сохраненной информации в социальные сети и публикации статистики в социальные сети.

Доступ пользователя к этим функциям должен быть простым и удобным. В основе реализации графического интерфейса и элементов управления должны лежать современные методики и подходы, использующиеся в разработке мобильных приложений. Интерфейс должен быть простым и не перегружать небольшой экран смартфона. Пользователь должен интуитивно понимать, какая кнопка какую функцию выполняет и куда его приведет то или иное действие, сделанное им на экране устройства.

На рисунке 3.1 изображен сценарий использования приложения пользователем.

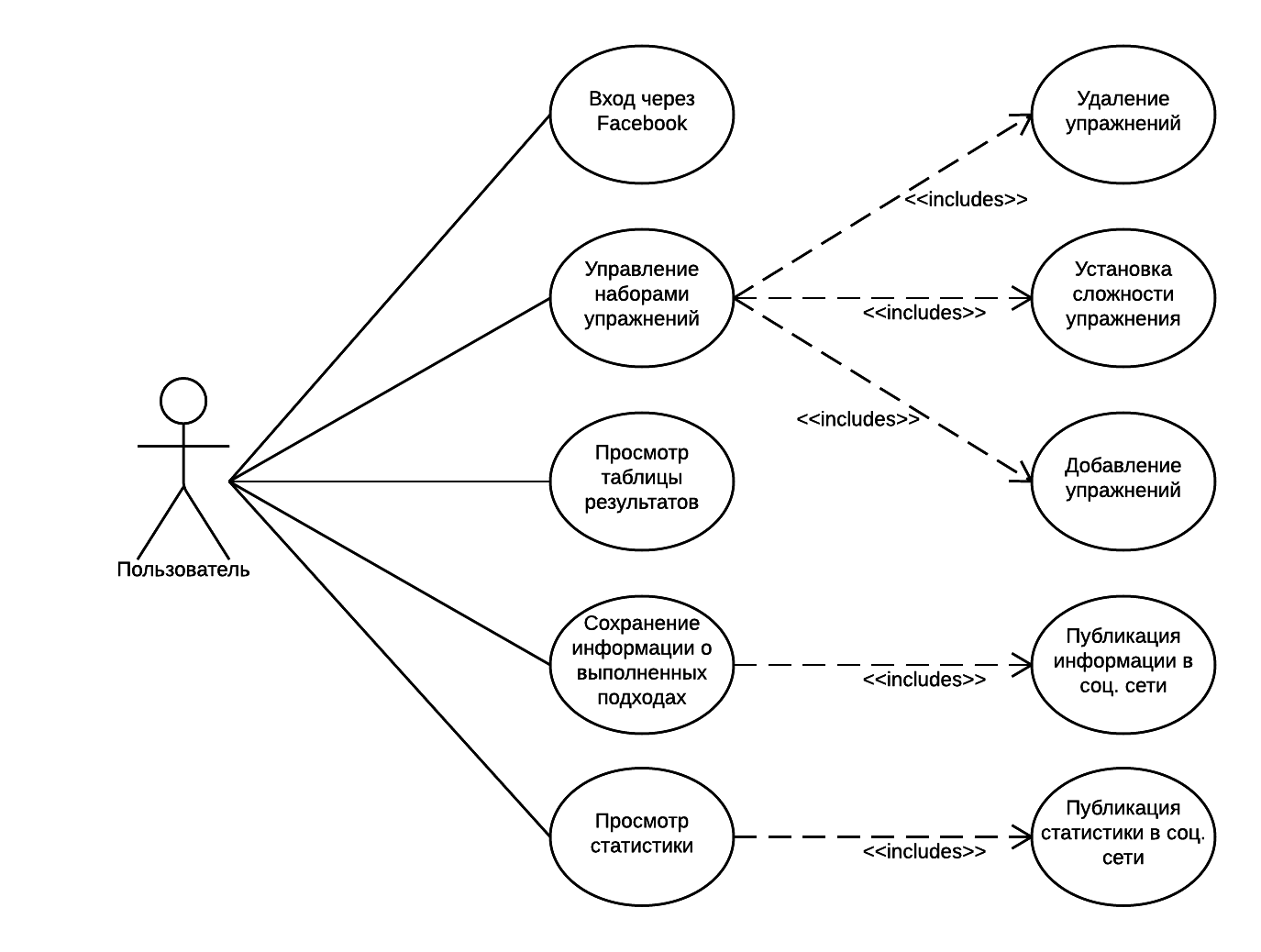


Рисунок 3.1 – Сценарий использования приложения для пользователя

Разработка дипломного проекта предполагает реализацию всего выше описанного функционала.

3.2 Разработка архитектуры программного комплекса

Разрабатываемое приложение построено с использованием трехуровневой архитектуры. Концептуальная схема архитектуры системы отображена на рисунке 3.2:

Все уровни приложения взаимодействуют друг с другом посредством четко определенных интерфейсов и классов. Это помогает создать гибкую систему, позволяющую менять реализацию на уровнях без изменений, связанных с этим уровнем других уровней. Более низкий слой при данном проектировании не может напрямую обращаться к более высокому слою.

*Уровень представления*

*Уровень бизнес-логики*

*Удаленный сервер*

*Уровень доступа к данным*

Рисунок 3.1 – Концептуальная схема архитектуры приложения

Уровень представления. Взаимодействие пользователя с системой происходит непосредственно через данный уровень. Уровень представления содержит пользовательский интерфейс, состоящий из графических элементов, элементов управления и форм, отражающих и изменяющих состояние бизнес-модели приложения посредством взаимодействия с уровнем бизнес-логики.

Уровень бизнес-логики. На данном уровне реализуются бизнес-сущности системы, а также методы и классы для работы с ними. Бизнес-сущности описывают состояния объектов. Для изменения состояния бизнес-модели реализуется набор методов и классов, которые оперируют бизнес-сущностями.

Уровень доступа к данным. Задача данного уровня – инкапсуляция работы с базой данных. На данном уровне описываются сущности БД, которые непосредственно проецируются на ее структуру. Для обеспечения сохранения и получения состояния бизнес-модели реализуется набор классов, через которые производится взаимодействие с БД.

В нашем проекте каждому из этих трех слоев соответствует собственный модуль:

* Web (слой представления);
* MakeHappy.WebApi (слой связи между логикой и представлением, относится к слою представления);
* MakeHappy.Logic (слой логики);
* MakeHappy.DataAccess (слой доступа к данным).

Вся работа с базой данных в нашем случае содержится в модуле MakeHappy.DataAccess.

Также в проекте присутствует два модуля, которые не относятся ни к одному из трех слоев и сами по себе являются независимыми:

* MakeHappy.Common (логирование, helper-классы, helper-объекты);
* MakeHappy.Dtos (сущности приложения);

В описанной модели трехуровневой архитектуры запрещены обращения от низ лежащих уровней к верхним. Обращения с верхних уровней допустимо только к соседним: так, например, запрещены всяческие обращения к уровню доступа к данным с уровня представления.

3.2.1 Проектирование структуры БД

Создание базы данных *MSSQL* и ее структуры происходит с помощью специализированных скриптов при разворачивании приложения.

Первым делом создается именованная база данных для нашего приложения:

CREATE DATABASE CityGame;

Затем по очереди в созданной базе данных выполняются команды создания таблиц, а после для созданных таблиц выполняются команды создания ключей. Пример:

create table Authentication (

Id bigint not null,

ProfileId bigint not null,

Username nvarchar(50) not null,

Password nvarchar(256) not null

)

go

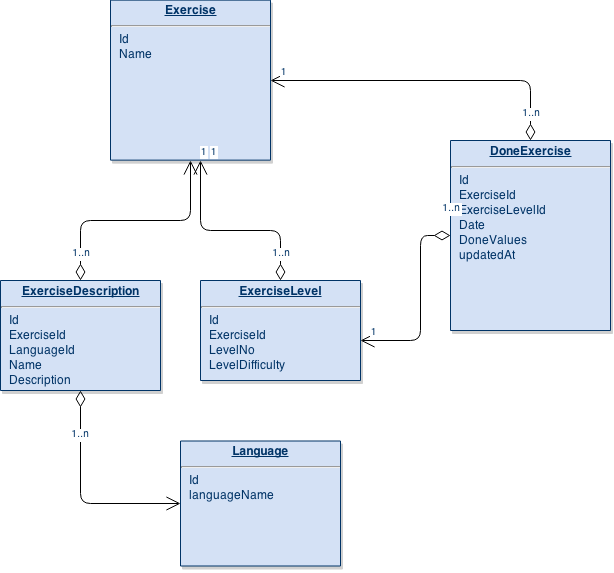


Рисунок 3.2 – Укрупненная схема БД приложения

На рисунке 3.2 изображена взаимосвязь между таблицами базы данных, являющимися хранилищем для сущностей приложения. Как видно из рисунка, архитектура базы данных достаточно простая, состоящая из пяти таблиц. Таблица Exercise содержит в себе уникальный идентификатор и названия вида упражнения. Эта таблица служит связующей между другими таблицами. Таблица Language также содержит уникальный идентификатор и имя языка, и связывает текстовые ресурсы для упражнения с языком, для которого эти тексты актуальны. Это нужно для того, чтобы пользователи приложения, для которых в системе установлен различный язык, могли с удобством пользоваться приложением. Таблица ExerciseDescription содержит в себе связь между упражнением, языком и для определенного упражнения и языка хранит соответствующее имя и описание приложения.

Таблица ExerciseLevel описывает для каждого вида упражнения уровни сложности. А таблица DoneExercise хранит в себе информацию о выполненных пользователем уровнях.

При реализации приложения, не планируется использовать какие-либо ORM фреймворки. Для получение данных из базы данных и их сохранения в базу будет использоваться подход написания SQL запросов к базе данных.

3.2.2 Проектирование уровня доступа к данным

Задача уровня доступа к данным (DAL) – обеспечить высокоуровневый прозрачный и унифицированный интерфейс для взаимодействия с БД. В данном дипломном проекте этот архитектурный слой реализуется при помощи достаточно простого подхода: классов-сущностей и объекта доступа к данным.

Андроид позволяет работать с базой данных посредством создания объекта класса SqliteHelper. Этот класс инкапсулирует логику создания базы данных и получения ссылки на объект базы данных, для выполнения запросов. При реализации приложения, будет использован паттерн Singleton и объект для работы с базой данных будет создан однажды и будет един для приложения. Это позволит не плодить одинаковые сущности, реализующий доступ к данным и избежать рассинхронизации выполнения запросов к базе данных.

Объекты классов-сущностей являются простыми объектами-контейнерами, реализующим минимум логики и хранящие в своих полях соответствующие поля таблиц баз данных. Оперирование объектами классов-сущностей происходит на уровне бизнес логики и представления.

3.2.3 Уровень бизнес-логики

При проектировании связи бизнес-логики и отображения был использован шаблон проектирования CQRS. При использовании данного шаблона, связь между данными слоями представления и логики возможна только с использованием команд (Command) и запросов (Query). В отношении базы данных команда может выполнять запрос только изменения (добавление, сохранение, удаление), а запрос может выполнять только запрос выборки данных.

