**Реферат**

Node.js, React, RestFull-API, MySQL, Sequelize, express, axios.

Основной целью курсового проекта является разработка веб-приложения «Каталог стартапов».

Пояснительная записка состоит из введения, пяти разделов и заключения.

В введении представлена общая информация, дающая представление о предстоящей работе.

В первом разделе рассматриваются алгоритмы решения и актуальность разработки проекта.

Во втором разделе представлена основная архитектура проекта, модели данных и их взаимодействие.

В третьем разделе рассматривается работа с базой данных и произведен обзор контроллеров.

В четвёртом разделе показывается пользовательский функционал приложения.

В пятом разделе описывается тестирование приложения.

Заключение состоит из итогов курсового проекта и задач, решенных в ходе проектирования и разработки приложения.

Пояснительная записка курсового проекта содержит 31 страницу, 24 рисунка, 4 источника литературы, 4 приложения.

Содержание

[Введение 3](#_Toc513755486)

[1. Постановка задачи 4](#_Toc513755487)

[1.1. Алгоритмы решения 4](#_Toc513755488)

[1.2. Обзор прототипов 4](#_Toc513755489)

[1.3. Актуальность задачи 5](#_Toc513755490)

[2. Разработка архитектуры проекта 6](#_Toc513755491)

[2.1. Обобщенная структура 6](#_Toc513755492)

[2.2. Структура моделей и классов 8](#_Toc513755493)

[3. Разработка функциональной модели и модели данных ПС 11](#_Toc513755494)

[3.1. Выполняемые функции 11](#_Toc513755495)

[3.2. Модель базы данных 12](#_Toc513755496)

[4. Руководство пользователя 13](#_Toc513755497)

[5. Тестирование приложения 17](#_Toc513755498)

[5.1. Преимущества и недостатки функционального тестирования 17](#_Toc513755499)

[5.2. Процесс тестирования приложения курсового проекта 17](#_Toc513755500)

[5.3. Тестирование с использованием Unit-тестов 19](#_Toc513755501)

[Заключение 21](#_Toc513755505)

[Список использованных источников 22](#_Toc513755506)

[ПРИЛОЖЕНИЕ А 23](#_Toc513755507)

[ПРИЛОЖЕНИЕ Б 25](#_Toc513755508)

[ПРИЛОЖЕНИЕ В 27](#_Toc513755509)

[ПРИЛОЖЕНИЕ Г 30](#_Toc513755510)

# Введение

**Стартап**— это временная структура, предназначенная для поиска и реализации масштабируемой бизнес-модели.

Более простыми словами можно определить стартап как новый коммерческий проект, который создаётся с целью получения прибыли от бизнеса после его успешного развития.

Классические стартапы имеют одну характерную черту – они имеют уникальную (оригинальную, эксклюзивную) идею. Настоящий стартап никогда не копирует уже известные коммерческие проекты, а представляет собой нечто кардинально новое.

Стартап должен быть удачным и востребованным, чтобы привлечь целевую аудиторию, а вместе с ней – деньги инвесторов.

В рамках курсового проекта необходимо разработать веб-приложение, с помощью которого пользователь сможет опубликовать свой стартап, найти интересные проекты и даже присоединиться к команде.

Главный показатель успешности нового начинания – готовность бизнес-партнеров вкладывать в него свои финансы. Не всегда стартап основан на чистой коммерции: он бывает информационным, гуманитарным, научным, но в любом случае – перспективным.

Начальная стадия работы над любым стартапом – это создание его прототипа. По мере превращения прототипа в полноценный продукт стартап трансформируется, развивается и масштабируется. Под масштабированием понимается увеличение сферы влияния проекта, связанное с вложением финансов.

Данное приложение поможет пользователю как на начальном уровне развития стартапа для получения помощи, так и на стадии его завершения.

# Постановка задачи

Целью курсового проекта является разработка приложения «Каталог-стартапов». Данное приложение должно состоять из двух частей: сервера и UI для браузера.

## 1.1. Алгоритмы решения

Для реализации курсового проекта использовалась технология React Router v4 для части front-end. Back-end часть была написана на Node.js, в качестве базы данных использована реляционная база MySQL.

## 1.2. Обзор прототипов

На сегодняшний день существует большое количество веб-приложений, являющихся каталогами стартапов. В качестве примера на рисунке 1.2.1 предоставлен сайт http://www.towave.ru/start.

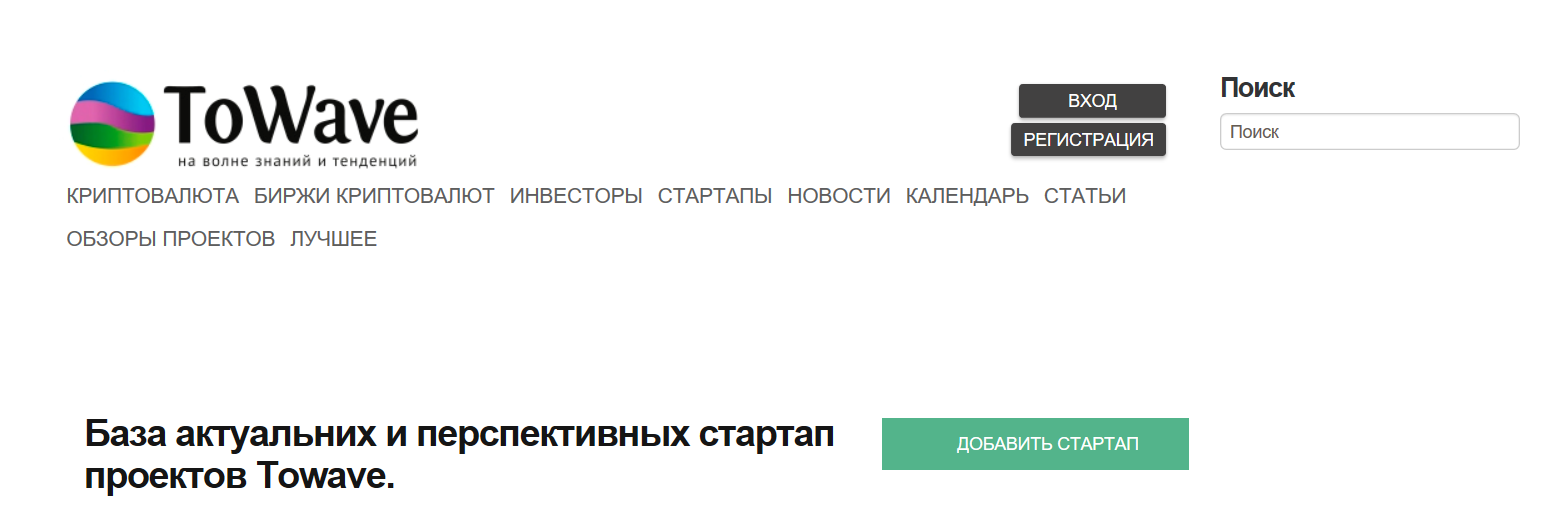


Рисунок 1.2.1 – пример каталога стартапов

Данный сайт предоставляет пользователю возможность просмотра стартапов, навигации и удобного поиска.

