МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка программного средства «Iucome – финансовый менеджер»

Выполнил студент Пилик Семён Игоревич

(Ф.И.О.)

Руководитель проекта пр. ст. Бондарчик Е.Н.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой д.к.т.н., Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты пр. ст. Бондарчик Е.Н.

(учен. степень, звание, должность, подпись, Ф.И.О.)

ст. преп. Блинова Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтроллер пр. ст. Бондарчик Е.Н.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 25 страниц, 7 источников литературы и одного приложения.

Основными целями курсового проекта являются: разработка программного средства – «Iucome – финансовый менеджер».

Пояснительная записка состоит из введения, пяти разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе представлена постановка задачи.

Во втором разделе рассматривается разработка архитектуры проекта.

В третьем разделе рассматривается разработка функциональной модели приложения.

В четвертом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки программного средства.

Содержание

[Введение 6](#_Toc515317847)

[1. Постановка задачи. 7](#_Toc515317848)

[2. Архитектура проекта 9](#_Toc515317849)

[3. Схема базы данных 10](#_Toc515317850)

[4. Программная реализация приложения 17](#_Toc515317851)

[4.1 Реализация сервера 17](#_Toc515317852)

[4.2 Реализация мобильного приложения 17](#_Toc515317853)

[4.2.1 REST-клиент 18](#_Toc515317854)

[5. Руководство пользователя 21](#_Toc515317855)

[5.1 Android приложение 21](#_Toc515317856)

[Заключение 27](#_Toc515317857)

[Список использованных источников 28](#_Toc515317858)

[Приложение А 29](#_Toc515317859)

[Приложение B 29](#_Toc515317860)

# Введение

Мобильное приложение — это программное обеспечение, специально разработанное под конкретную мобильную платформу (iOS, Android, Windows Phone и т. д.). Предназначено для использования на смартфонах, фаблетах, планшетах, умных часах и других мобильных устройствах.

Основные плюсы мобильных приложений: интерфейс программы создан конкретно под работу на мобильном устройстве через сенсорный экран или кнопки,

удобная и понятная для пользователей гаджетов навигация, мобильное меню, лучшее взаимодействие с пользователем через сообщения, пуш-уведомления, напоминания. Приложение может выполнять функции даже в фоновом режиме, чего нельзя сказать о сайте. Для работы с программой не нужно открывать браузер, а многие приложения поддерживают ряд функций и при отключенном интернете.

Темой курсового проекта является разработка программного средства «Iucome – финансовый менеджер» - мобильного приложения, основной задачей которого является отслеживание доходов и расходов. Отслеживание долгов.

Целью курсового проекта является проектирование и реализация программного средства. А так же изучение новых, ранее не знакомых, технологий.

# Постановка задачи.

**Express.js**, или просто Express, фреймворк web-приложений для Node.js, реализованный как [свободное и открытое программное обеспечение](https://ru.wikipedia.org/wiki/Свободное_и_открытое_программное_обеспечение) под [лицензией MIT](https://ru.wikipedia.org/wiki/Лицензия_MIT). Он спроектирован для создания [веб-приложений](https://ru.wikipedia.org/wiki/Веб-приложение) и [API](https://ru.wikipedia.org/wiki/API)[[2]](https://ru.wikipedia.org/wiki/Express.js" \l "cite_note-ExpressJS-2). Де-факто является стандартным каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описывает его как созданный на основе написанного на языке [Ruby](https://ru.wikipedia.org/wiki/Ruby) каркаса [Sinatra](https://ru.wikipedia.org/wiki/Sinatra), подразумевая, что он минималистичен и включает большое число подключаемых [плагинов](https://ru.wikipedia.org/wiki/Плагин).

**Visual Studio Code** — [редактор исходного кода](https://ru.wikipedia.org/wiki/Редактор_исходного_кода), разработанный Microsoft для [Windows](https://ru.wikipedia.org/wiki/Windows), Linux и [macOS](https://ru.wikipedia.org/wiki/MacOS). Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки [веб](https://ru.wikipedia.org/wiki/Веб-приложение)- и облачных приложений. Включает в себя [отладчик](https://ru.wikipedia.org/wiki/Отладчик), инструменты для работы с Git ,[подсветку синтаксиса](https://ru.wikipedia.org/wiki/Подсветка_синтаксиса), IntelliSense и средства для [рефакторинга](https://ru.wikipedia.org/wiki/Рефакторинг). Имеет широкие возможности для [кастомизации](https://ru.wikipedia.org/wiki/Кастомизация): пользовательские темы, сочетания клавиш и [файлы конфигурации](https://ru.wikipedia.org/wiki/Файл_конфигурации). Распространяется бесплатно, разрабатывается как программное обеспечение с [открытым исходным кодом](https://ru.wikipedia.org/wiki/Открытое_программное_обеспечение), но готовые сборки распространяются под [проприетарной лицензией](https://ru.wikipedia.org/wiki/Проприетарное_программное_обеспечение).