Классы команды и запроса не должны содержать в себе логики. Они должны содержать в себе только данные, необходимые для выполнения команды, а также наследоваться от базового класса команды или запроса. Базовый класс имеет следующий вид:

namespace MakeHappy.Logic.Queries

{

/// <summary>

/// Command could only read data from DB

/// </summary>

public abstract class QueryBase

{

/// <summary>

/// Query identifier

/// </summary>

public abstract Guid QueryId { get; }

}

}

Класс готового запроса будет выглядеть следующим образом:

namespace MakeHappy.Logic.Queries

{

public class GetLevelQuery : QueryBase

{

public static Guid Id = new Guid("DC3C0BBC-5951-4611-8663-411E3C3183EE");

public override Guid QueryId

{

get { return Id; }

}

public int LevelId { get; set; }

public int UserId { get; set; }

}

}

В слое бизнес-логики каждой команде/запросу ставится в соответствие ее/его обработчик, базовый класс которого имеет следующий вид:

namespace MakeHappy.Logic.Blos

{

/// <summary>

/// Base business logic

/// </summary>

public abstract class BloBase

{

/// <summary>

/// Command registrator interface

/// </summary>

protected ICommandQueryRegistrator CommandQueryRegistrator { get; private set; }

protected ValidationManager ValidationManager { get; set; }

protected BloBase(ValidationManager validationManager, ICommandQueryRegistrator commandQueryRegistrator)

{

ValidationManager = validationManager;

CommandQueryRegistrator = commandQueryRegistrator;

RegisterCommandsAndQueries();

}

/// <summary>

/// Validate entity dto

/// </summary>

/// <param name="entityDto">entity Dto</param>

/// <returns></returns>

protected void Validate(DtoCommandBase entityDto)

{

var errors = ValidationManager.Validate(entityDto);

if (errors.Count != 0)

throw new ValidationException(errors);

}

protected abstract void RegisterCommandsAndQueries();

}

}

Для функционирования данной связи все обработчики необходимо зарегистрировать в памяти приложения. Для этого используется интерфейс ICommandQueryRegistrator:

namespace MakeHappy.Logic

{

/// <summary>

/// Interface for command registration

/// </summary>

public interface ICommandQueryRegistrator

{

/// <summary>

/// Register command

/// </summary>

/// <param name="commandId">Command identifier</param>

/// <param name="commandHandler">Command execution handler</param>

void RegisterCommand(Guid commandId, Func<CommandBase, ExecutionResult> commandHandler);

/// <summary>

/// Register query

/// </summary>

/// <param name="queryId">Query identifier</param>

/// <param name="queryHandler">Query execution handler</param>

void RegisterQuery(Guid queryId, Func<QueryBase, ExecutionResult> queryHandler);

}

}

Для вызова команды/запроса из слоя отображения, необходимо передать нужную команду/запрос на выполнение объекту-диспетчеру, который сопоставит ей/ему ее/его обработчик, выполнит его и вернет результат. Базовый интерфейс диспетчера имеет следующий вид:

namespace MakeHappy.Logic

{

/// <summary>

/// Interface for command dispatchering

/// </summary>

public interface ICommandQueryDispatcher

{

/// <summary>

/// Eхeсute command

/// </summary>

/// <param name="command">Command to execute</param>

/// <returns>Command execution result</returns>

ExecutionResult ExecuteCommand(CommandBase command);

/// <summary>

/// Eхeсute query

/// </summary>

/// <param name="query">Query to execute</param>

/// <returns>Query execution result</returns>

ExecutionResult ExecuteQuery(QueryBase query);

}

}

Реализацию диспетчера можно посмотреть в разделе «Приложения».

Объект-диспетчер и объект регистрации являются singleton-объектами: создаются только один раз и впоследствии везде используется ссылка на ранее созданный объект. Это реализовано с помощью IoC .NET библиотеки Ninject. Пример файла конфигурации:

namespace MakeHappy.Logic.IoC

{

public class LogicModule : NinjectModule

{

public override void Load()

{

Bind<CommandQueryDispatcher>()

.To<CommandQueryDispatcher>()

.InSingletonScope();

Bind<ICommandQueryDispatcher, ICommandQueryRegistrator>()

.ToMethod(impl => impl.Kernel.Get<CommandQueryDispatcher>());

Bind<ValidationManager>()

.To<ValidationManager>()

.InSingletonScope();

Bind<ProfileBlo>()

.ToSelf()

.InSingletonScope()

.InstantlyCreate();

Bind<AuthBlo>()

.ToSelf()

.InSingletonScope()

.InstantlyCreate();

Bind<LevelBlo>()

.ToSelf()

.InSingletonScope()

.InstantlyCreate();

Bind<RouteBlo>()

.ToSelf()

.InSingletonScope()

.InstantlyCreate();

Bind<ShowplaceBlo>()

.ToSelf()

.InSingletonScope()

.InstantlyCreate();

}

}

}

3.2.3 Уровень представления

Уровень представления взаимодействует с пользователем, посредством отображение графического интерфейса и обработку пользовательских действий в приложении. Уровень представления взаимодействует с уровнем бизнес-логики посредством интерфейса классов бизнес логики, отправляя на обработку набор данных, полученных пользователем и отображая результаты обработки этих данных.

## 4 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

4.1 Реализация базы данных

Рассмотрим на физическом уровне таблицы, являющиеся хранилищами для основных бизнес-сущностей приложения, таких как:

– Exercise;

– Language;

– ExerciseDescription;

– ExerciseLevel;

– DoneExercise;

Рассмотрим описание данных таблицы Exercise на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS ` Exercise ` (

`Id` int NOT NULL auto\_increment,

`Name` varchar(128) NOT NULL,

PRIMARY KEY (`Id`))

Краткое описание полей таблицы Exercise:

– Id – первичный ключ таблицы;

– Name – название упражнения в системе. Это системное название-идентификатор, позволяющее приложению находить информацию о упражнении.

Рассмотрим описание данных таблицы Language на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS ` Language ` (

`Id` int NOT NULL auto\_increment,

`Name` varchar(128) NOT NULL,

PRIMARY KEY (`Id`))

Данная таблица хранит доступные языки в приложении. Краткое описание полей таблицы Language приведено ниже:

– Id – первичный ключ таблицы;

– Name – название языка. Используется международная форма, такая как ru, en и т.д. Приложение на основе данных об установленном в системе языке, выбирает его из таблицы и для этого языка получает данные описания упражнений из таблицы ExerciseDescription.

Рассмотрим описание данных таблицы ExerciseDescription. Данная таблица хранит данные описания упражнений из таблицы на каком-то конкретном языке.

Ниже показана структура данной таблицы ExerciseDescription на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS ` ExerciseDescription ` (

`Id` int(11) NOT NULL auto\_increment,

`Name` varchar(128) NOT NULL,

`Description` varchar(2048) NOT NULL,

PRIMARY KEY (`Id`),

KEY `ExerciseId` (`ExerciseId `),

KEY `LanguageId` (`LanguageId`)

)

Подробная информация о полях:

– Id – первичный ключ таблицы;

– Name – название упражнения на конкретном языке;

– Description – описание упражнения на конкретном языке;

– ExerciseId – индекс, являющийся ссылкой на упражнение в таблице Exercise.

– LanguageId – индекс, являющийся ссылкой на язык в таблице Language.

Рассмотрим описание данных таблицы ExerciseLevel. Данная таблица описывает уровни сложности и подходы к выполнению упражнений на каждом уровне.

Ниже показана структура таблицы ExerciseLevel на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS ` ExerciseLevel ` (

`Id` int(11) NOT NULL auto\_increment,

`LevelNo` int(11) NOT NULL,

`LevelDifficulty` varchar(128) NOT NULL,

PRIMARY KEY (`Id `),

KEY ` ExerciseId ` (`ExerciseId `)

)

Подробная информация о полях:

– Id – первичный ключ таблицы;

– LevelNo – номер уровня сложности упражнения;

– LevelDifficulty – поле, описывающее подходы текущего уровня. Является паттерном, принятым в приложении. К примеру, если на текущем сложности уровня 3 подхода по 5 раз, будет выглядеть как 5:5:5;

– ExerciseId – индекс, являющийся ссылкой на упражнение в таблице Exercise.

Рассмотрим описание данных таблицы DoneExercise. Данная таблица описывает уровни сложности и подходы к выполнению упражнений на каждом уровне.

Ниже показана структура данной таблицы DoneExercise на DDL-языке SQL:

CREATE TABLE IF NOT EXISTS ` DoneExercise ` (

`Id` int(11) NOT NULL auto\_increment,

`Date` int(11) NOT NULL,

`DoneValues` varchar(128) NOT NULL,

`updatedAt` int(11) NOT NULL,

PRIMARY KEY (`Id `),

KEY ` ExerciseId ` (`ExerciseId `)

KEY ` ExerciseLevelId ` (`ExerciseLevelId `)

)

– Id – первичный ключ таблицы;

– Date – дата выполнения подходов уровня;

– DoneValues – поле, описывающее выполнения подходов для текущего уровня. Является паттерном, принятым в приложении. К примеру, если на текущем сложности уровня 3 подхода и пользователь выполнил первый и третий, будет выглядеть как 1:0:1;

– updatedAt –дата, когда информация об подходе была успешно отправлена на сервер. Для строк, которые не были синхронизированы с сервером, содержит 0;

– ExerciseId – индекс, являющийся ссылкой на упражнение в таблице Exercise.

– ExerciseLevelId – индекс, являющийся ссылкой на упражнение в таблице ExerciseLevel.

4.2 Реализация уровня доступа к данным

Как уже говорилось ранее в данном дипломном проекте уровень, именуемый слоем доступа к данным реализуется при помощи класса синглтона работы с объектом базы данных и простых классов-сущностей, которые хранят в себе данные таблиц.

Рассмотрим класс OpenHelper, реализующий доступ к базе данных.

Данный класс является реализацией стандартного Андроид класса SQLiteOpenHelper и содержит в себе методы создания, обновления базы данных, а также доступ к объекту базы данных, через готорых будут выполняться запросы к базе данных.

Ссылку на класс работы с базой данных OpenHelper, содержит в себе класс-синглтон MyCoachDB. Этот класс имплементирует в себе вызовы к базе данных, такие как получение и сохранение объектов в базу данных, их обновление и удаление.

Классы-сущности реализуют в себе поля, соответствующие полям таблиц и доступ к этим полям.

4.3 Реализация уровня бизнес-логики

Основной уровень приложения, представляющий бизнес-логику, достаточно объемен и его рассмотрение будет представлено поверхностно, описывая основные подходы к реализации того или иного функционала, используемого в приложении.

В приложении используется набор реализованных классов для работы с АПИ удаленного сервера. Само АПИ представляет из себя REST API с использованием протокола https. Приложение умеет делать запросы на сервер, используя методы АПИ и утвержденные форматы отправки данных, а также обрабатывать ответы сервера, и присланные данные, грамотно их приводя к объектам и сохраняя в базе данных, если это нужно.

За работу с АПИ приложения отвечает класс API. Этот класс содержит ряд методов для выполнения запросов к удаленному серверу. Рассмотрим метод registerUser:

**public** StringResponce registerNewUser(String email, String device\_id) {

StringResponce resp = **null**;

String token = **null**;

JSONObject j = **new** JSONParser().registerNewUser(email, device\_id);

**try** {

**int** code = j.getInt("code");

JSONObject data = j.getJSONObject("data");

token = data.getString("token");

resp = **new** StringResponce(token, code);

} **catch** (JSONException e) {

return ERROR\_RESPONSE;

e.printStackTrace();

}

**return** resp;

}

Данный метод получает на вход пользовательский ящик электронной почты и уникальный идентификатор устройства. Затем он упаковывает эти данные согласно принятому формату АПИ, и вызывает метод, непосредственно отправляющий данные на сервер с помощью класса работы с Https протоколом. Это метод класса JSONParser c названием registerNewUser. Ниже представлен его код:

**public** JSONObject registerNewUser(String email, String device\_id) {

JSONObject out = **null**;

httpClient = getNewHttpClient();

Constants constants = **new** Constants();

URI url = **new** URI(constants.getRegisterUrl());

HttpPost httpPost = **new** HttpPost(url);

JSONObject jsonP = **new** JSONObject();

jsonP.put("email", email);

jsonP.put("deviceId", device\_id);

jsonP.put("os", constants.*os*);

jsonP.put("appId", MyCoachApp.*type*);

String toPush = "data=" + jsonP.toString();

ByteArrayEntity baEntity = **new** ByteArrayEntity(

toPush.getBytes("UTF8"));

httpPost.setEntity(baEntity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

*is* = httpEntity.getContent();

out = getJSON(*is*);

out.put("code", RESP\_CODE);

**return** out;

}

Метод подготавливает объект работы с протоколом http, упаковывает данные и отправляет их на сервер. Ждет ответа с сервера и отправляет ответ в качестве возвращаемого параметра. А тот метод, который получил этот ответ уже знает, как его обработать. Таким образом мы разделяем логику работы, где каждый класс выполняет определенный объем работ и не делает больше, чем нужно.