Рассмотрим еще один пример каталога стартапов на рисунке 1.2.2

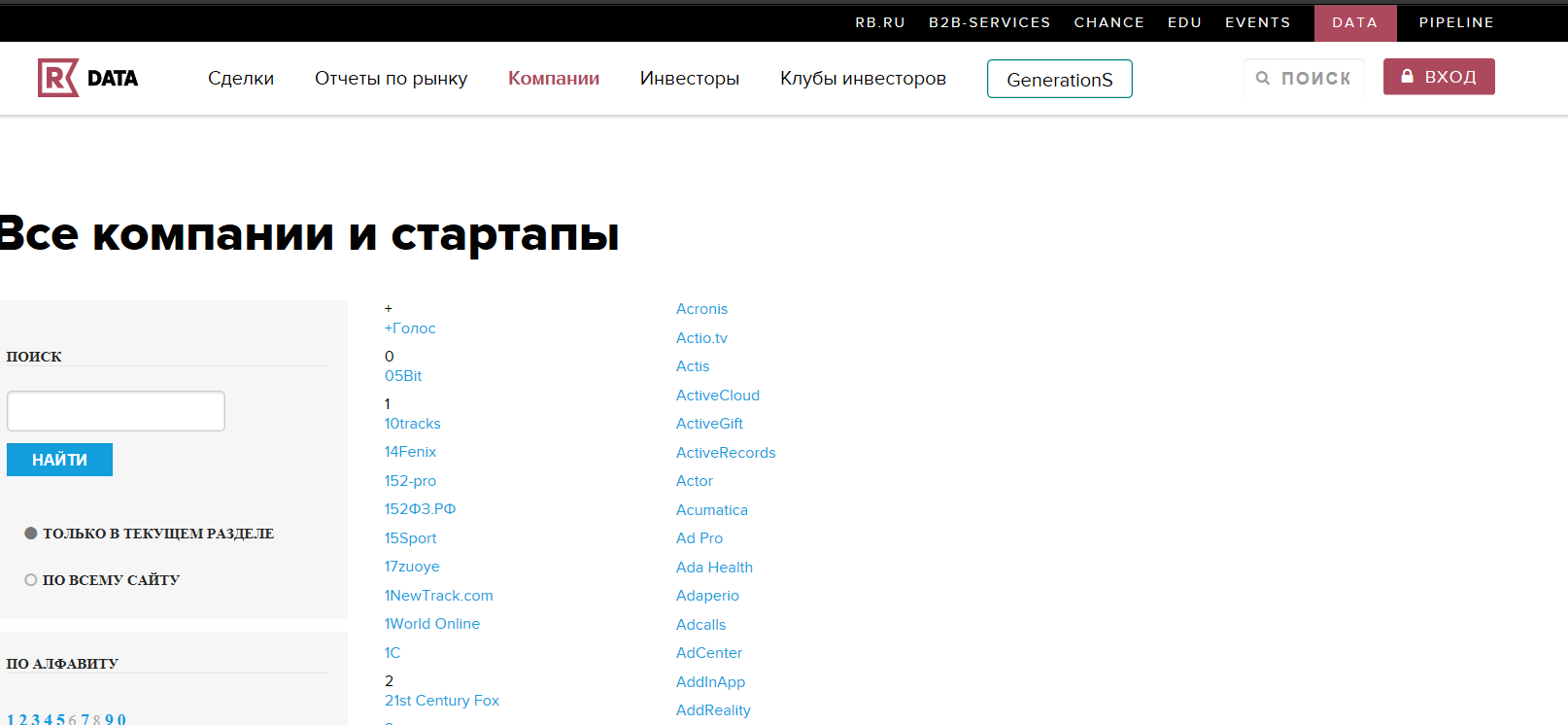


Рисунок 1.2.2 – ещё один пример каталога стартапов

Исходя из рассмотренных примеров можем заметить, что большинство сайтов предоставляют огромный функционал для поиска, однако данный поиск не является обоснованным, по причине того, что редко пользователь ищет стартап по конкретному имени. Поэтому большинство из этого огромного функционала просто не используется.

## 1.3. Актуальность задачи

В данный момент в Беларуси развивается порядка 6234 зарегистрированных стартапов. С ростом количества стартапов растёт и их дифференциация. Встречаются не только новомодные стартапы связанные с ИТ-технологиями в области дополненной, виртуальной реальности и искусственного интеллекта, но и стартапы по организации велозаездов, спортивных мероприятий и даже организации деятельности оркестров.

В Беларуси проводится огромное количество хакатонов и конкурсов, на которых может зародится и развиться стартап почти в любой сфере деятельности.

# Разработка архитектуры проекта

Архитектура проекта будет основываться на архитектуре двух его основных частей: back-end и front-end.

## 2.1. Обобщенная структура

Существует множество различных видов и типов архитектур, которые успешно применяются. Одной их наиболее используемых является классическая трехуровневая система, которая подразумевает разделение приложения на три уровня.

В качестве архитектуры back-end части использовалась N-layer архитектура. Рассмотрим её подробнее.

Layer представляет логический уровень. То есть у нас может быть уровень доступа к данным, уровень бизнес-логики, уровень представления, уровень сервисов и так далее. При этом логические уровни не совпадают с физическими. Так, обычно уровень предоставления в приложении содержит и контроллеры, которые обрабатывают ввод, и представления, которые отображаются в веб-браузере, то есть разделяется на два физических уровня.

Классическая трехуровневая система состоит из уровней, показанных на рисунке 2.1.1

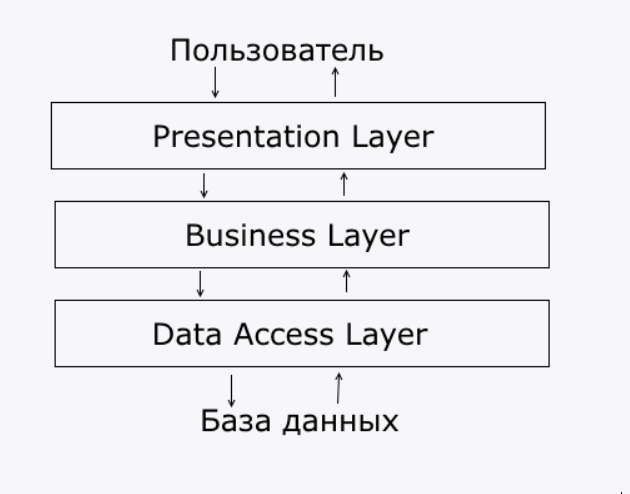
****

Рисунок 2.1.1 – трёхуровневая система

**Presentation layer** (уровень представления): это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. На данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы html, javascript), а также модели представлений, контроллеры, объекты контекста запроса.

**Business layer** (уровень бизнес-логики): содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

**Data Access layer** (уровень доступа к данным): хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Приведём пример реализации многоуровневой архитектуры в курсовом проекте на рисунке 2.1.2

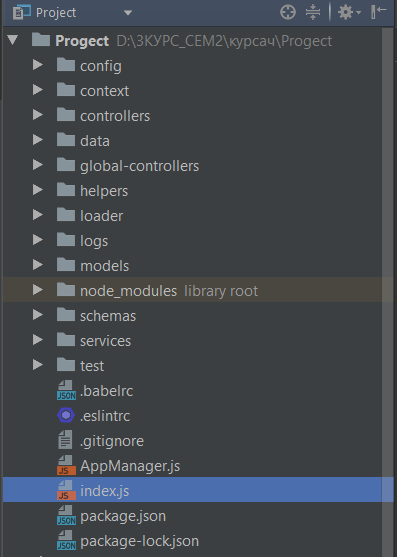


Рисунок 2.1.2 – структура проекта

## 2.2. Структура моделей и классов

Рассмотрим модели данных, используемые Sequelize для работы с БД.

Библиотека Sequelize — это Node.js ORM система для работы с такими базами данных, как PostgreSQL, MySQL, SQLite и MSSQL.