**Flutter** — SDK с открытым исходным кодом для создания мобильных приложений от компании [Google](https://ru.wikipedia.org/wiki/Google_(компания)). Он используется для разработки приложений под Android и [iOS](https://ru.wikipedia.org/wiki/IOS), а также это пока единственный способ разработки приложений под [Google Fuchsia](https://ru.wikipedia.org/wiki/Google_Fuchsia).  
 Приложения Flutter пишутся на языке **Dart**. В Android, а также под [Windows](https://ru.wikipedia.org/wiki/Windows), macOS и Linux с помощью Flutter Desktop Embedding, Flutter работает в виртуальной машине Dart с J[IT-компилятором](https://ru.wikipedia.org/wiki/JIT-компилятор). Из-за ограничений на динамическое выполнение кода в [App Store](https://ru.wikipedia.org/wiki/App_Store), под iOS Flutter использует [AOT-компиляцию](https://ru.wikipedia.org/wiki/AOT-компиляция).Одно из главных преимуществ платформы Dart — «горячая перезагрузка», когда изменение исходного кода применяется сразу в работающем приложении без необходимости его перезапуска.

**PostgreSQL**— это объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), основанная на POSTGRES, Version 4.2 — программе, разработанной на факультете компьютерных наук Калифорнийского университета в Беркли. В POSTGRES появилось множество новшеств, которые были реализованы в некоторых коммерческих СУБД гораздо позднее.

PostgreSQL — СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций;

При разработке сервера следует реализовать следующие API:

* REST – это стиль архитектуры программного обеспечения для распределенный систем, таких как WorldWideWeb, который, как правило используется для построения веб-служб.
* SOAP – протокол обмена структурированными сообщениями в распределенной вычислительной среде. Первоначально предназначался для реализации удаленного вызова процедур, сейчас используется для обмена сообщениями в формате XML или JSON.

Клиентом будет выступать мобильное приложение на устройстве с операционной системой Android 4.0.3 и выше.

# Архитектура проекта

Распределенное приложение состоит из 2-х частей:

Мобильное приложение. Эта часть состоит из Dart-классов, с помощью которых реализована основная логика мобильного приложения и осуществляется обращение к серверу.

Сервер. Данная часть предоставляет Node.js приложение, которое обрабатывает запросы с клиента с помощью контроллеров и взаимодействует с базой данных посредством библиотеки pg-promise.

Графическое представление архитектуры распределенного приложения изображено на рисунке 2.1.

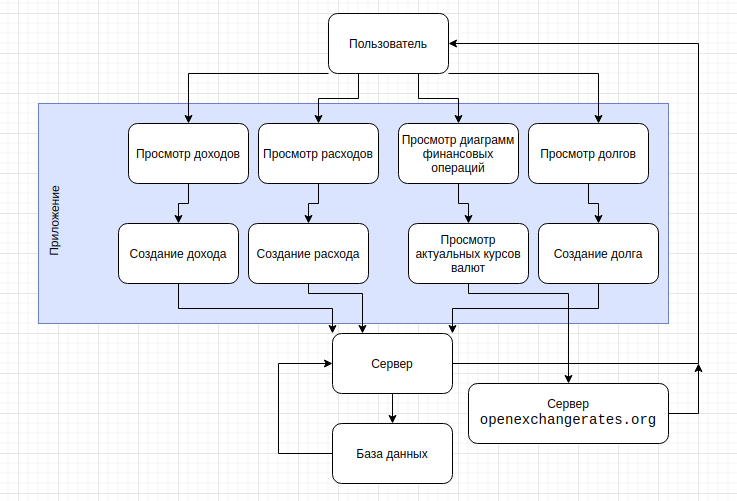


Рисунок. 2.1 – архитектура распределенного приложения

# Схема базы данных

В ходе написания приложения была разработана удаленная база данных «Iucome»,схема которой представлена на рисунке 3.1.

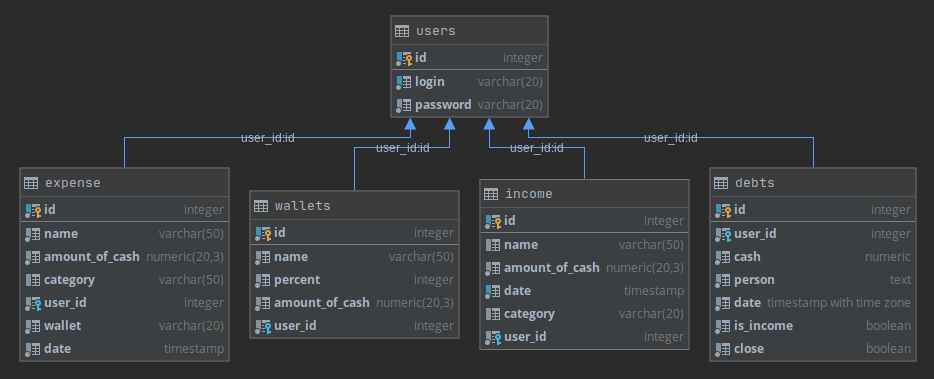


Рисунок 3.1 – структура базы данных

Таблица Users предназначена для хранения пользователей, зарегистрированных в приложении. Wallets – для хранении информации о кошельках Incomes и Expenses — для хранения информации о доходах и расходах соответственно. Debts — отслеживание долгов.