Классы и методы обработки данных в приложении можно увидеть на диаграмме классов, которая представлена с графическими материалами дипломного проекта. В качестве примера реализации бизнес логики приведем пример реализации установки напоминаний об выполнении упражнений. Данная логика вплотную взаимодействует с уровнем доступа к данным приложения и с АПИ системы Андроид. Рассмотрим ее подробнее.

Логика установки напоминания срыта в классе AlarmService, который имеет метод setNotification:

**public** **static** **void** setNotification(Context context,

**long** waitTimeInMiliseconds) {

Intent i = **new** Intent(context, OnAlarmReceiver.**class**);

PendingIntent pi = PendingIntent.*getBroadcast*(context, 0, i, 0);

AlarmManager alarmManager = (AlarmManager) context

.getSystemService(Context.*ALARM\_SERVICE*);

alarmManager.set(AlarmManager.*RTC\_WAKEUP*, System.*currentTimeMillis*()

+ waitTimeInMiliseconds, pi);

}

Этот метод получает информацию, когда должна показаться нотификация пользователя о том, что нужно выполнять упражнение и с помощью системных средств АПИ Андроид устанавливает бродкаст, который будет передам системой в метод doReminderWork класса AlarmService, как только время, установленное в этом ресивере наступит. Метод doReminderWork составной и состоит из вызова методов подготовки внешнего вида уведомления, подготовки списка упражнений, доступных для выполнения и собственно показ уведомления.

Таким образом бизнес-логика приложения, в плотную взаимодействуя с уровнем доступа к данным и вызываемая через интерфейсы классов бизнес-логики из уровня представления, обеспечивает работу приложения и всех его функций.

4.4 Реализация уровня представления

Уровень представления реализован при помощи стандартных средств разработки под операционную систему Андроид. Он представляет из себя разметку экранов приложения, описанную с помощью языка xml и реализации классов активностей, фрагментов и вью, которые отображают эту разметку и работают с объектами графического интерфейса, такими, как кнопки, текстовые поля и другие.

Разметка экранов и элементов управления описывается файлами xml с помощью атрибутов Андроид, задающих размеры, положение элементов, цвета, тип и другие особенности. После Андроид система на лету создает из этой разметки классы объектов управления и собственно рисует изображение на экране.

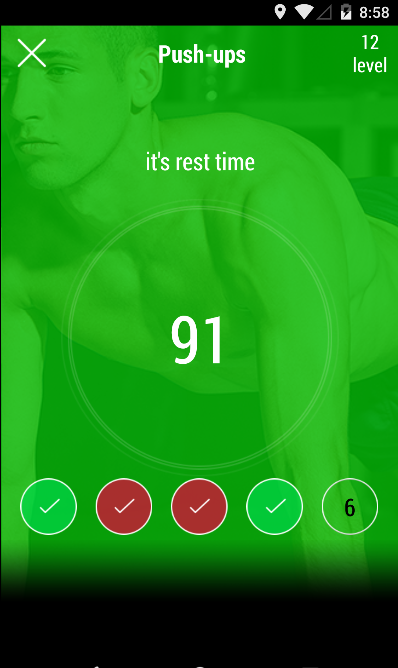


Рисунок 4.1 – Экран выполнения подходов упражнения

На рисунке 4.1 изображен экран выполнения подходов упражнения. Как видно на этом экране есть некоторые интерактивные элементы, такие как таймер с анимацией, анимированные значки выполнения подхода, текстовые поля информации об уровне упражнения, виде упражнения и другие элементы. Все эти элементы реализуются на уровне представления. К примеру, для таймера и значков были разработаны свои классы, объекты которых умеют рисовать такие элементы на экране, грамотно обрабатывать их анимацию и взаимодействие на разметке с другими объектами.

Рассматривая более подробно этот экран хочется отметить, что на нем задействованы почти все основные функции приложения. При старте этого экрана, активити TimerWorkoutActivity запрашивает у уровня бизнес логики информацию по текущему уровню сложности выбранного упражнения. Уровень бизнес логики для этого спрашивает базу данных, которая управляется уровнем доступа к данным. Получая данные с уровня доступа к данным, классы бизнес логики предоставляют ее в удобном виде уровню представления, а уровень представления создает по этой информации вид экрана. Также, стоит отметить, что в программе предусмотрена функция сохранения незаконченного уровня сложности, когда пользователь может завершить подходы, если по каким-то причинам он их не завершил.

После выполнения подходов, приложение сохраняет статистику в базу данных, а затем делает попытку отправить информацию для синхронизации с сервером. Если попытка была успешной – информация о подходе помечается соответствующим образом, иначе она будет обновлена при следующем подключении к серверу.

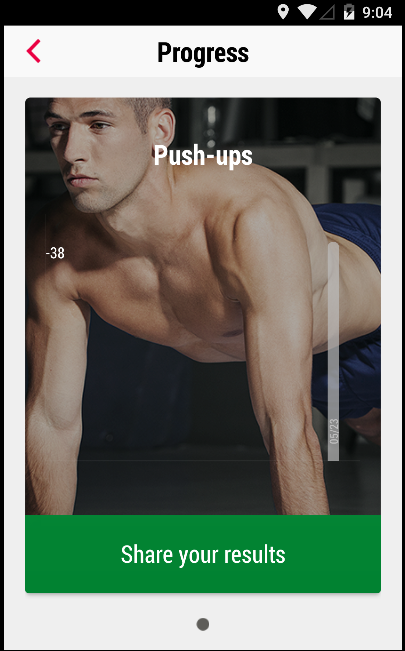


Рисунок 4.2 – Экран статистики выполнения упражнений

Как было описано выше, в приложении есть возможность просматривать статистику по выполненным упражнениям пользователем. Пример экрана статистики представлен на рисунке 4.2. Для этого был разработан экран просмотра статистики, и собственные элементы статистики, такие как рисование графика в виде столбцов с уровнем выполненных упражнений. На этом экране реализована возможность вызова меню, позволяющего поделиться результатами в виде картинки графика с друзьями в социальных сетях. Внешний вид этой панели приведен на рисунке 4.3

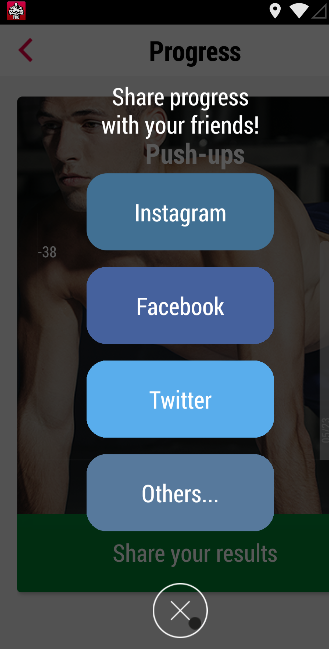


Рисунок 4.3 – Панель отправки статистики в социальные сети

На рисунке 4.3 представлен пример еще одного экрана приложения – экрана отправки статистики в социальные сети. С помощью этого экрана пользователь может отправить картинку на стену в социальной сети Facebook, Twitter либо в фото-сеть Instagram. Логика, реализуемая в классах уровня представления, получает на обработку нажатия соответствующих кнопок и с помощью классов бизнес-логики выполняет желаемые действия. Получая от уровня бизнес-логики результаты выполняемых операций, уровень представления может сообщить об удачной отправке картинки либо об ошибке.

Стоит отметить, что реализация взаимодействия с социальными сетями в приложении сделана посредством подключения предоставляемых этими соц. сетями SDK, внутри которых есть АПИ для вызова методов авторизации и публикации. Благодаря стандартным механизмам взаимодействия Андроид приложений, приложение может отправить запрос на выполнение действий другому, и в случае успеха, получит ответ от запрашиваемого приложения.

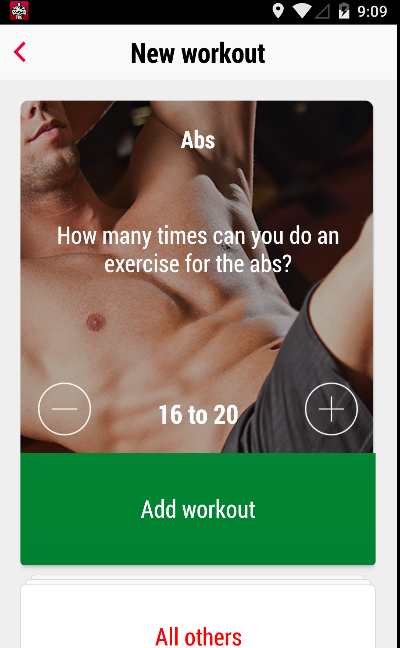


Рисунок 4.4 – Настройка сложности упражнения

На рисунке 4.4 изображен экран настройки упражнений. На этом экране пользователю предлагается установить желаемую сложность занятий.

В общем экраны приложения, на которых стандартные элементы Андроид в сочетании с специально разработанными элементами графического интерфейса представляют данные и обрабатывают действия пользователя над этими данными, с помощью классов и методов этих классов образуют из себя целостный уровень представления приложения, доступный для расширения и изменения, без затрагивания процессов в уровне бизнес логики и уровне доступа к данным.

## 5 ТЕСТИРОВАНИЕ И ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ

5.1 Тестирование модулей

Для тестирования функциональности использовалась .NET библиотека NUnit, которая позволяет с минимальным использованием усилий, реализовать все виды Unit-тестирования приложения.

В качестве примера рассмотрим тестирование работоспособности получения информации о пользователе.

namespace MakeHappy.Logic.Test.Blos

{

[TestFixture]

public class UserProfileBloTest

: TestFixtureBase<ProfileBlo>

{

[Test]

public void GetUserProfileTest()

{

var query = new GetUserCardQuery() { UserId = 1 };

var executionResult = CommandQueryDispatcher.ExecuteQuery(query);

Assert.IsNotNull(executionResult);

}

}

}

namespace MakeHappy.Logic.Test

{

public class TestFixtureBase<T>

where T : BloBase

{

private ISessionManager \_sessionManager;

private StandardKernel \_kernel;

protected CommandQueryDispatcher CommandQueryDispatcher;

protected T Blo;

[TestFixtureSetUp]

public virtual void TestFixtureSetUp()

{

\_kernel = new StandardKernel(new DataAccessModule(), new LogicModule());

\_sessionManager = \_kernel.Get<ISessionManager>();

CommandQueryDispatcher = \_kernel.Get<CommandQueryDispatcher>();

Blo = \_kernel.Get<T>();

}

[SetUp]

public void SetUp()

{

\_sessionManager.OpenSession();

\_sessionManager.BeginTransaction();

}

[TearDown]

public void Dispose()

{

\_sessionManager.RollbackTransaction();

\_sessionManager.CloseSession();

\_kernel.Dispose();

}

}

}

Как видно из примера, мы создаем экземпляр объекта-команды и присваиваем ему в передаваемые данные id пользователя. Следующим шагом мы выполняем команду и сравниваем полученный результат с необходимым результатом. Если результаты равны, то тест считается пройденным. При запуске любого теста мы конфигурируем нашу среду выполнения. Для этого мы, при помощи Ninject-а, создаем и инициализируем наш объект диспетчера, который впоследствии будет использоваться в самих тестах.

5.2 Бета тестирование

Перед началом общеуниверситетской городской игры программный продукт нужно протестировать на ограниченном круге пользователей, называемых соответственно альфа и бета тестировщиками. Для альфа и бета тестирования программный продукт был предложен небольшой группе студентов БГУИР, которые смогли протестировать систему и оставили свои комментарии и пожелания по модернизации программного продукта.