Модель пользователя представлена на рисунке 2.2.1

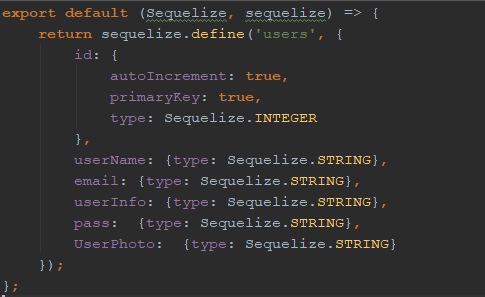


Рисунок 2.2.1 – модель User

Модель стартапа представлена на рисунке 2.2.2

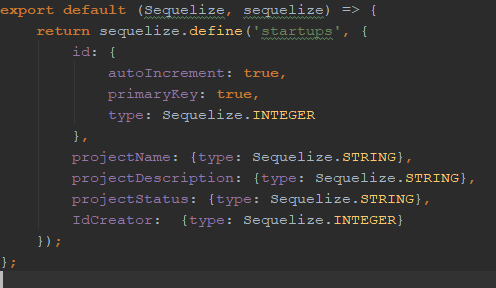


Рисунок 2.2.2 – модель Startup

Модель приглашения представлена на рисунке 2.2.3

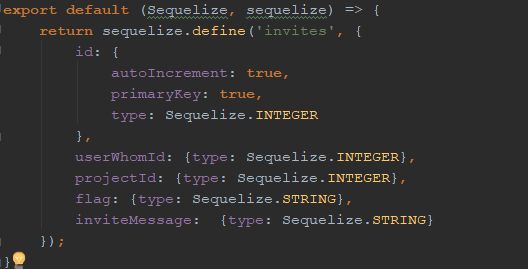


Рисунок 2.2.3 – модель Invite

Модель категории представлена на рисунке 2.2.4

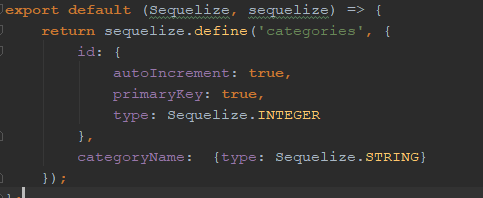


Рисунок 2.2.4 – модель Category

Для валидации моделей использовалась библиотека Joi.

Joi - язык описания схемы объектов и валидатор для объектов JavaScript. Эта библиотека позволяет создавать чертежи или схемы для объектов JavaScript (объект, который хранит информацию) для обеспечения проверки ключевой информации.

В качестве примера приведём модель валидации пользователя на рисунке 2.2.5

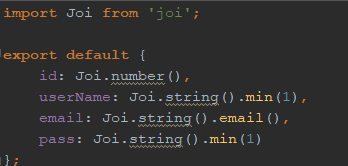


Рисунок 2.2.5 – модель валидации User

В приложении A приведен код всех моделей данных и моделей валидации.

Таким образом у нас существует набор основных операций для работы с базой. Каждый набор которых основывается на классе-конструкторе на рисунке 2.2.6

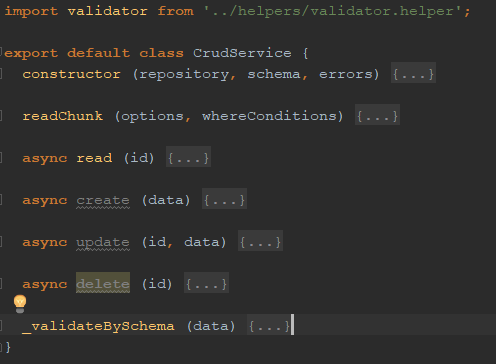


Рисунок 2.2.6 – класс crudService

В приложении B приведен код всех сервисов. Слой сервисов предназначен для уменьшения зависимостей кода, он делает расширение проекта наиболее простым и логически понятным. Так же использовался паттерн Repository.

Одним из наиболее часто используемых паттернов при работе с данными является паттерн Repository. Repository позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

# 3. Разработка функциональной модели и модели данных ПС

## 3.1. Выполняемые функции

Приложение основывается на архитектурном стиле Restful API.

REST (Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса.

Таким образом для реализации REST у нас существует набор основных операций для каждой модели данных. Каждый набор CRUD-операций основывается на классе-конструкторе на рисунке 3.1.1

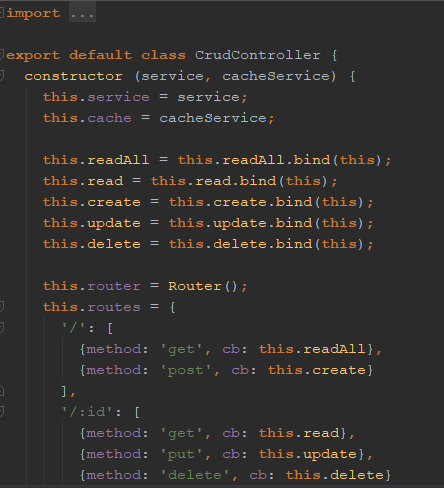


Рисунок 3.1.1 – класс конструктор

В приложении С приведен код всех контроллеров для реализации REST API.

## 3.2. Модель базы данных

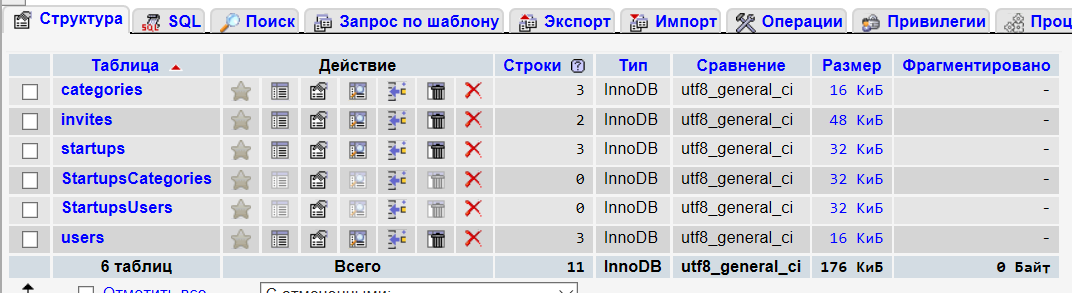
В курсовом проекте использовалась база данных MySQL. База состоит из 6 таблиц, приведенных на рисунке 3.2.1.