# Программная реализация приложения

## 4.1 Реализация сервера

Для реализации сервера я использовал язык Node.js и го фраемворк Express.js в первую очередь потому что я поставил себе задачу испытать новые языки, а во вторую, потому что это один из популярнейших языков для написания сервера на данный момент.

Мой сервер состоит из 11 обработчиков, которые в свою очередь обращаются к базе данных и возвращают результат.

Сервер имеет следующие пути для обработки HTTP-запросов (Приложение А)

* /debts – получение долгов пользователя;
* /debts/add – добавление долга пользователя;
* /debts/close – закрытие долга пользователя;
* /debts/open – возобновление долга пользователя;
* /users/create – регистрация пользователя;
* /users/auth – аутентификация пользователя;
* /db/createEx – создание расхода пользователя;
* /db/createInc – создание дохода пользователя;
* /db/expenses – получение расходов пользователя;
* /db/incomes – получение доходов пользователя;
* / - проверка соединения

Обращение к базе данных осуществляется с помощью библиотеки pg-promise описанной в официальной документации postgresql.

Код сервера описан в Приложении А.

## 4.2 Реализация мобильного приложения

Мобильное приложение реализовано для операционной системы Android. Данное приложение было разработано в среде разработки –Visual Studio Code.

Графическая структура приложения отображена на рисунке 4.2.1

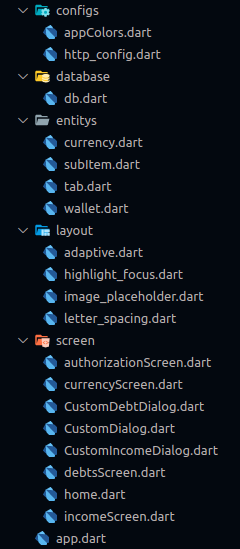


Рисунок 4.2.1 – структура проекта

### Проект в себе содержит 5 классов страниц:

1. incomeScreen – содержит страницу доходов, с выводом и добавлением.
2. home – содержит страницу графиков операций, кошельки и расходы, с выводом и добавлением.
3. debtsScreen – класс, содержащий долги.
4. authorizationScreen – экран авторизации.
5. currencyScreen – экран с курсами валют.

Остальные классы и их содержимое отвечают за конфигурацию, взаимодействие с локальной базой данных и сервером, описание основных сущностей, синхронизация данных.

### **4.2.1 Реализация REST**

Для взаимодействия мобильного приложения с сервером применяется архитектурный стиль REST. Основная задача сервера в этой архитектуре предоставить клиентам доступ к ресурсам по их идентификаторам. Под доступом подразумевается как получение информации, так и ее изменение или удаление. В зависимости от реализации сервиса, в ответ на наш запрос придёт ответ в формате JSON. На рисунке 4.2.1.1 представлен пример одного из get-запросов.

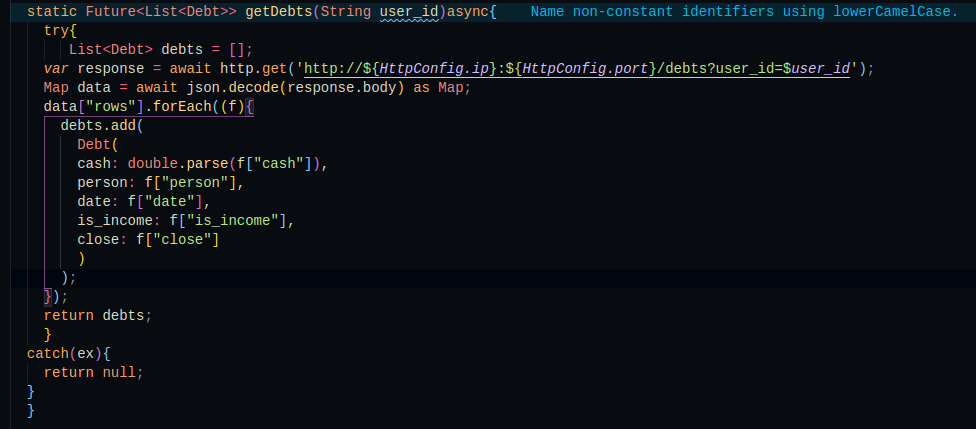


Рисунок 4.2.1.1 – get-запрос

В данном методе можно увидеть, что в него передаётся ID, по которому затем будет производить get-запрос. Внутри метода есть объект response, который будет содержать ответ от сервера, в нашем случае JSON-файл, содержащий объекты типа Events.