Во время альфа-теста проверялся только технический функционал системы. Было найдено несколько ошибок, связанных с масштабирование сайта на различных мобильных платформах. Эти ошибки были устранены перед началом второго этапа тестирования.

Бета-тест проходил уже в городских условиях. Были подготовлены 4 полноценных маршрута, включающих в себя 10 точек каждый. Команды подбирались таким образом, что задействовать как можно большее количество различных современных мобильных платформ для более успешного и детального тестирования системы. Во время бета-теста ошибок выявлено не было.

5.3 Запуск городской игры в БГУИР

Во время мероприятия «Неделя факультета», приуроченного к юбилею факультета компьютерных систем и сетей, прошла городская игра «Найди», использующая разработанную в рамках данного дипломного проекта систему. Для самой игры были разработаны 10 уникальных маршрутов для команд-участников. Во время игры система показала стабильную работу, без сбоев и ошибок. Пользователи хорошо оценили как техническую, так и визуальную составляющую системы. Так же во время работы не наблюдалось неполадок, связанных с масштабирование – интерфейс одинаково качественно отображался на различных мобильных платформах.

Запуск системы в рамках университетской игры показал, что система работает качественно, обладает необходимым быстродействием и является неотъемлемым дополнением игры для обеспечения интерактивности и поддержания темпа игры.

## 6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО КОМПЛЕКСА СОПРОВОЖДЕНИЯ ИНТЕЛЛЕКТУАЛЬНО\_СПОРТИВНОЙ ИГРЫ

**6.1 Характеристика программного продукта**

Целью дипломного проекта является разработка программного комплекса сопровождения интеллекту­ально-спортивной игры.

Разработанное программное средство является узкоспециализированным. Это обусловлено начальной задачей и требованиями игры. Программное средство предоставляет участникам доступ в систему для наиболее оперативного получения новых заданий и выполнения уже имеющихся.

В рамках дипломного проекта было получено полностью готовое программное средство, которое было успешно запущено при проведения игры в рамках университета.

С точки зрения экономической эффективности для разработчика прибыль заключается в продаже программного средства. Для расчета экономической эффективности нужно высчитать объем и трудоемкость ПО, смету затрат и итоговую цену программного средства.

**6.2** **Экономический эффект у разработчика**

Данный программный модуль относится ко 2 категории сложности, так как имеется возможность пользоваться им с любой мобильной платформы и персонального компьютера. При этом коэффициент сложности равняется 1,0.

По степени новизны ПС относится к категории А с коэффициентом новизны , так как ПО является новым и не разрабатывалось на основе нового ПК либо новой ОС.

При разработке модуля используются существующие технологии и средства разработки, которые охватывают около 40 – 45% реализуемых функций, поэтому коэффициент использования стандартных модулей принимается равным 0,7.

**6.2.1** **Определение объема и трудоемкости ПО**

Для того, чтобы рассчитать плановую смету затрат на разработку ПО, требуется определить общий объем ПС (). Общий объем программного продукта определяется исходя из количества и объема функций, реализуемых программой. Существует несколько вариантов единиц измерения общего объема ПС. В качестве единицы измерения возьмем количество строк исходного кода (Lines of Code, LOC). Объем разрабатываемых функций определяется по каталогу функций (см. таблицу 6.1).

На основе общего объема ПС рассчитывается нормативная трудоемкость ПО (). Так как общий объем ПС = 20110, категория сложности ПО – 1, нормативная трудоемкость = 464.

Таблица 6.1 – Каталог функций ПО

|  |  |  |
| --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функций (строк исходного текста) |
| 101 | Организация ввода информации | 150 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 |
| 201 | Генерация структуры базы данных | 4300 |
| 204 | Обработка набора и записей базы данных | 2670 |
| 207 | Манипулирование данными | 9550 |
| 401 | Генерация рабочих программ | 2670 |
|  | Итого | 20110 |

Тогда общая трудоемкость разработки определяется по формуле:

(6.1)

где – дополнительный коэффициент сложности; – коэффициент, учитывающий использование типовых программ и модулей; – коэффициент новизны.

Подставляя значения в 6.1, получим:

(человеко-дней)

На основе общей трудоемкости определяется плановое число разработчиков по формуле:

, (6.2)

где – численность исполнителей проекта; – общая трудоемкость разработки проекта (чел./дн.); – срок разработки проекта (лет); – эффективный фонд времени работы одного работника в течение года (дн).

Срок разработки проекта составляет половину года ( = 0,5 года). Эффективный фонд времени определяется по формуле 6.3.

, (6.3)

где – количество дней в году; – количество праздничных дней в году; – количество выходных дней в году; – количество дней отпуска.

Подставляя значения в формулу 9.3, получим:

(дней).

При решении сложных задач с длительным периодом разработки ПО трудоемкость определяется по стадиям разработки: техническое задание (ТЗ), эскизный проект (ЭП), технический проект (ТП), рабочий проект (РП), внедрение (ВН). При этом трудоемкость разработки ПО отличается в зависимости от стадий. Общий вид формулы, по которой рассчитывает трудоемкость изготовления ПС:

, (6.4)

где – трудоемкость изготовления ПО на данной стадии; – нормативная трудоемкость; – коэффициент, учитывающий сложность изготовления ПО на выбранной стадии; – удельный вес трудоемкости выбранной стадии разработки ПО в общей трудоемкости разработки ПО; – коэффициент, учитывающий степень новизны ПО на выбранной стадии.

Таблица 6.2 – Расчет общей трудоемкости разработки ПО и численности исполнителей с учетом стадий

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| 1. Коэффициенты удельных весов трудоемкости стадии разработки ПО () | 0,11 | 0,09 | 0,11 | 0,55 | 0,14 | 1,00 |
| 2. Коэффициент сложности ПО () | 1,0 | 1,0 | 1,0 | 1,0 | 1,0 |  |
| 3. Коэффициент, учитывающий использование стандартных модулей |  |  |  | 0,70 |  |  |
| 4. Коэффициент, учитывающий новизну ПО () | 1,0 | 1,0 | 1,0 | 1,0 | 1,0 |  |
| 5. Общая трудоемкость ПО (), чел./дн. | 51,04 | 41,76 | 51,04 | 178,64 | 64,96 | 387,44 |

При этом для стадии «Рабочий проект» полученное значение трудоемкости изготовления ПО требуется умножить на . Общая трудоемкость определяется как сумма трудоемкости изготовления ПО на каждой из стадий разработки. Результаты расчетов трудоемкости по стадиям сведены в таблицу 6.2.

На основе уточненной трудоемкости разработки ПО с использованием формулы 6.2 найдем общую численность разработчиков, которые требуются, чтобы вложиться в заданные сроки:

(человек).

**6.2.2 Расчет сметы затрат и цены заказного ПО**

Основная статья расходов на создание ПО является заработная плата разработчиков проекта, то есть людей, непосредственно занимающихся разработкой.

Месячная тарифная ставка каждого специалиста () определяется по формуле:

, (6.5)

где – месячная тарифная ставка первого разряда (275000 руб.); – тарифный коэффициент, соответствующий установленному тарифному разряду.

Часовая тарифная ставка :

, (6.6)

где – часовая тарифная ставка (тыс.руб); – среднемесячная норма рабочего времени в часах (составляет 170 часов).

Расчет месячных и часовых тарифных ставок сведен в таблицу 6.3.

Основная заработная плата исполнителей рассчитывается по формуле:

, (6.7)

Таблица 6.3 – Расчет месячных и почасовых тарифных ставок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Должность | Количество ставок | Тарифный разряд | Тарифный коэффициент | Месячная тарифная ставка (руб.) | Часовая тарифная ставка (руб.) |
| Ведущий инженер-программист | 1 | 15 | 3,48 | 957000 | 5629,41 |
| Инженер-программист 1-ой категории | 2 | 14 | 3,25 | 893750 | 5257,35 |

где n – количество исполнителей; – часовая тарифная ставка i-го исполнителя (ден.ед.); – количество часов работы в день, ч; – коэффициент премирования; – плановый фонд рабочего времени -го исполнителя (дн.).

(руб).

Дополнительная заработная плата () включает в себя оплаты отпусков и другие выплаты, предусмотренные законодательством. Определяется по формуле:

, (6.8)

где – норматив дополнительной заработной платы (10-20%).

(руб).

Отчисления в фонд социальной защиты определяется по формуле:

, (6.9)

где – норматив отчислений в фонд социальной защиты наделения.

Отчисления в фонд социальной защиты – 34%, отчисления в фонд социального страхования – 0.6%. Исходя из этого, получаем:

(руб).

Расходы по статье «Материалы» отражают расходы на магнитные носители, бумагу, тонер и прочие вещи, необходимые для разработки ПО. Нормы расхода материалов в суммарном выражении () определяется в расчете на 100 строк исходного кода. Сумма затрат на расходные материалы определяется по формуле:

, (6.10)

где – норма расхода материалов в расчете на 100 строк исходного кода ПО (руб); – общий объем ПО (строк исходного кода).

(руб).

Расходы по статье «Машинное время» включает оплату машинного времени, необходимого для разработки и отладки ПО. Норматив на 100 строк исходного кода () зависит от характера решаемых задачи и типа приложений. Расходы по этой статье определяются по формуле:

, (6.11)

где – цена одного машино-часа (руб); – общее время работы над проектом (часов).

(руб).

Расходы по статье «Научные командировки» () определяются по формуле:

, (6.12)

где – норматив расходов на командировки в целом по организации (%).

(руб).

Расходы по статье «Прочие затраты» () включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяется по формуле:

, (6.13)

где – норматив прочих затрат в целом по организации.

(руб).

Затраты по статье «Накладные расходы» () связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

, (6.14)

где – норматив накладных расходов в целом по организации.

(руб).

Общая сумма расходов по смете ():

(руб).

Затраты на сопровождение и адаптацию ПО ():

, (6.15)

где – норматив расходов на сопровождение (%).

(руб).

Затраты на производство и продвижение программы определяется по формуле:

(6.16)

Подставляя значения в формулу 6.16, получим:

(руб).

Прибыль от создаваемого ПО определяется по формуле:

, (6.17)

где – уровень рентабельности ПО (%).

Подставляя значения в формулу 6.17, получим:

(руб).

Прогнозируемая цена ПО без налогов () определяется по формуле:

(6.18)

Подставляя значения в формулу 6.18, получим:

(руб).

Налог на добавленную стоимость ():

, (6.19)

где – норматив НДС (%).

(руб).

Прогнозируемая отпускная цена ():

(6.20)

Подставляя значения в формулу 6.20, получим:

(руб).

Прибыль за вычетом налога на прибыль составит рублей. Данная сумма остается при реализации собственнику разработки и представляет собой экономический эффект у собственника.

Таким образом, в результате оценки экономического эффекта у разработчика были получены следующие результаты:

* себестоимость проекта составила 49,2 миллиона рублей;
* прогнозируемая отпускная цена – 59,5 миллиона рублей;
* прибыль за вычетом налога составит 6,3 миллиона рублей.

**6.3** **Вывод**

Для определения экономического эффекта от использования нового ПО у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПО (расходы на заработную плату с начислениями, затраты на расходные материалы расходы на машинное время) с расходами по соответствующим статьям базового варианта. При этом за базовый вариант следует принять аналогичное программное средство. Создание нового ПО окажется экономически целесообразным лишь в том случае, если все капитальные затраты окупятся за счет получаемой экономии в ближайшее время (до 2 лет).

Однако из-за того, что отсутствует актуальная база для сравнения, расчет экономического эффекта у пользователя невозможен.