Рисунок 3.2.1 – таблицы БД

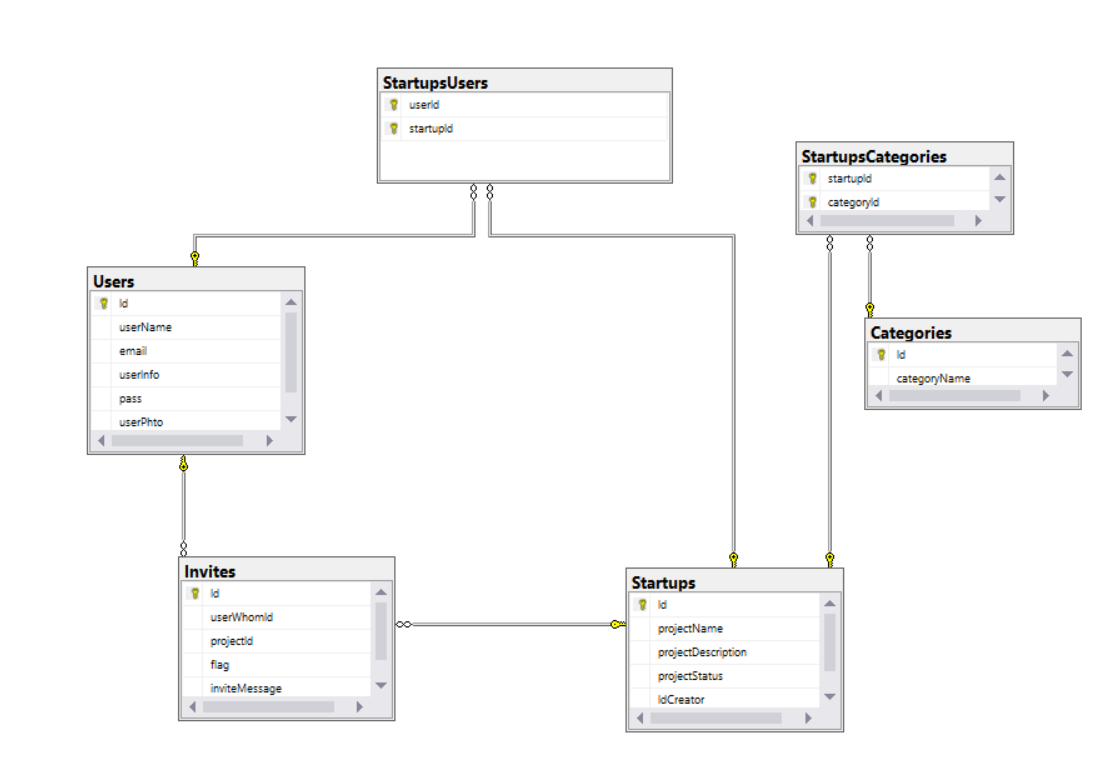
Две из используемых таблиц являются организацией связей многие ко многим. Более полно связи таблиц в базе покажем на рисунке 3.2.2.

Рисунок 3.2.2 – связи таблиц БД

Для работы с базой данных используется набор моделей, описанных с помощью Sequelize и представленных в пункте 2.2, а также набор сервисов, позволяющих выполнять действия с моделями базы, сервисы представлены в пункте 2.3

# 4. Руководство пользователя

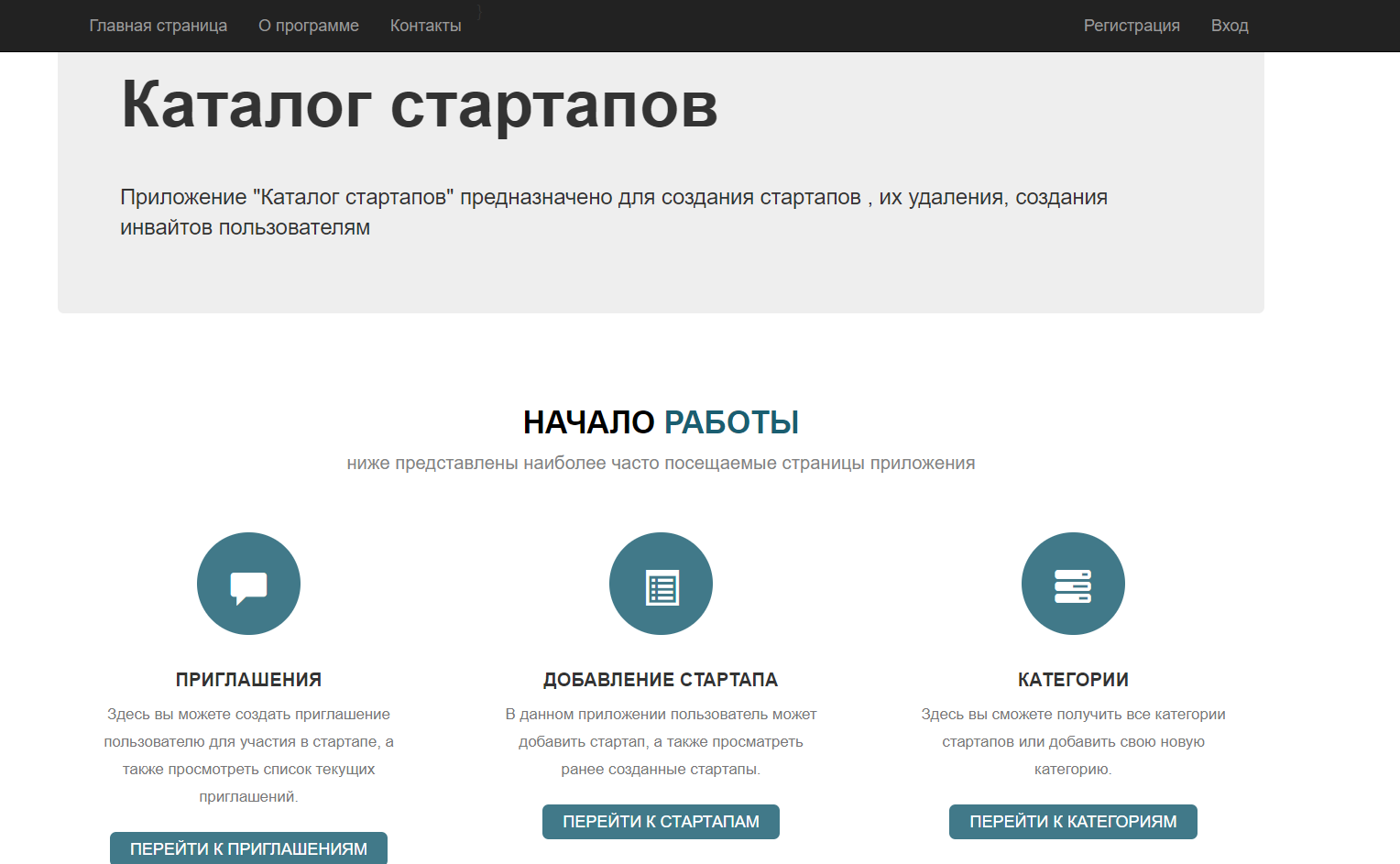
После запуска приложения пользователь видит главное окно приложения, представленное на рисунке 4.1.

Рисунок 4.1 – главная страница

Пользователь может войти в свою учетную запись, форма входа представлена на рисунке 4.2.

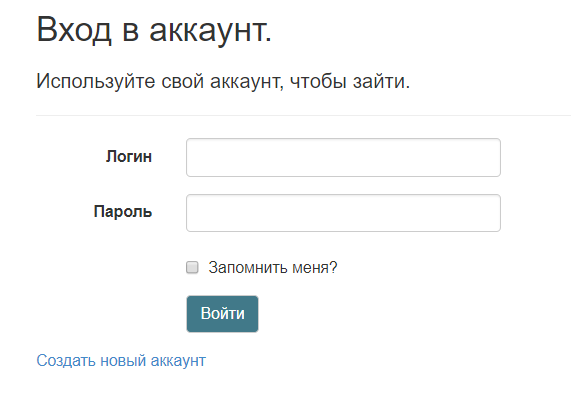
­

Рисунок 4.2 – авторизация

Если пользователь не имеет своей учетной записи, то он может зарегистрироваться, форма регистрации представлена на рисунке 4.3.

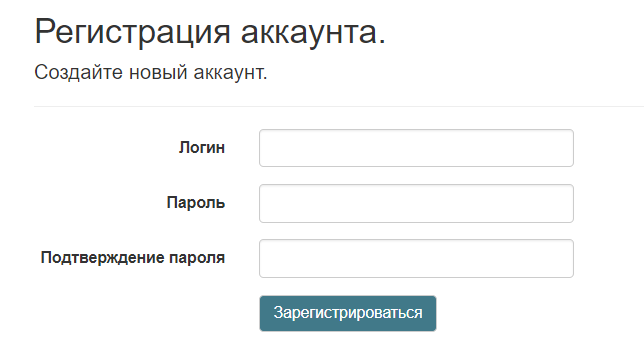
­

Рисунок 4.3 – регистрация

Пользователь может увидеть список категорий стартапов, а также добавить свою категорию рисунок 4.4.