На рисунке 4.2.1.2 приведен пример post-запроса.

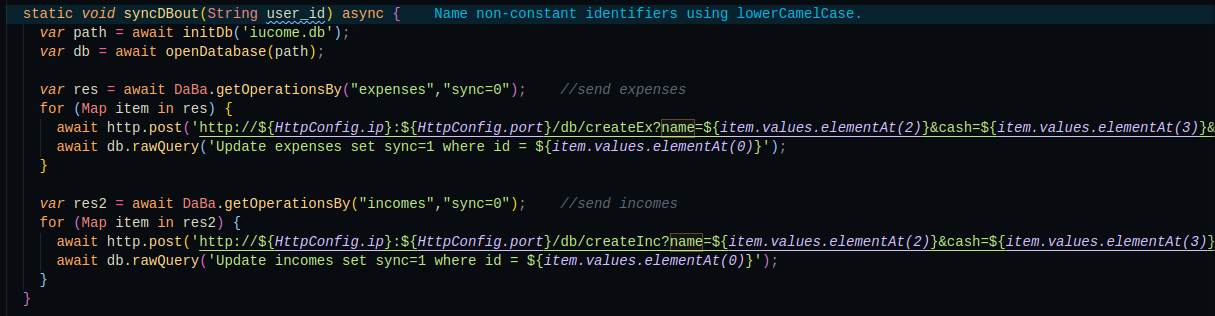


Рисунок 4.2.1.2 – post-запроса

В данном примере в метод передаётся логин и пароль пользователя.

Затем в случае успеха на клиент придет ID пользователя, которое в дальнейшем будет использоваться для различных манипуляций с приложением.

GET – позволяет получить данные, а также конкретный ответ по заданным параметрам. POST – позволяет передать данные на сервер.

Таким образом можно сказать, что REST – это взаимодействие «клиент-серверной» архитектуры с помощью HTTP-запросов и HTTP-ответов по протоколу, не подтверждающему состояние.

# Руководство пользователя

## 5.1 Android приложение

Для начала необходимо установить приложение на мобильный устройство. Запустим наше приложение. Сразу после запуска мы увидим окно авторизации (Рисунок 5.1.1).

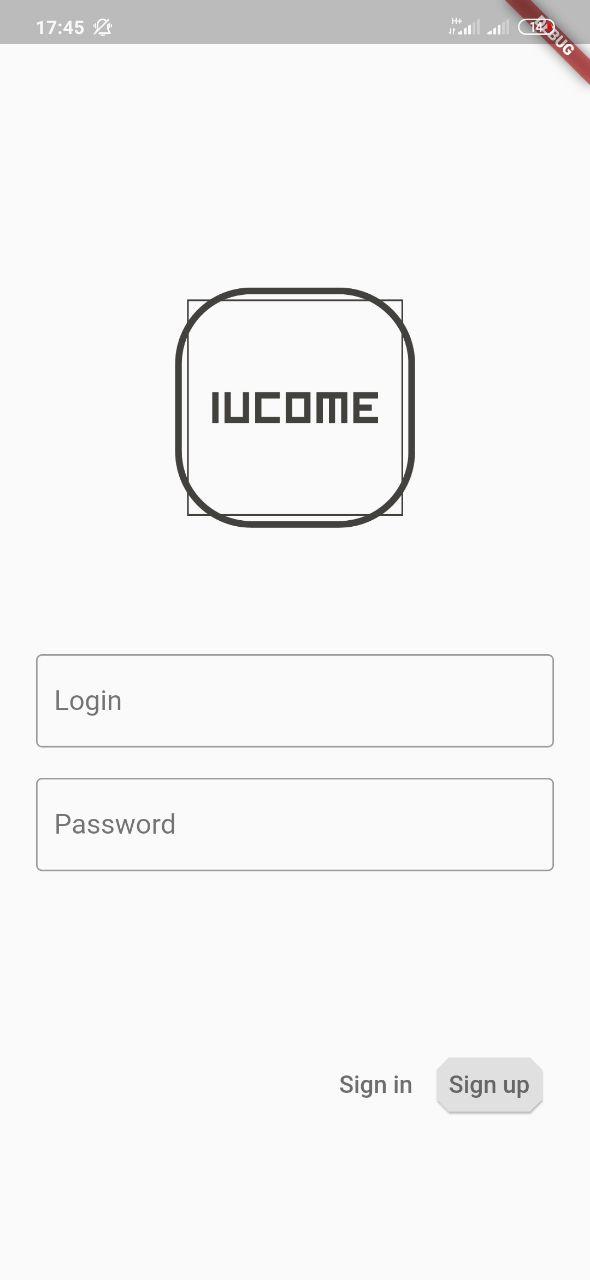


Рисунок 5.1.1 – окно авторизации

Если пользователь еще не зарегистрирован в системе, то необходимо ввести данные и нажать на поле «Регистрация» и произойдет регистрация (Рисунок 5.1.2).