## 7 ЭРГОНОМИЧЕСКАЯ ОЦЕНКА ПРОГРАММНОГО ПРОДУКТА ФИТНЕС ПРИЛОЖЕНИЯ ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ АНДРОИД

Целью дипломного проекта является разработка программного комплекса сопровождения интеллекту­ально-спортивной игры. Данный программный продукт обладает узкоспециализированной направленностью, обусловленной требованиями игры. С точки зрения эргономики, самое важное в программе – создать такой пользовательский интерфейс, который сделает работу эффективной и производительной, а также обеспечит удовлетворенность пользователя от работы с программой. Рассмотрим, что это означает с точки зрения концепта игры.

Разрабатываемое ПС предполагается использовать для автоматизации игрового процесса, повышения гибкости, интерактивности и привлекательности игры. Так как всё основное действие игры происходит в городских условиях в достаточно быстром темпе, пользовательский интерфейс программного средства должен быть выполнен в стиле минимализма. Это не отвлекает от основного процесса игры и помогает быстро и интуитивно взаимодействовать с ПС. С другой стороны, это помогает минимизировать количество возможных ошибок, связанных с вводом информации пользователем. А этот аспект в игре является одним из главных.

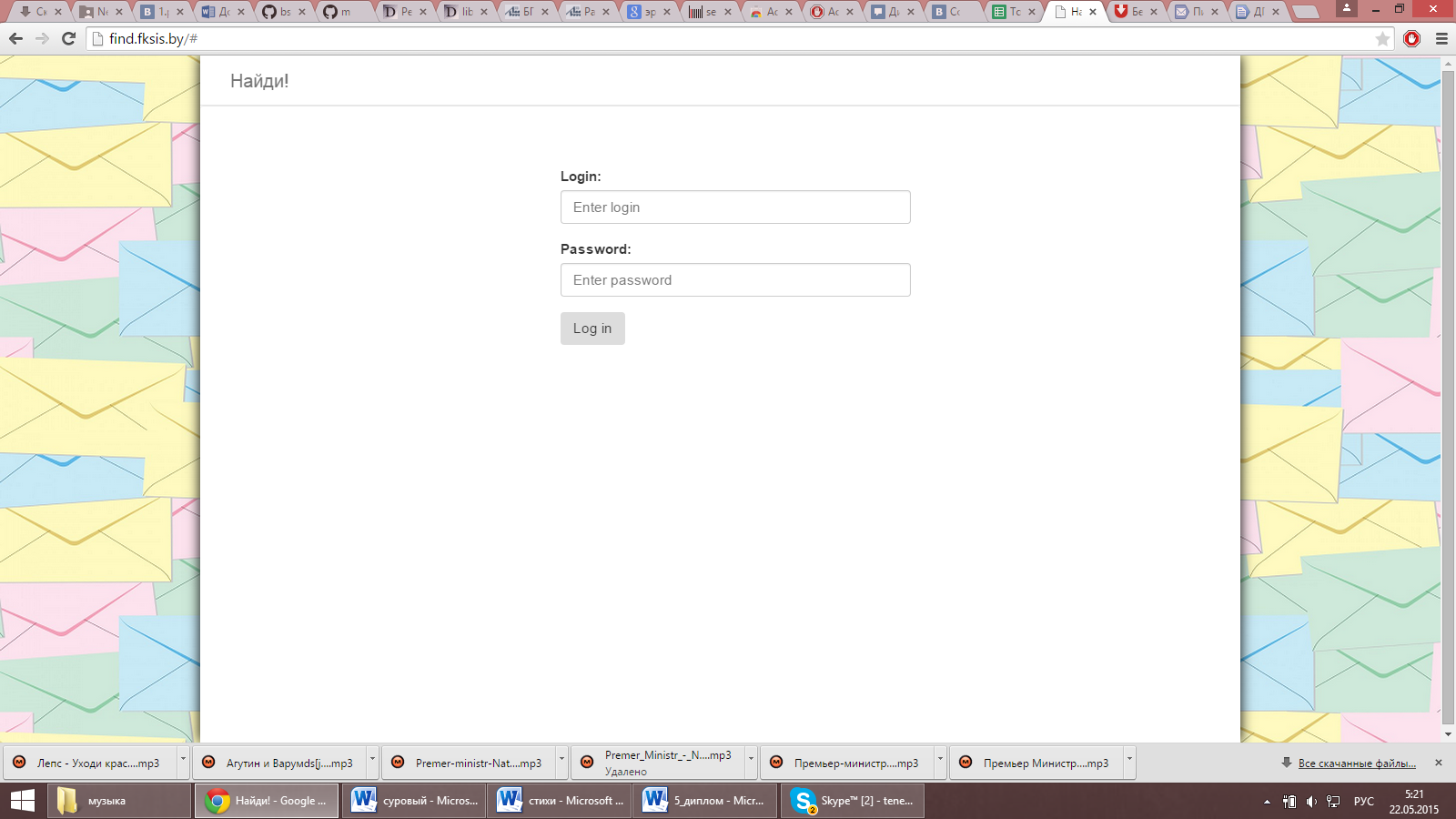


Рисунок 7.1 – Страница идентификации пользователя

Единое стилевое оформление. Это свойство пользовательского интерфейса необходимо для первоначальной адаптации пользователя при работе с программным средством. Приоритетной, в связи с начальными этапами внедрения продукта, является факультетская стилистика. Наиболее ярко это выражается при выделении элементов интерфейса. Пример – зеленая окантовка полей «Login» и «Password» на начальной странице, когда в эти поля уже введена какая-либо информация.

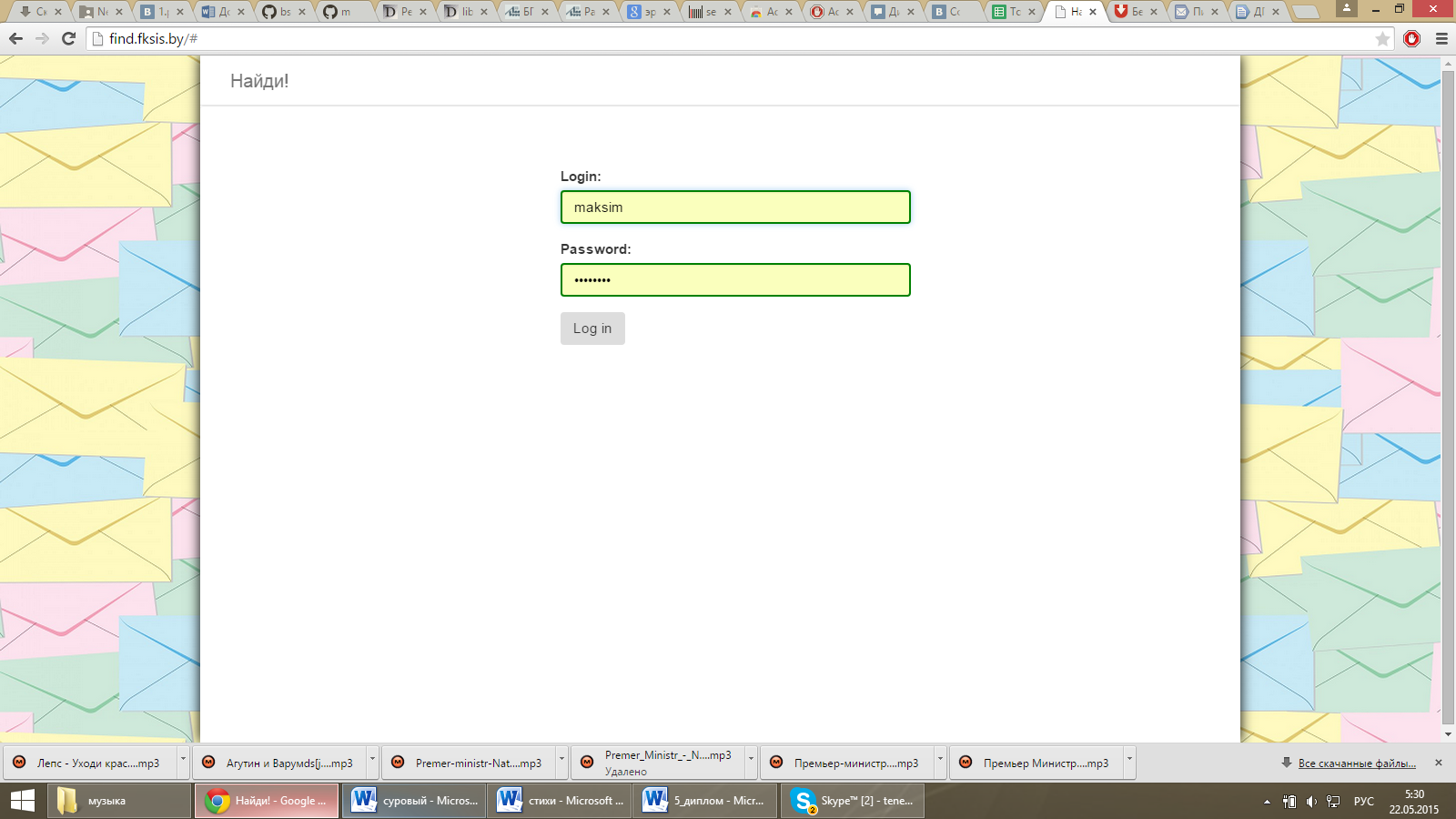


Рисунок 7.2 – Выделение элементов интерфейса при вводе данных

 Адаптивность к действиям пользователя. При не верном либо ошибочном вводе пользовательских данных программное средство сообщает о допущенной ошибке.

Зона, где была допущена ошибка, выделяется ярким цветом для привлечения внимания.

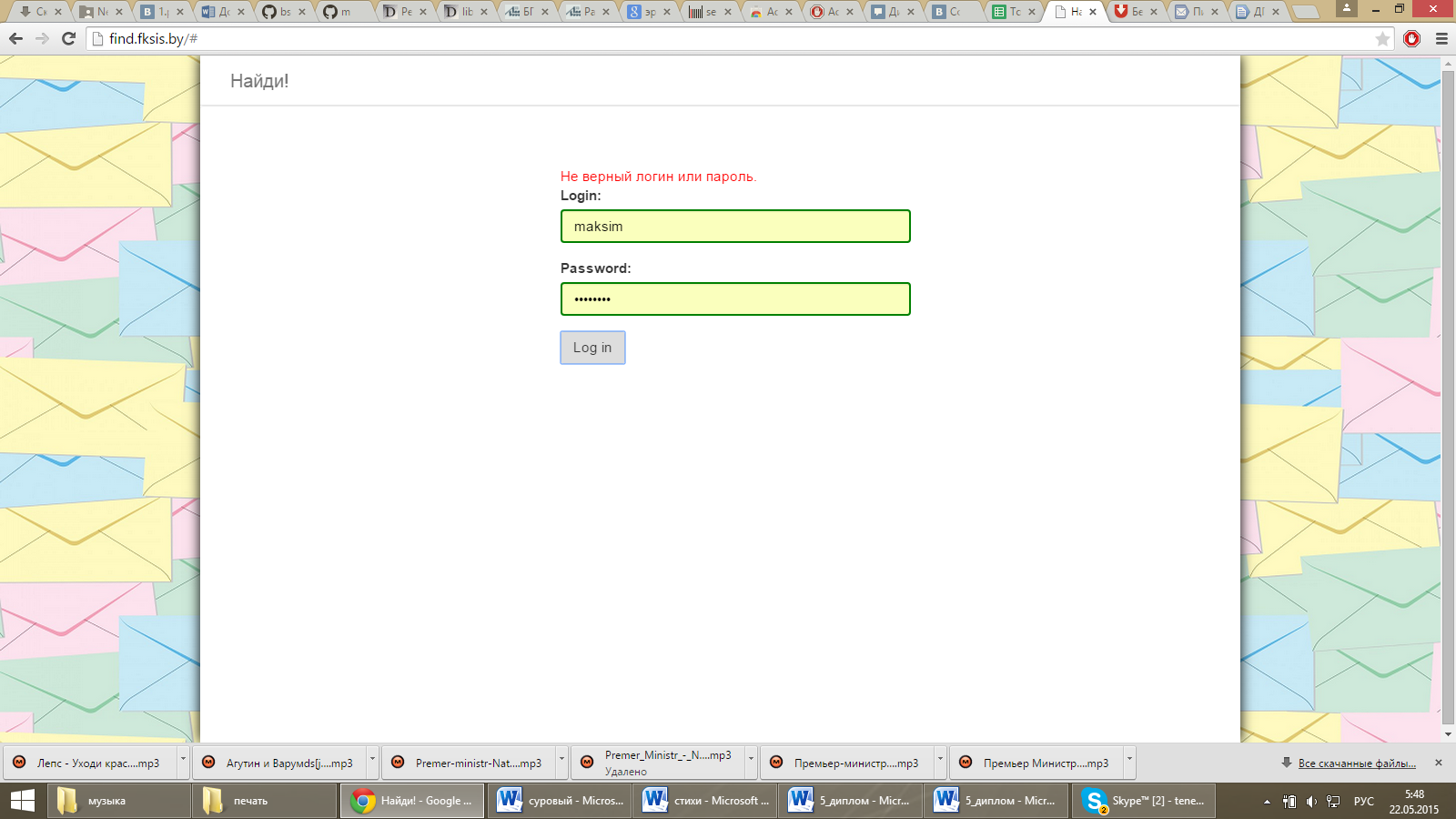


Рисунок 7.3 – Пример некорректного ввода пользовательских данных

Для положительного пользовательского опыта возле ошибочного поля размещается надпись, информирующая об ошибке, а так же возможных путях исправления.