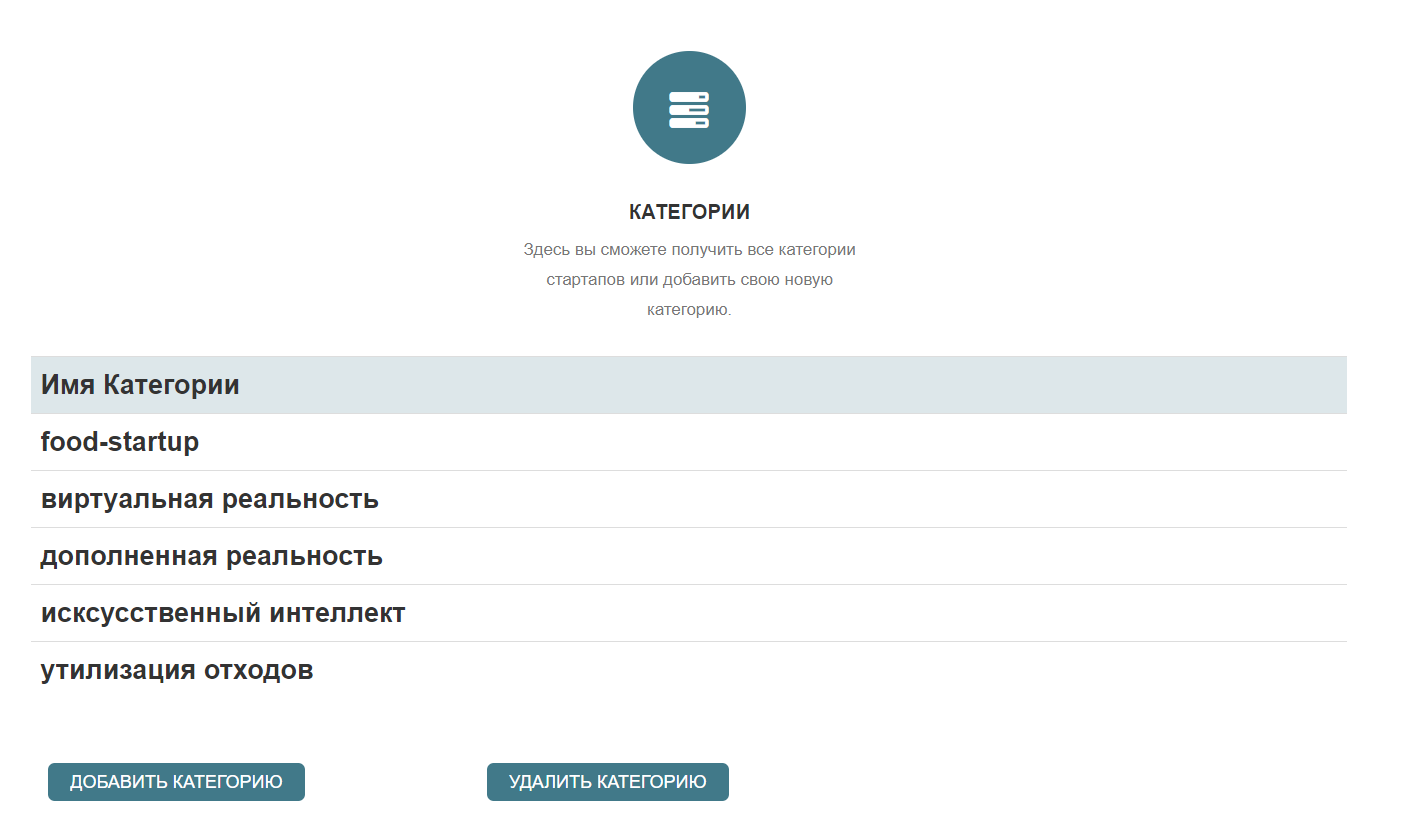


Рисунок 4.4 – работа с категориями

Пользователь может увидеть список отправленных приглашений в стартапы, а также выслать новое приглашение рисунок 4.5.

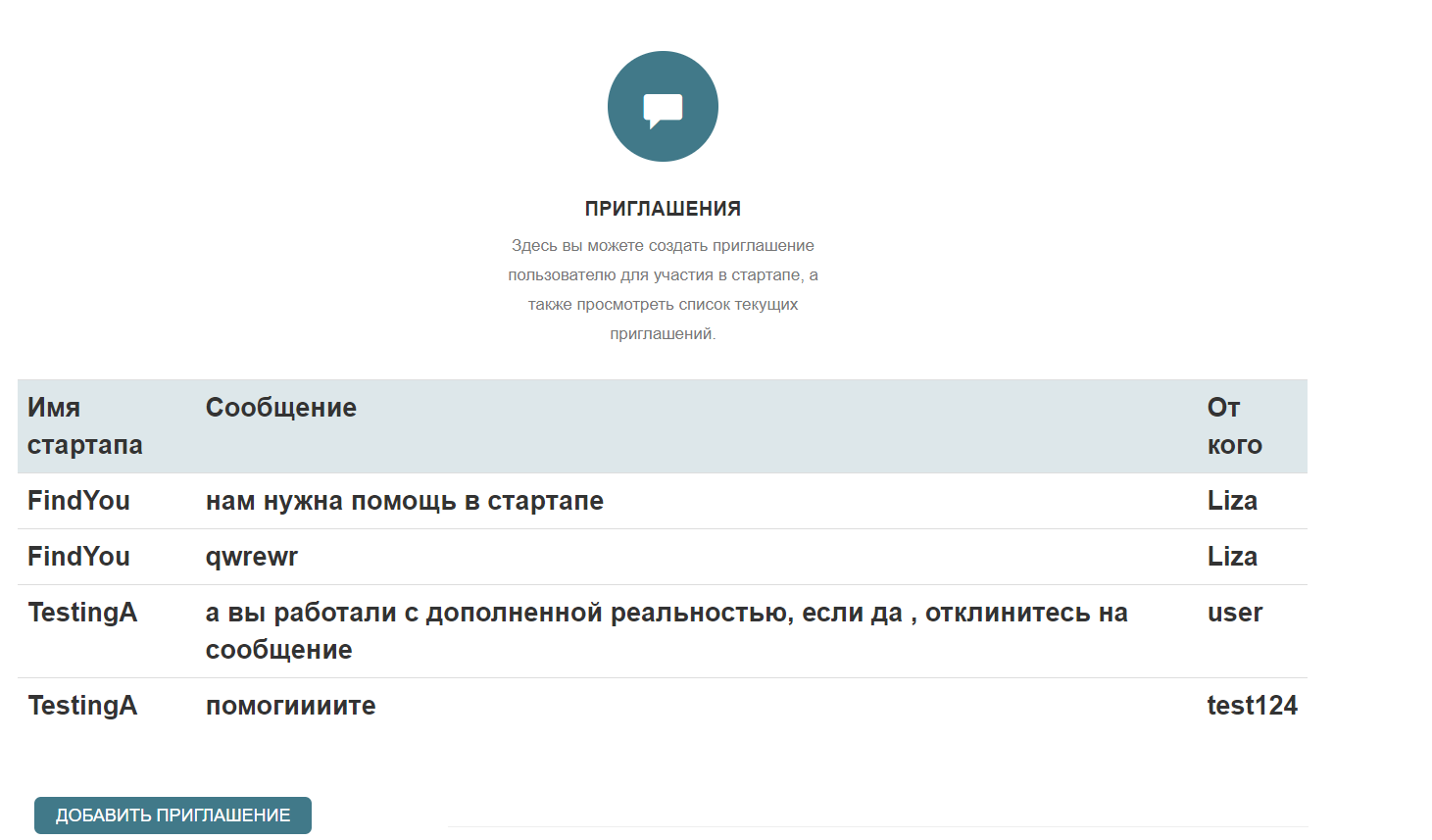


Рисунок 4.5 – список приглашений

Добавление нового приглашения представлено на рисунке 4.6.