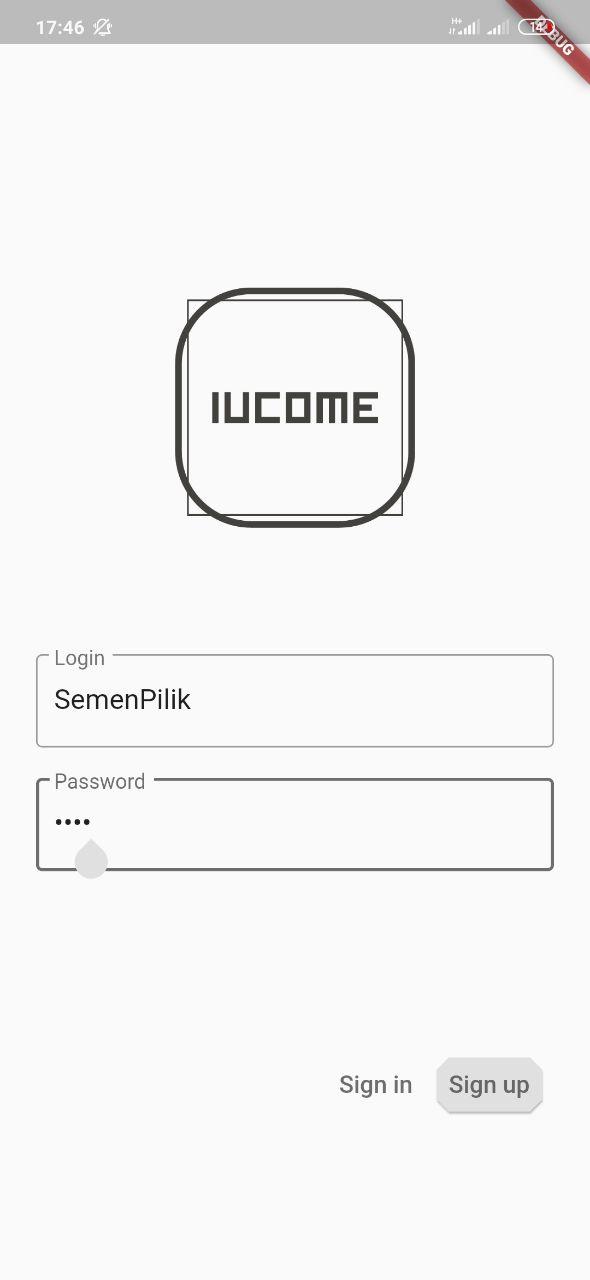
******

Рисунок 5.1.2 – окно регистрации

Если успешно пройти авторизацию или регистрацию пользователя на первой странице, то произойдет переход на главную страницу с четырьмя экранами на которых можно просмотреть списки расходов, доходов, долгов и граффики, но при первом запуске они будут пустыми (Рисунок 5.1.3).



Рисунок 5.1.3 – главная страница

При нажатии на кнопку «+» происходит переход на страницу создания дохода или расхода, где пользователь может ввести такие данные как: название дохода или расхода, сумма, выбрать или создать категорию и выбрать кошелёк (или не выбирать если это доход). Идея заключается в том, что мной заранее предопределены 7 кошельков между которыми разделяется доход согласно их проценту от дохода. (Рисунок 5.1.4).