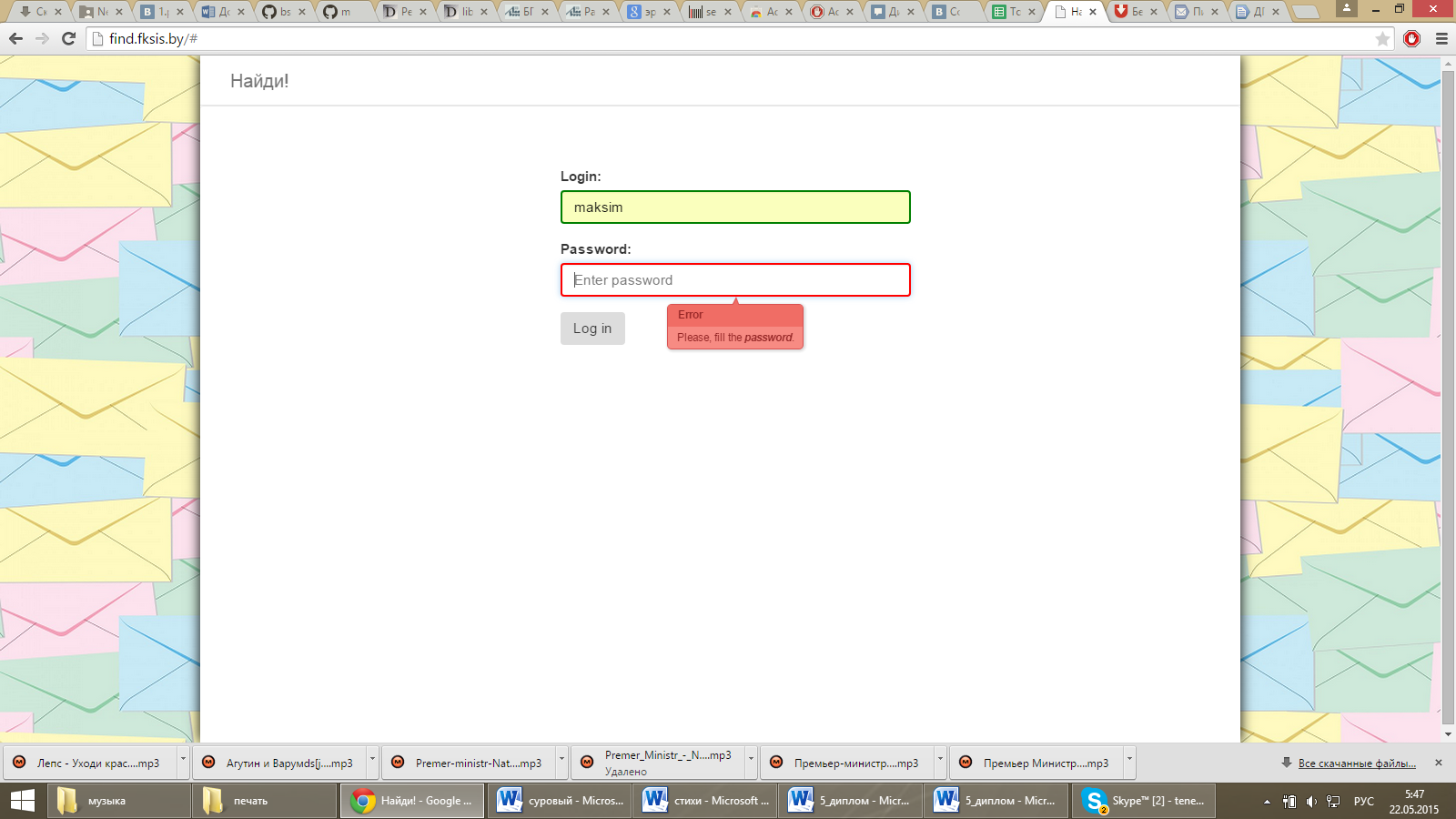


Рисунок 7.4 –Всплывающая подсказка при некорректном вводе данных

Интерфейс программного средства адаптирован к новейшим мобильным устройствам. Это свойство является одним из первоочередных для программного средства. Во время игры, которая происходит в городских условиях, наиболее быстрый и оперативный доступ в Сеть команды имеют не через персональные компьютеры, которые достаточно тяжело всё время держать при себе, а через мобильные устройства. В связи с этим, программное средство должно одинаково качественно работать на всех основных мобильных платформах.

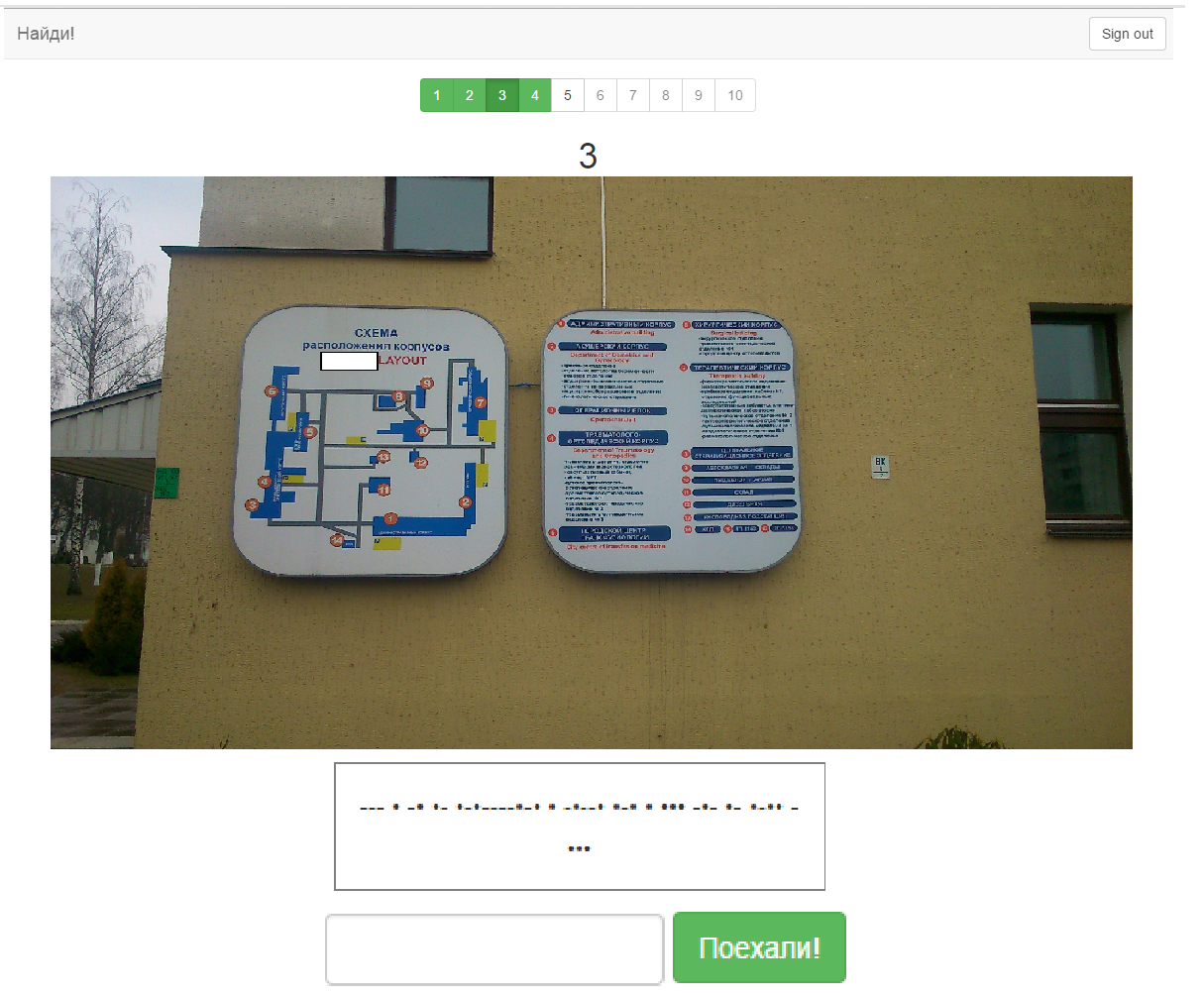


Рисунок 7.5 – Скриншот страницы, сделанный с мобильного устройства с iOS 8

В программном средстве предусмотрены дополнительные функции для наиболее полного получения информации из предоставленного условия загадки.

Когда пользователь заходит в систему через мобильное приложение, в правом верхнем углу фотографии, которая является частью загадки, появляется значок лупы. При нажатии на этот значок фотография открывается в новом окне в полном разрешении, чтобы команда смогла более детально рассмотреть фото и получить какие-либо подсказки для своего маршрута. Если итоговая информация по загадке, необходимая для ввода, слишком мала по сравнению с общим размером фотографии, вышеописанная функция помогает команде лучше понять, что является финальным ответом.