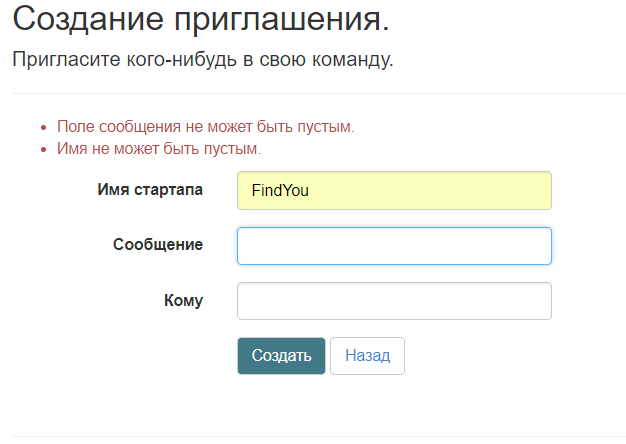
****

Рисунок 4.6 – добавления приглашения

Пользователь может увидеть список стартапов, а также создать свой стартап с уникальным именем и, желательно, содержимым рисунок 4.7.



Рисунок 4.7 – работа со стартапами

Добавление нового стартапа представлено на рисунке 4.8.

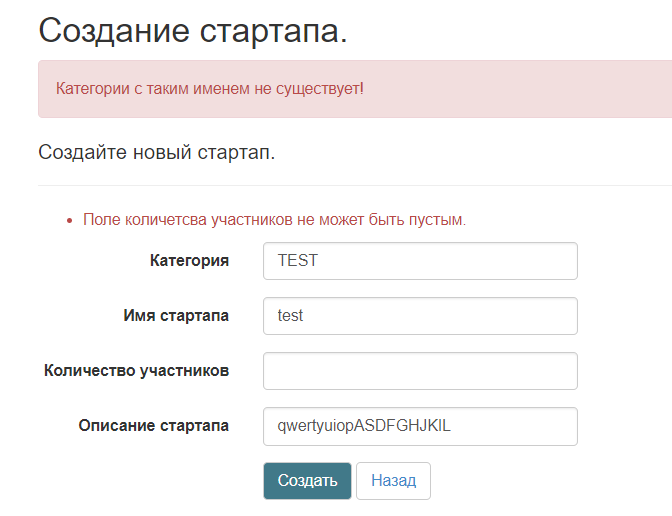
****

Рисунок 4.8 – добавления стартапа

# 5. Тестирование приложения

В рамках курсового проекта провелось как функциональное, так и автоматизированное тестирование.

## 5.1. Преимущества и недостатки функционального тестирования

**Функциональное тестирование**- процесс, основанный на заранее известном (в некоторых случая предполагаемом) поведении пользователя, основанный в первую очередь на детальном анализе и изучении функциональной спецификации приложения, системы или небольшого модуля (компонента).

Простыми словами: тестировщик ПО проводя функциональное тестирование исходя из своего личного опыта, документации к проекту, а также из общения с представителями заказчика и своей командой предполагает, как будет себя вести пользователь и на основе этого совершает те же действия, проверяя приложение.

Такое тестирование проводиться функциональными тестами, которые проектируются и создаются с помощью тест-дизайна. По другому их еще называют тест-кейсы (тестовые случаи).

В другом случае тестирование основывается на знании бизнес-процессов. При таком функциональном тестировании тестировщика интересует в целом, может ли пользователь от начала до конца пройти весь бизнес сценарий. Тестовые случаи при таком тестировании называют юз-кейсами. Сценарии использования достаточно интересная отдельная тема.

Преимущества функционального тестирования:

* Имитация реального пользователя, взгляд глазами этого пользователя;
* При правильном подходе или множестве тестировщиков, большое покрытие разнообразными функциональными тестами;

Недостатки функционального тестирования:

* велика вероятность при проверке функциональности упустить различные логические ошибки в ПО;
* вероятность избыточного тестирования.

При тестировании приложения, разработанного в рамках курсового проекта, пользователем будет вводится не валидная информация, ожидающая соответствующих исключений и предупреждений.

## 5.2. Процесс тестирования приложения курсового проекта

Тестирование входа в аккаунт на рисунке 5.2.1 при введении несуществующего пользователя или неверного пароля.

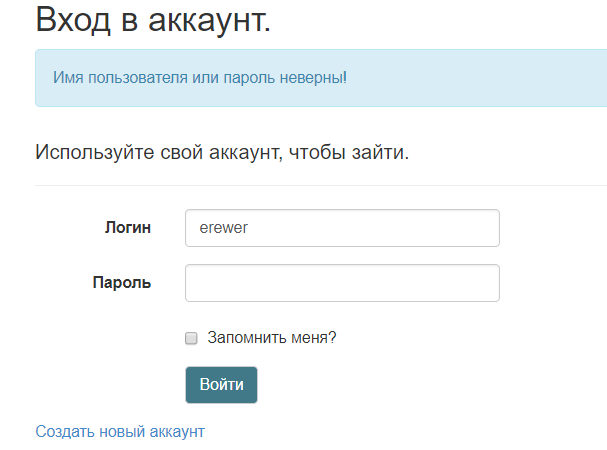


Рисунок 5.2.1 – тестирование входа

Тестирование регистрации аккаунта на рисунке 5.2.2 при введении не данных полей или же имени уже существующего пользователя.

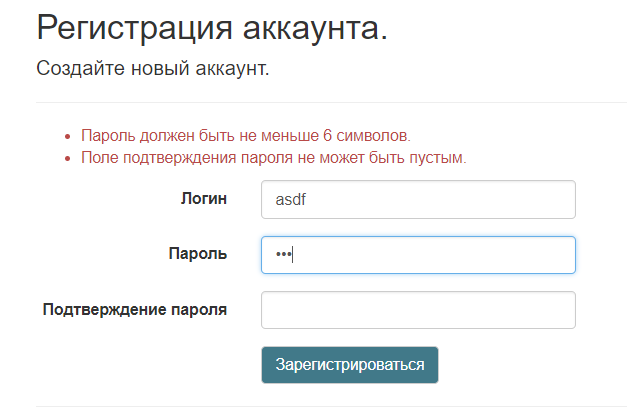


Рисунок 5.2.2 – тестирование регистрации

Тестирование удаления категории на рисунке 5.2.3 при попытке удалить категорию, к которой уже привязан стартап.

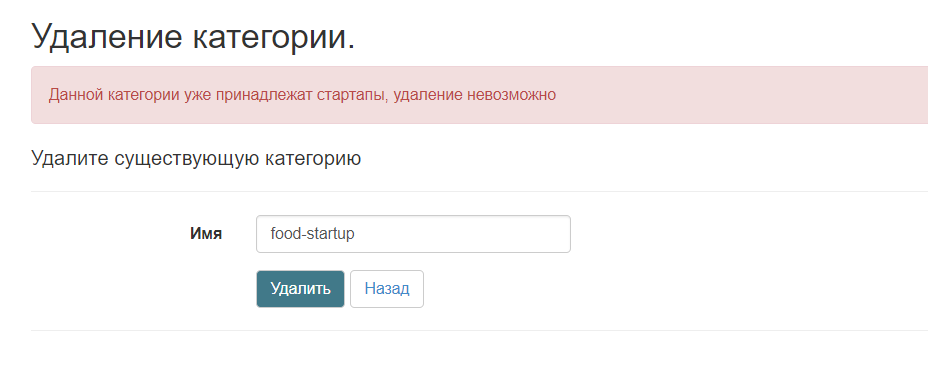


Рисунок 5.2.3 – тестирование удаления категории

Тестирование создания стартапа на рисунке 5.2.4 при попытке добавления несуществующей категории или же наличие незаполненных полей.