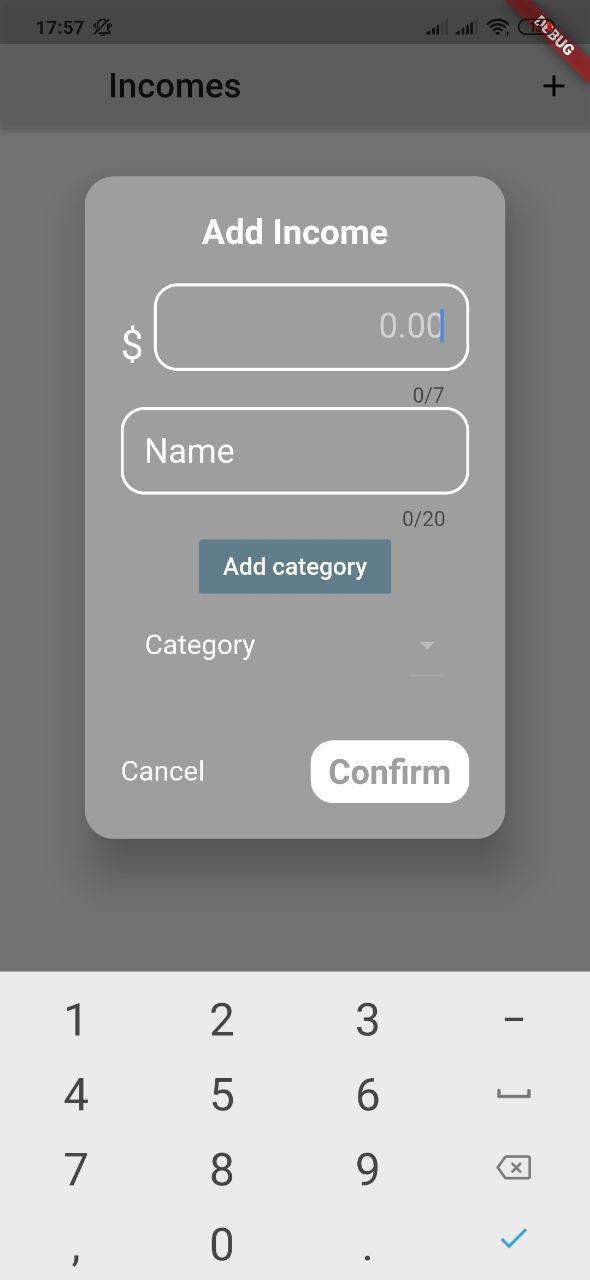


Рисунок 5.1.4 – создание дохода

После ввода данных и нажатии кнопки «создать» произойдет создание события и переход на главный экран приложения.(Рисунок 5.1.5)

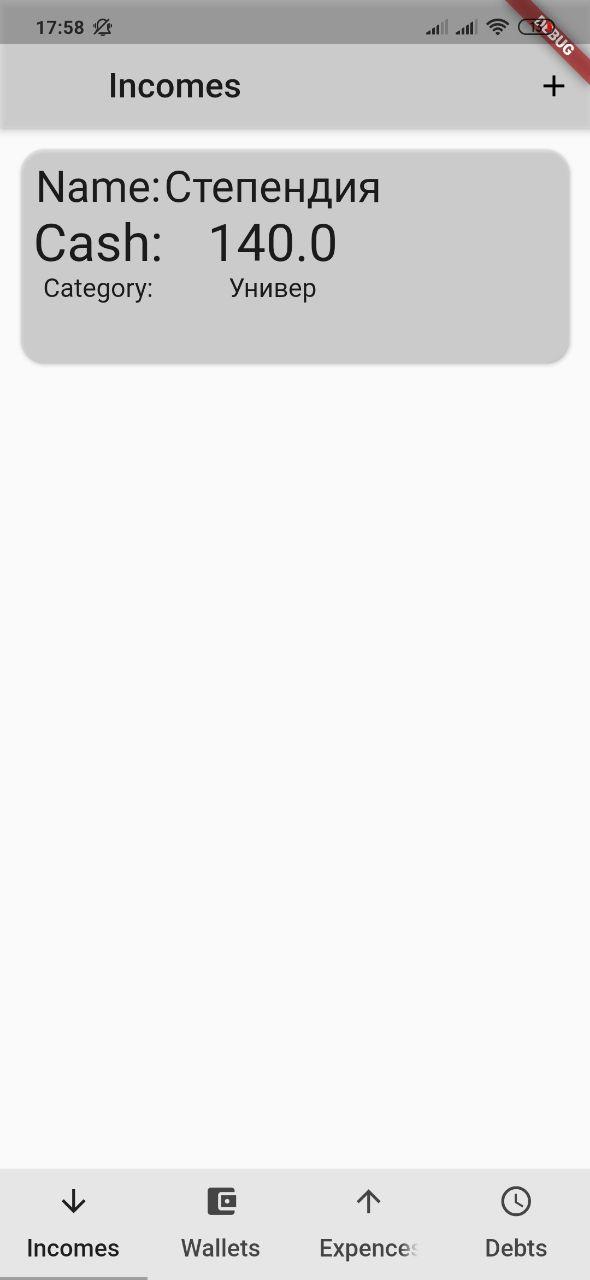
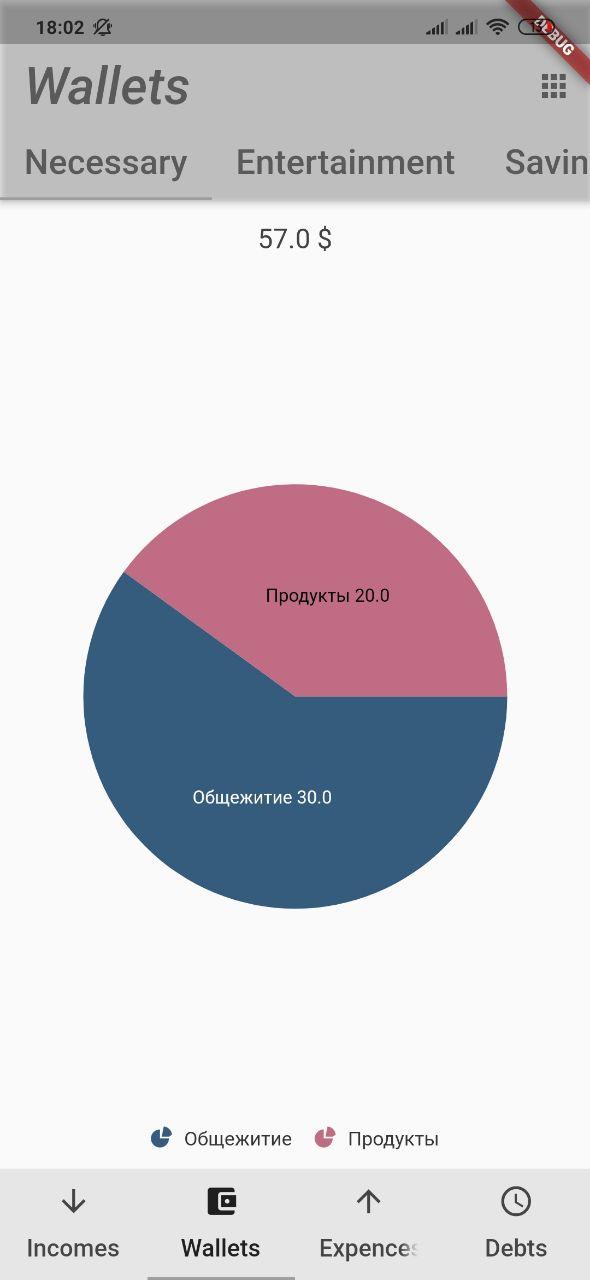