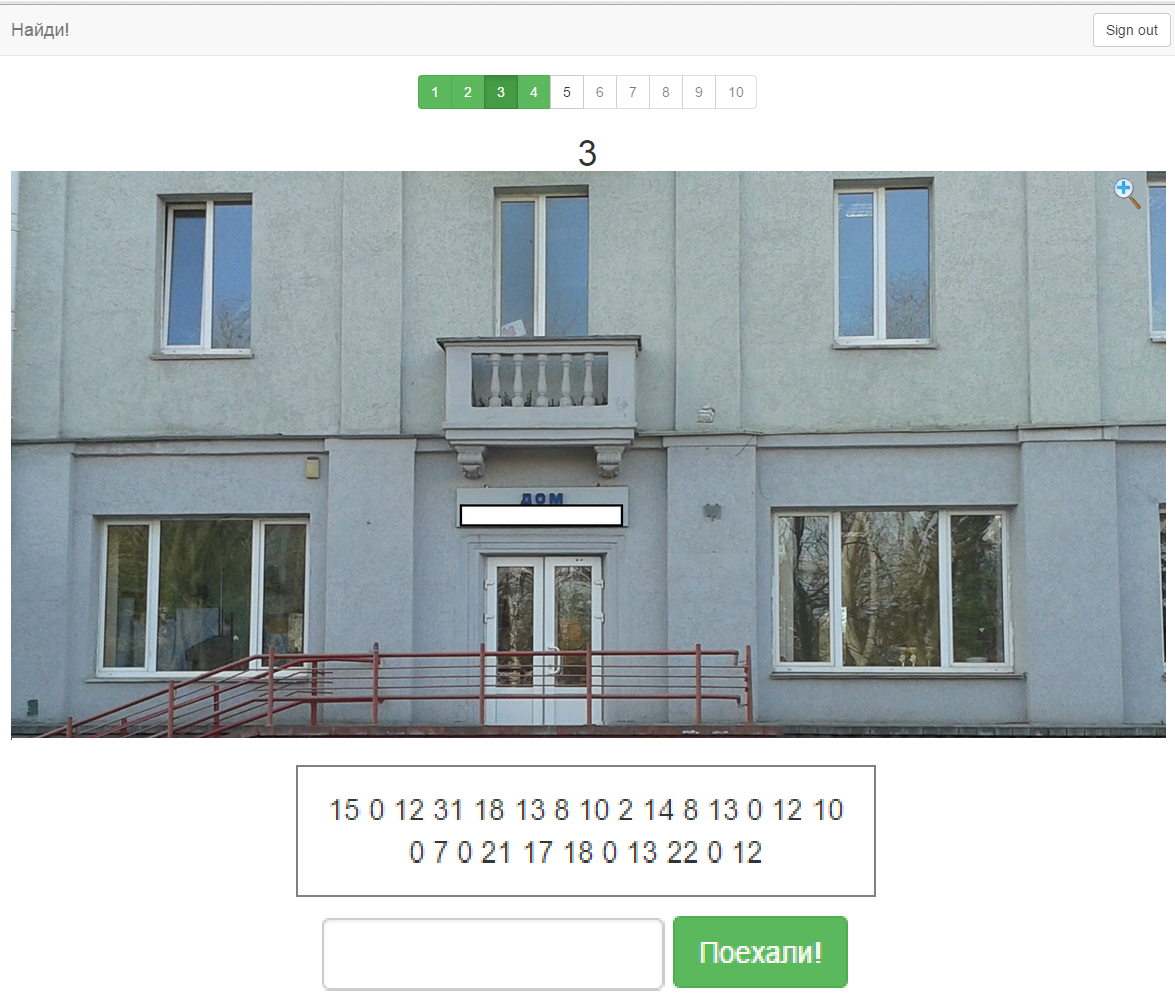


Рисунок 7.6 – Скриншот страницы с возможностью детального просмотра фотографии

Единое стилевое оформление, минимализм интерфейса, адаптивность к действиям пользователя и возможность работы программного средства на различных мобильных платформах. Все перечисленные свойства помогают системе сделать игру более гибкой, а пользователю предоставить необходимый функционал для участия в игре.

Таким образом, перечисленные выше методы позволяют в должной степени обеспечить эргономику программного продукта.

## ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта было разработан программный комплекс сопровождения интеллектуально-спортивной игры Перед разработкой программного средства были изучены теоретические основы разработки веб-приложений, существующие подходы и технологии, для эффективной разработки.

Также еще до непосредственного старта разработки приложения был произведен качественный анализ предметной области. Была продумана и описана архитектура будущего приложения. Приложение было реализовано согласно разработанной архитектуре. Также были проведены различные виды тестирования приложения, что помогло выявить ошибки реализации приложения и вовремя их исправить.

Разработанное приложение несет практическую пользу, являясь качественным комплексом сопровождения игры. А игра в свою очередь позволяет пользователям не только поддерживать свою физическую форму, но и тренировать интеллект посредством разнообразных загадок и шифров. Запуск системы в рамках университета показал, что система работает качественно, без сбоев и ошибок, а положительные оценки пользователей говорят о том, что система реализована правильно и пользователям она нравится.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] <https://developer.android.com/about/dashboards/index.html> - Интернет портал компании Google, который отражает текущую статистику распространенности Android

[2] <http://en.wikipedia.org/wiki/Android_%28operating_system%29> - Интернет энциклопедия Википедия, обзор операционной системы Android

[3] Рето Майер. Android 2. Программирование приложений для планшетных компьютеров и смартфонов – M.: ЭКСМО, 2011. —672 c.

[4] <https://ru.wikipedia.org/wiki/Клиент-сервер> - Интернет энциклопедия Википедия, обзор операционной архитектуры клиент-сервер

[5] Питер Роб, Карлос Коронел. Системы баз данных: проектирование, реализация и управление – M.: БХВ-Петербург, 2004. —1018 c.

[6] Графический интерфейс пользователя [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя>.

[7] Пономарев И.А. Методы оценки качества пользовательского интерфейса [Электронные данные]. – Режим доступа:

http://it-claim.ru/Library/Books/ITS/wwwbook/ist6/ponomarev2/ponomarev2.htm.

[8] Палицын В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Мн.: БГУИР, 2006. – 76 с.

[9] Постановление Минтруда и соцзащиты РБ от 21.09.2011 № 90 // Национальный реестр правовых актов Республики Беларусь. – 27.09.2011 – № 8/24207.

[10] Горбушие А.М. Экономический эффект программного продукта. – Мн: ВШ, 2007. – 275с.

## ПРИЛОЖЕНИЕ А

(обязательное)

**Код класса JSONParser.java**

package com.vadlabs.coach.api;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.UnsupportedEncodingException;

import java.net.URI;

import java.net.URISyntaxException;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import org.apache.http.HttpEntity;

import org.apache.http.HttpResponse;

import org.apache.http.client.ClientProtocolException;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpDelete;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.ByteArrayEntity;

import org.apache.http.entity.mime.HttpMultipartMode;

import org.apache.http.entity.mime.MultipartEntity;

import org.apache.http.entity.mime.MultipartEntityBuilder;

import org.apache.http.entity.mime.content.StringBody;

import org.apache.http.impl.client.DefaultHttpClient;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

import android.util.Log;

import com.vadlabs.mycoach.app.MyCoachApp;

import com.vadlabs.workouter.database.Program;

import com.vadlabs.workouter.database.ServExercise;

//import org.apache.http.entity.mime.MultipartEntity;

public class JSONParser {

static InputStream is = null;

static JSONObject jObj = null;

static String json = "";

List<HttpGet> https = new LinkedList<HttpGet>();

public HttpClient httpClient;

// constructor

public JSONParser() {

}

@SuppressWarnings("static-access")

public JSONObject registerNewUser(String email, String device\_id) {

JSONObject out = null;

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getRegisterUrl());

HttpPost httpPost = new HttpPost(url);

JSONObject jsonP = new JSONObject();

jsonP.put("email", email);

jsonP.put("deviceId", device\_id);

jsonP.put("os", constants.os);

jsonP.put("appId", MyCoachApp.type);

String toPush = "data=" + jsonP.toString();

Log.i("JPARS", "login params " + jsonP.toString());

httpPost.addHeader("Content-Type",

"application/x-www-form-urlencoded");

ByteArrayEntity baEntity = new ByteArrayEntity(

toPush.getBytes("UTF8"));

httpPost.setEntity(baEntity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

Log.i("JPARS", "REGISTER NEW USER STATUS CODE " + RESP\_CODE);

out = getJSON(is);

if (out == null) {

out = new JSONObject();

}

// JSONObject jCode = new JSONObject();

out.put("code", RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

Log.i("JPARS", "" + out);

return out;

}

@SuppressWarnings("static-access")

public JSONObject registerDeviceID(String device\_id) {

JSONObject out = null;

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getRegisterDeviceUrl());

Log.i("JPARS", constants.getRegisterDeviceUrl());

HttpPost httpPost = new HttpPost(url);

JSONObject jsonP = new JSONObject();

jsonP.put("deviceId", device\_id);

jsonP.put("os", constants.os);

jsonP.put("appId", MyCoachApp.type);

String toPush = "data=" + jsonP.toString();

Log.i("JPARS", "login params " + jsonP.toString());

httpPost.addHeader("Content-Type",

"application/x-www-form-urlencoded");

ByteArrayEntity baEntity = new ByteArrayEntity(

toPush.getBytes("UTF8"));

httpPost.setEntity(baEntity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

Log.i("JPARS", "REGISTER NEW USER STATUS CODE " + RESP\_CODE);

out = getJSON(is);

if (out == null) {

out = new JSONObject();

}

// JSONObject jCode = new JSONObject();

out.put("code", RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

Log.i("JPARS", "" + out);

return out;

}

public JSONObject AUTH(String email) {

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getAuthUrl());

HttpPost httpPost = new HttpPost(url);

JSONObject jsonP = new JSONObject();

jsonP.put("email", email);

String toPush = "data=" + jsonP.toString();

Log.i("JPARS", "login params " + jsonP.toString());

httpPost.addHeader("Content-Type",

"application/x-www-form-urlencoded");

ByteArrayEntity baEntity = new ByteArrayEntity(

toPush.getBytes("UTF8"));

httpPost.setEntity(baEntity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

Log.i("JPARS", "" + getJSON(is));

return getJSON(is);

}

public JSONObject setUserData(String token, String full\_name, String ava,

String fb\_id) {

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getUserUrl());

HttpPut httpPut = new HttpPut(url);

httpPut.addHeader("Content-Type",

"application/x-www-form-urlencoded");

httpPut.addHeader("X-User-Token", token);

// httpPut.addHeader("Content-Type","application/x-www-form-urlencoded");

JSONObject jsonP = new JSONObject();

if (ava != null)

jsonP.put("avatar", ava);

if (fb\_id != null)

jsonP.put("fbId", fb\_id);

if (full\_name != null)

jsonP.put("fullName", full\_name);

// MultipartEntityBuilder builder = MultipartEntityBuilder.create();

// builder.setMode(HttpMultipartMode.BROWSER\_COMPATIBLE);

// builder.addTextBody("data", jsonP.toString());

// builder.addPart("data", new StringBody("SHIT"));

// JSONObject toPush = new JSONObject();

// toPush.put("data", jsonP);

String toPush = "data=" + jsonP.toString();

// toPush = "SHIT";

Log.i("JPARS", "set user json " + jsonP);

ByteArrayEntity baEntity = new ByteArrayEntity(

toPush.getBytes("UTF8"));

// HttpEntity entity = builder.build();

// httpPut.setEntity(entity);

httpPut.setEntity(baEntity);

Log.i("JPARS", "put set user entity lenght "

+ httpPut.getEntity().getContentLength());

HttpResponse httpResponse = httpClient.execute(httpPut);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

Log.i("JPARS", "UPDATE USER STATUS CODE "

+ httpResponse.getStatusLine().getStatusCode());

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return getJSON(is);

}

public JSONObject getUser(String token) {

JSONObject j = new JSONObject();

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getUserUrl());

HttpGet httpGet = new HttpGet(url);

httpGet.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpGet);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

/\*

\* switch (RESP\_CODE) { case 401: try { JSONObject data = new

\* JSONObject(); data.put("code", RESP\_CODE); j = new JSONObject();

\* j.put("data", data);

\*

\* return j; } catch (Exception e) { // TODO: handle exception }

\*

\* break;

\*

\* default: break; }

\*/

Log.i("JPRS", "GET USER RESP CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public JSONObject postExercise(String token, Program programm, int amount,

int hp, int actionID) {

JSONObject j = new JSONObject();

try {

JSONObject data = new JSONObject();

data.put("level", programm.getLevel() - 1);

data.put("exerciseId", programm.getID());

data.put("amount", amount);

data.put("actionId", actionID);

data.put("date", programm.getLastActivity());

data.put("hp", hp);

// j.put("data", data);

Log.i("JPARS", "POST EX " + data);

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getExerciseUrl());

HttpPost httpPost = new HttpPost(url);

httpPost.setHeader("X-User-Token", token);

MultipartEntityBuilder builder = MultipartEntityBuilder.create();

builder.setMode(HttpMultipartMode.BROWSER\_COMPATIBLE);

builder.addTextBody("data", data.toString());