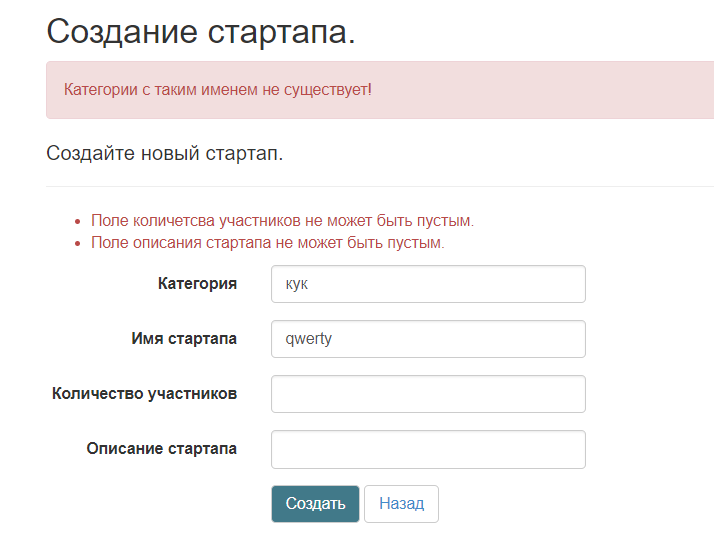


Рисунок 5.2.4 – тестирование добавления стартапа

Таким образом был протестирован почти весь функционал приложения для исключения возможных непредвиденных ошибок при работе.

## 5.3. Тестирование с использованием Unit-тестов

На основную логику проекта были разработаны и успешно выполнены unit-тесты. Unit-тесты дают уверенность, что ваша программа работает как задумано. Такие тесты можно запускать многократно. Успешное выполнение тестов покажет разработчику, что его изменения не сломали ничего, что ломать не планировалось.

Провалившийся тест позволит обнаружить, что в коде сделаны изменения, которые меняют или ломают его поведение. Исследование ошибки, которую выдает провалившийся тест, и сравнение ожидаемого результата с полученным даст возможность понять, где возникла ошибка, будь она в коде или в требованиях. На рисунке 5.3.1. и 5.3.2 представлены результаты выполненных unit-тестов.

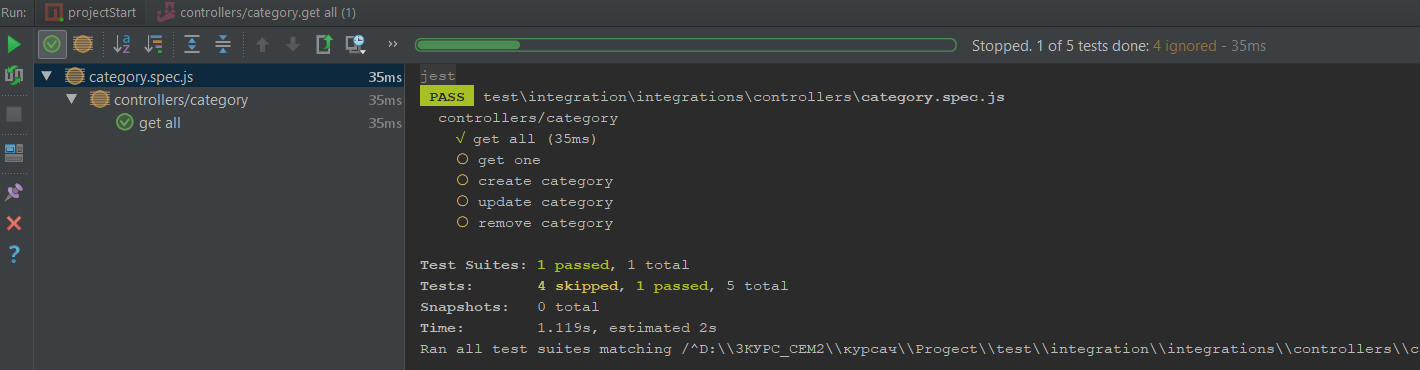




Рисунок 5.3.1 – тестирование модели Category

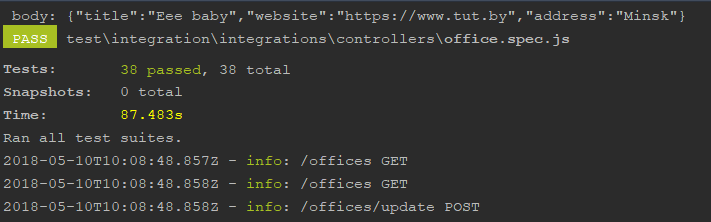


Рисунок 5.3.2 – тестирование полного проекта

# Заключение

В заключение курсовой работы необходимо сделать вывод о результатах проделанной работы. Главным выводом является то, что разработка приложения является очень трудоемким и долгим процессом, а также требует очень хорошей теоретической подготовки. При написании работы было прочитано много технической литературы, а также просмотрены примеры возможной реализации той или иной задачи.

Разработку приложения можно разделить на этапы:

* разработка функциональных возможностей приложения;
* разработка БД для хранения информации;
* разработка архитектуры;
* разработка пользовательского интерфейса;
* разработка шаблонов и стилей пользовательского интерфейса;
* написание исходного кода, выполняющего логику приложения;
* тестирование приложения;
* разработка инструкции пользователя.

Выполненные задачи:

* + сохранение и изменение информации в БД
  + просмотр информации, хранящейся в БД
  + осуществление работы с REST API
  + разработка UI для браузера

Также стоит отметить то, что при написании курсового проекта для достижения желаемых результатов, было изучено множество возможностей технологии Node.js.

Таким образом в рамках курсового проекта было разработано приложение, позволяющее работать со стартапами. Можно заметить, что данное приложение имеет и ряд недостатков, например, не до конца продуманный функционал или недостаточно полный интерфейс. Из плюсов данного приложения можно выделить логирование всех запросов пользователя, продуманную структуру базы данных.

# Список использованных источников

1. Sequelize [Электронный ресурс]: git tutorial. 2018. №1. URL: https://accetone.github.io/cwp/10/(дата обращения 12.03.2018);
2. Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с.
3. React [Электронные ресурс]: official tutorial. 2018. №1. URL: https://reactjs.org/tutorial/tutorial.html (дата обращения 02.04.2018).
4. Фримен, Адам ASP.NET MVC 3 Framework с примерами на C# для профессионалов / Адам Фримен, Стивен Сандерсон. - М.: Вильямс, 2011. - 672 c.