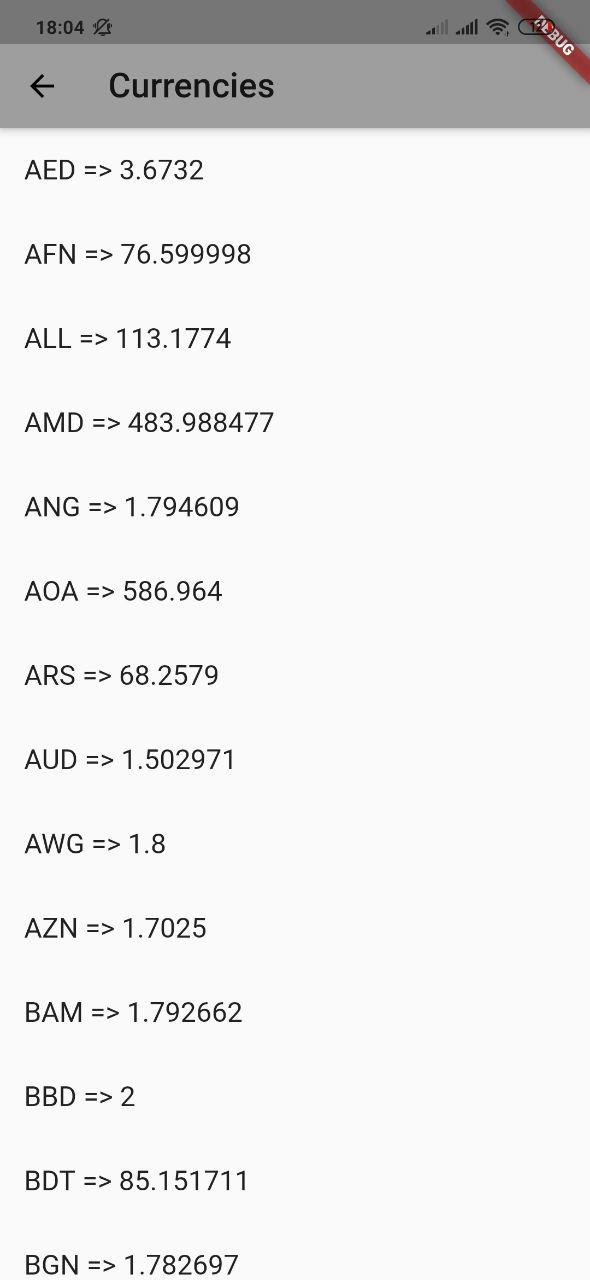
Рисунок 5.1.5 – экран дохода

Если удержать палец на элементе списка , то откроется окно создания расхода или дохода с автоматически-веденными данными их шаблона. Таким образом можно многократно повторять уже выполненные операции.

Если заполнить приложение данными то в окне кошельков появляются графики расходов по категориям и сумма в кошельках (Рисунок 5.1.6).