HttpEntity entity = builder.build();

// ByteArrayEntity baEntity = new

// ByteArrayEntity(j.toString().getBytes("UTF8"));

httpPost.setEntity(entity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// switch (RESP\_CODE) {

// case 401:

// try {

// data = new JSONObject();

// data.put("code", RESP\_CODE);

// j = new JSONObject();

// j.put("data", data);

//

// return j;

// } catch (Exception e) {

// // TODO: handle exception

// }

//

// break;

//

// default:

// break;

// }

Log.i("JPRS", "POST EX CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public JSONObject postExercises(String token,

ArrayList<ServExercise> exercises) {

JSONObject j = new JSONObject();

String toPush = null;

JSONArray data = new JSONArray();

try {

for (int i = 0; i < exercises.size(); i++) {

ServExercise ex = exercises.get(i);

JSONObject jEx = new JSONObject();

if (ex.getId() != 0)

jEx.put("exerciseId", ex.getId());

if (ex.getLevel() != 0)

jEx.put("level", ex.getLevel());

if (ex.getAmount() != 0)

jEx.put("amount", ex.getAmount());

if (ex.getDate() != 0)

jEx.put("date", ex.getDate());

if (ex.getActionID() != 0)

jEx.put("actionId", ex.getActionID());

if (ex.getHP() != 0)

jEx.put("hp", ex.getHP());

data.put(jEx);

}

MultipartEntityBuilder builder = MultipartEntityBuilder.create();

builder.setMode(HttpMultipartMode.BROWSER\_COMPATIBLE);

builder.addTextBody("data", data.toString());

Log.d("OLD\_PROGRAMS", data.toString(1));

// MultipartEntity entity=new

// MultipartEntity(HttpMultipartMode.BROWSER\_COMPATIBLE);

//

// entity.addPart("data", new StringBody(data.toString()));

// toPush = "data="+data;

// Log.i("JPARS", "POST EX "+toPush);

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getExercisesUrl());

HttpPost httpPost = new HttpPost(url);

httpPost.setHeader("X-User-Token", token);

// ByteArrayEntity baEntity = new

// ByteArrayEntity(toPush.toString().getBytes("UTF8"));

HttpEntity entity = builder.build();

httpPost.setEntity(entity);

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

Log.d("OLD\_PROGRAMS", "RESP\_CODE " + String.valueOf(RESP\_CODE));

Log.d("OLD\_PROGRAMS", "JSON " + j.toString());

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

/\*

\* if (RESP\_CODE != 200) { j = new JSONObject(); j.put("code",

\* RESP\_CODE); return j; }

\*/

Log.i("JPRS", "POST EX CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (JSONException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public JSONObject getExercises(String token, long sync\_date) {

JSONObject j = new JSONObject();

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

String sUrl = constants.getExercisesUrl();

if (sync\_date != 0)

sUrl = sUrl + "/" + sync\_date;

URI url = new URI(sUrl);

HttpGet httpGet = new HttpGet(url);

httpGet.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpGet);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

Log.d("OLD\_PROGRAMS", "RESP\_CODE " + String.valueOf(RESP\_CODE));

Log.d("OLD\_PROGRAMS", "JSON " + j.toString());

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

/\*

\* switch (RESP\_CODE) { case 401: try { JSONObject data = new

\* JSONObject(); data.put("code", RESP\_CODE); j = new JSONObject();

\* j.put("data", data);

\*

\* return j; } catch (Exception e) { // TODO: handle exception }

\*

\* break;

\*

\* default: break; }

\*/

Log.i("JPRS", "GET EXERSISES RESP CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public void deleteExercise(String token, int deleteID) {

JSONObject j = null;

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getExerciseUrl() + "/" + deleteID);

HttpDelete httpDelete = new HttpDelete(url);

httpDelete.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpDelete);

// HttpEntity httpEntity = httpResponse.getEntity();

// is = httpEntity.getContent();

//

// j = getJSON(is);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public JSONObject getExerciseById(String token, int id) {

JSONObject j = new JSONObject();

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getExerciseUrl() + "/" + id);

HttpGet httpGet = new HttpGet(url);

httpGet.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpGet);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// switch (RESP\_CODE) {

// case 401:

// try {

// JSONObject data = new JSONObject();

// data.put("code", RESP\_CODE);

// j = new JSONObject();

// j.put("data", data);

//

// return j;

// } catch (Exception e) {

// // TODO: handle exception

// }

//

// break;

//

// default:

// break;

// }

Log.i("JPRS", "GET EX CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public JSONObject getExercisesByDate(String token, long sync\_date) {

JSONObject j = new JSONObject();

try {

httpClient = getNewHttpClient();

Constants constants = new Constants();

URI url = new URI(constants.getExerciseUrl() + "s/" + sync\_date);

HttpGet httpGet = new HttpGet(url);

httpGet.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpGet);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// switch (RESP\_CODE) {

// case 401:

// try {

// JSONObject data = new JSONObject();

// data.put("code", RESP\_CODE);

// j = new JSONObject();

// j.put("data", data);

//

// return j;

// } catch (Exception e) {

// // TODO: handle exception

// }

//

// break;

//

// default:

// break;

// }

Log.i("JPRS", "GET EX CODE " + RESP\_CODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

// Log.i("JPARS", "AS "+getJSON(is));

return j;

}

public JSONObject getJSON(InputStream iis) {

try {

BufferedReader reader = new BufferedReader(new InputStreamReader(

iis, "UTF-8"), 8);

StringBuilder sb = new StringBuilder();

String line = null;

while ((line = reader.readLine()) != null) {

sb.append(line + "\n");

// Log.i("JPARS", "line "+line);

}

is.close();

json = sb.toString();

} catch (Exception e) {

Log.e("Buffer Error", "Error converting result " + e.toString());

}

// try parse the string to a JSON object

try {

jObj = new JSONObject(json);

} catch (JSONException e) {

Log.e("JSON Parser", "Error parsing data " + e.toString());

}

// Log.i("JPARS", ""+jObj.toString());

return jObj;

}

public HttpClient getNewHttpClient() {

HttpClient httpClient = new DefaultHttpClient();

return httpClient;

}

public JSONObject getLiderboard(String token, int limit, int skip) {

JSONObject j = new JSONObject();

try {

httpClient = getNewHttpClient();

URI url = new URI(Constants.getLeaderBoardUrl() + "?" + "limit="

+ String.valueOf(limit) + "&" + "skip="

+ String.valueOf(skip));

HttpGet httpGet = new HttpGet(url);

httpGet.setHeader("X-User-Token", token);

HttpResponse httpResponse = httpClient.execute(httpGet);

HttpEntity httpEntity = httpResponse.getEntity();

is = httpEntity.getContent();

j = getJSON(is);

int RESP\_CODE = httpResponse.getStatusLine().getStatusCode();

try {

j.put("code", RESP\_CODE);

} catch (JSONException e) {

e.printStackTrace();

}

} catch (Exception ex) {

ex.printStackTrace();

}

return j;

}

}

## ПРИЛОЖЕНИЕ Б

(обязательное)

**Код разметки главного экрана**

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:background="@android:color/white" >

<ImageView

android:id="@+id/ivTW"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:scaleType="centerCrop"

android:src="@drawable/pushup" />

<RelativeLayout

android:id="@+id/rlTWFade"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:alpha="0"

android:background="#00AB00" >

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlTWHead"

android:layout\_width="match\_parent"

android:layout\_height="50dp"

android:layout\_alignParentTop="true" >

<TextView

android:id="@+id/tvTWTitle"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerVertical="true"

android:layout\_toLeftOf="@+id/tvTWLevel"

android:layout\_toRightOf="@+id/buttonTWBack"

android:gravity="center"

android:paddingLeft="10dp"

android:paddingRight="10dp"

android:text="My Coach"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/white"

android:textStyle="bold" />

<TextView

android:id="@+id/tvTWLevel"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentRight="true"

android:layout\_centerVertical="true"

android:layout\_marginRight="10dp"

android:gravity="center\_horizontal"

android:text="7\nlevel"

android:textAppearance="?android:attr/textAppearanceMedium"

android:textColor="@android:color/white" />

<TextView

android:id="@+id/tvTWLevelDescr"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_above="@+id/tvTWLevel"

android:layout\_alignParentRight="true"

android:layout\_centerHorizontal="false"

android:layout\_marginRight="7dp"

android:layout\_marginTop="3dp"

android:textAppearance="?android:attr/textAppearanceSmall"

android:textColor="@android:color/white" />

<Button

android:id="@+id/buttonTWBack"

android:layout\_width="35dp"

android:layout\_height="35dp"

android:layout\_alignParentLeft="true"

android:layout\_alignParentTop="false"

android:layout\_centerVertical="true"

android:layout\_marginLeft="10dp"

android:background="@drawable/close\_but" />

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlWorkoutTop"

android:layout\_width="match\_parent"

android:layout\_height="100dp"

android:layout\_above="@+id/buttonTWDone" >

<RelativeLayout

android:id="@+id/rlWOne"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/empty\_circule" >

<TextView

android:id="@+id/tvTWOne"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="1"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/black" />

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlWTwo"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/empty\_circule" >

<TextView

android:id="@+id/tvTWTwo"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="2"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/black" />

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlWThree"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/empty\_circule" >

<TextView

android:id="@+id/tvTWThree"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="3"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/black" />

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlWFour"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/empty\_circule" >

<TextView

android:id="@+id/tvTWFour"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="4"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/black" />

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlWFive"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:background="@drawable/empty\_circule" >

<TextView

android:id="@+id/tvTWFive"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="5"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/black" />

</RelativeLayout>

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlTWProgressWeel"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true" >

<RelativeLayout

android:id="@+id/rlTWCircleOne"

android:layout\_width="170dp"

android:layout\_height="170dp"

android:layout\_centerInParent="true"

android:background="@drawable/big\_circ\_big" >

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlTWCircleTwo"

android:layout\_width="180dp"

android:layout\_height="180dp"

android:layout\_centerInParent="true"

android:alpha="0.5"

android:background="@drawable/big\_circ\_big" >

</RelativeLayout>

<RelativeLayout

android:id="@+id/rlTWCircleThree"

android:layout\_width="190dp"

android:layout\_height="190dp"

android:layout\_centerInParent="true"

android:alpha="0.2"

android:background="@drawable/big\_circ\_big" >

</RelativeLayout>

<TextView

android:id="@+id/tvProgressWheel"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:gravity="center\_horizontal"

android:text="10"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/white"

android:textSize="60sp" />

</RelativeLayout>

<TextView

android:id="@+id/tvTWDescription"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_above="@+id/rlTWProgressWeel"

android:layout\_centerInParent="true"

android:layout\_marginBottom="20dp"

android:layout\_marginLeft="20dp"

android:layout\_marginRight="20dp"

android:layout\_marginTop="10dp"

android:gravity="center\_horizontal"

android:text="@string/to\_do"

android:textAppearance="?android:attr/textAppearanceLarge"

android:textColor="@android:color/white" />

<RelativeLayout

android:id="@+id/rlTWFooterFade"

android:layout\_width="match\_parent"

android:layout\_height="100dp"

android:layout\_alignParentBottom="true"

android:background="@drawable/fade\_tw" >

</RelativeLayout>

<Button

android:id="@+id/buttonTWDone"

android:layout\_width="145dp"

android:layout\_height="50dp"

android:layout\_alignParentBottom="true"

android:layout\_alignParentLeft="true"

android:layout\_margin="10dp"

android:background="@null"

android:text="@string/done"

android:textColor="#00CC00" />

<Button

android:id="@+id/buttonTWTried"

android:layout\_width="145dp"

android:layout\_height="50dp"

android:layout\_alignParentBottom="true"

android:layout\_alignParentRight="true"

android:layout\_margin="10dp"

android:background="@null"

android:text="@string/tried"

android:textColor="#FF0000" />

</RelativeLayout>