Рисунок 5.1.6 – окно кошельков

При нажатии на кнопку в верхнем правом углу открывается окно курсов валют(Рисунок 5.1.7)

Рисунок 5.1.7 – окно курсов валют

Если обобщить все вышесказанное и показанное, то на данных скриншотах приведена работа приложение и с помощью текста изложены все нюансы при работе с ним.

При создании долга , создаётся элемент красного цвета(не выплачен)Б при нажатии на него цвет меняется на зеленый (выплачен)(Рисунок 5.1.8)

Рисунок 5.1.8 – не выплаченный долг

Так же хотелось бы сказать что в приложении присутствует синхронизация локальной и удалённой баз данных. Локальная база данных повторяет удаленную и создана для локального хранения данных (без сети). Все данные при авторизации выгружаются с удалённого сервера на локальную базу данных и последующее взаимодействие происходит с локальной бд. В момент следующей авторизации базы данных обмениваются данными.

# Заключение

Решая поставленную задачу, я пришел к удовлетворительному результату. Таким образом, была достигнута цель, и был создан проект «Iucome», который включает в себя android–клиент и сервер.

Программная система имеет простой, интуитивно понятный пользователю интерфейс.

Были использованы технологии ни с одной из которых я не работал ранее.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанный продукт работает корректно, а требования технического задания выполнены в полном объеме.

В результате приложение обладает следующими особенностями:

* получение данных с сервера;
* реализована отправка данных на удалённый сервер по технологии REST API;
* реализован полностью работоспособный сервер и удалённая база данных.

# Список использованных источников

1. Stackoverflow - https://stackoverflow.com;

2. Cyberforum - https://cyberforum.ru;

3. Github - https://github.com;

4. Android documentation - <https://developer.android.com/docs>.

5. Java documentation - <https://docs.oracle.com/en/java/>

6. ASP.NET documentation - <https://docs.microsoft.com/en-us/aspnet/core/>

7. SQLServer documentation - <https://docs.microsoft.com/en-us/sql/sql-server/>

# Приложение А

const express = require('express');

const app = express();

var pgp = require("pg-promise")();

var conf = require("./db\_config.json");

const connectionConf = {

host: conf.host,

port: conf.port,

database: conf.database,

user: conf.user,

password: conf.password

}

var db = pgp(connectionConf);

app.get('/debts',function(req,res){

db.result('select \* from get\_debt\_by\_id($1)',[req.query.user\_id]).then((data)=>{

res.send(data);

})

})

app.get('/debts/add',function(req,res){

let dbString = `select add\_debt(${req.query.user\_id},${req.query.cash},${req.query.person},${req.query.is\_income})`;

db.result(dbString);

res.send('ok');

})

app.get('/debts/close',function(req,res){

let dbString = `update debts set close=true where user\_id=${req.query.user\_id} and person=${req.query.person}`

db.result(dbString);

res.send('ok');

})

app.get('/debts/open',function(req,res){

let dbString = `update debts set close=false where user\_id=${req.query.user\_id} and person=${req.query.person}`

db.result(dbString);

res.send('ok');

})

//Create user

app.post('/users/create', function(req,res){

db.result("Select create\_user($1,$2)",[req.query.login,req.query.pass]).then((data)=>{

if(data.rows[0].create\_user != null) res.send(data.rows[0].create\_user);

else res.send('none');

});

})

//auth

app.post('/users/auth', function(req,res){

db.result("Select check\_password\_by\_login($1,$2)",[req.query.login,req.query.pass]).then((data)=>{

console.log("Auth",req.query.login,req.query.pass);

if(data.rows[0].check\_password\_by\_login != null)

res.send(`${data.rows[0].check\_password\_by\_login}`);

else res.send('none');

});

})

//add expense

app.post('/db/createEx',(req,res)=>{

db.result('select add\_expense($1,$2,$3,$4,$5);',[req.query.name,req.query.cash,req.query.category,req.query.user,req.query.wallet]).then((data)=>{

res.send(data);

});

})

//add income

app.post('/db/createInc',(req,res)=>{

db.result('select add\_income($1,$2,$3,$4);',[req.query.name,req.query.cash,req.query.category,req.query.user]).then((data)=>{

res.send(data);

});

})

app.get('/db/expenses',(req,res)=>{

db.result('select \* from get\_expenses\_by\_id($1,$2)',[req.query.user\_id,req.query.date]).then((data)=>{

res.send(data);

})

})

app.get('/db/incomes',(req,res)=>{

db.result('select \* from get\_incomes\_by\_id($1,$2)',[req.query.user\_id,req.query.date]).then((data)=>{

res.send(data);

})

})

app.get('/',(req,res)=>{

res.send("ok");

})

app.listen(3002, function () {

console.log('Example app listening on port 3002!');

});