# ПРИЛОЖЕНИЕ А

**/models/categories.js**

export default (Sequelize, sequelize) => {

return sequelize.define('categories', {

id: {

autoIncrement: true,

primaryKey: true,

type: Sequelize.INTEGER

},

categoryName: {type: Sequelize.STRING}

});

};

**/models/invites.js**

export default (Sequelize, sequelize) => {

return sequelize.define('invites', {

id: {

autoIncrement: true,

primaryKey: true,

type: Sequelize.INTEGER

},

userWhomId: {type: Sequelize.INTEGER},

projectId: {type: Sequelize.INTEGER},

flag: {type: Sequelize.STRING},

inviteMessage: {type: Sequelize.STRING}

});

};

**/models/startups.js**

export default (Sequelize, sequelize) => {

return sequelize.define('startups', {

id: {

autoIncrement: true,

primaryKey: true,

type: Sequelize.INTEGER

},

projectName: {type: Sequelize.STRING},

projectDescription: {type: Sequelize.STRING},

projectStatus: {type: Sequelize.STRING},

IdCreator: {type: Sequelize.INTEGER}

});

};

**/models/users.js**

export default (Sequelize, sequelize) => {

return sequelize.define('users', {

id: {

autoIncrement: true,

primaryKey: true,

type: Sequelize.INTEGER

},

userName: {type: Sequelize.STRING},

email: {type: Sequelize.STRING},

userInfo: {type: Sequelize.STRING},

pass: {type: Sequelize.STRING},

UserPhoto: {type: Sequelize.STRING}

});

};

**/schema/user.js**

import Joi from 'joi';

export default {

id: Joi.number(),

userName: Joi.string().min(1),

email: Joi.string().email(),

pass: Joi.string().min(1)

};

**/schema/startup.js**

import Joi from 'joi';

export default {

id: Joi.number(),

projectName: Joi.string().min(3),

projectDescription: Joi.string().min(10),

projectStatus: Joi.string().min(3),

IdCreator: Joi.number()

};

**/schema/invite.js**

import Joi from 'joi';

export default {

id: Joi.number(),

userWhomId: Joi.number(),

projectId: Joi.number(),

flag: Joi.string().min(3),

inviteMessage: Joi.string().min(7)

};

# ПРИЛОЖЕНИЕ Б

**/services/crud.service.js**

import validator from '../helpers/validator.helper';

export default class CrudService {

constructor (repository, schema, errors) {

this.repository = repository;

this.errors = errors;

this.schema = schema;

this.defaults = {

readChunk: {

limit: 5,

offset: 0,

sortOrder: 'asc',

sortField: 'id'

}

};

}

readChunk (options, whereConditions) {

options = {

...this.defaults.readChunk,

...options

};

let limit = Number(options.limit) || this.defaults.readChunk.limit;

let offset = Number(options.offset) || this.defaults.readChunk.offset;

return this.repository.findAll({

where: {...whereConditions},

limit,

offset,

order: [[options.sortField, options.sortOrder.toUpperCase()]],

raw: true

});

}

async read (id) {

id = parseInt(id);

if (isNaN(id)) {

throw this.errors.invalidId;

}

const item = await this.repository.findById(id, {raw: true});

if (!item) {

throw this.errors.notFound;

}

return item;

}

async create (data) {

this.\_validateBySchema(data);

const item = await this.repository.create(data);

return item.get({plain: true});

}

async update (id, data) {

this.\_validateBySchema(data);

await this.repository.update(data, {where: {id: id}, limit: 1});

return this.read(id);

}

async delete (id) {

return this.repository.destroy({where: {id: id}});

}

\_validateBySchema (data) {

let validCheck = validator(this.schema, data);

if (!validCheck.isValid) {

throw this.errors.validError(validCheck.errors);

}

}

}

# ПРИЛОЖЕНИЕ В

**/controllers/route/crud.controller.js**

import {Router} from 'express';

import wrap from '../../helpers/wrap.helper';

import {send} from '../../helpers/sender.helper';

export default class CrudController {

constructor (service, cacheService) {

this.service = service;

this.cache = cacheService;

this.readAll = this.readAll.bind(this);

this.read = this.read.bind(this);

this.create = this.create.bind(this);

this.update = this.update.bind(this);

this.delete = this.delete.bind(this);

this.router = Router();

this.routes = {

'/': [

{method: 'get', cb: this.readAll},

{method: 'post', cb: this.create}

],

'/:id': [

{method: 'get', cb: this.read},

{method: 'put', cb: this.update},

{method: 'delete', cb: this.delete}

]

};

}

async readAll (req, res) {

let data = await this.service.readChunk(req.query);

this.cache.set(req, data);

send(req, res, data);

}

async read (req, res) {

let data = await this.service.read(req.params.id);

this.cache.set(req, data);

send(req, res, data);

}

async create (req, res) {

send(req, res, await this.service.create(req.body));

}

async update (req, res) {

send(req, res, await this.service.update(req.params.id, req.body));

}

async delete (req, res) {

send(req, res, await this.service.delete(req.params.id));

}

registerRoutes () {

Object.keys(this.routes).forEach(route => {

let handlers = this.routes[route];

if (!handlers || !Array.isArray(handlers)) return;

for (let handler of handlers) {

this.router[handler.method](route, wrap(handler.cb));

}

});

}

}

**/controllers/route/startup.controller.js**

import CrudController from './crud.controller';

import {send} from '../../helpers/sender.helper';

export default class StartupController extends CrudController {

constructor ({startupService, cacheService}) {

super(startupService, cacheService);

this.addUser = this.addUser.bind(this);

this.removeUser = this.removeUser.bind(this);

this.routes['/:id/Users'] = [

{method: 'put', cb: this.addUsers},

{method: 'delete', cb: this.removeUser}

];

this.registerRoutes();

}

async addUser(req, res) {

await this.service.addUser(req.body.userId, req.params.id);

send(req, res, {success: true});

}

async removeUser (req, res) {

await this.service.removeUser(req.body.userId, req.params.id);

send(req, res, {success: true});

}

};

**/controllers/route/user.controller.js**

export default class UserController extends CrudController {

constructor ({usersService, cacheService}) {

super(usersService, cacheService);

this.FindUser = this.FindUser.bind(this);

this.createStartup = this.createStartup.bind(this);

this.routes['/FindUser'] = [

{method: 'post', cb: this.FindUser}

];

this.routes['/:id/createStartup'] = [

{method: 'get', cb: this.createStartup}

];

this.registerRoutes();

}

async FindUser (req, res) {

const item = await this.service.FindUser(

{

userName: req.body.userName,

pass: req.body.pass

});

send(req, res, item);

}

async createStartup (req, res) {

const items = await this.service.createStartup(

{

id: req.params.id

});

send(req, res, items);

}

};

# ПРИЛОЖЕНИЕ Г

**/src/Repository/startup.js**

import axios from 'axios';

import { url } from '../Helpers/constants';

let config = {

headers: {

'Content-Type': 'application/json',

}

}

export default class StartupRepository {

createStartup(startup) {

let newStartup = JSON.stringify({

projectName: startup.projectName,

projectDescription: startup.description,

projectStatus : "new",

IdCreator: 1

});

return axios.post(`${url}/api/startup`, newStartup, config);

}

updateStartup(projectName, startup) {

let newStartup = {

id: startup.id,

projectName: projectName,

projectDescription: startup.projectDescription,

projectStatus: startup.projectStatus,

IdCreator: startup.IdCreator

};

axios.put(`${url}/api/startup/`, JSON.stringify(newStartup), config).then(response => {

alert('Успешно сохранено!');

}).catch(e => {

console.log(e);

});

}

getStartup = (id) => {

return axios.get(`${url}/api/startup/${id}`, config);

}

getAllStartups = () => {

return axios.get(`${url}/api/startup/`, config);

}

deleteStartup(startup) {

axios.delete(`${url}/api/startup/${startup.id}`, config)

.then(response => {

alert('Успешно удалено!');

})

.catch(e => {

console.log(e.message);

})

}

}

import axios from 'axios';

import { url } from '../Helpers/constants';

let config = {

headers: {

'Content-Type': 'application/json',

}

}

**/src/Repository/user.js**

export default class UserRepository {

createUser(user) {

let newUser = JSON.stringify({

userName: user.username,

email: user.email,

pass: user.password,

});

return axios.post(`${url}/api/user`, newUser, config);

}

findUser = (user) => {

let newUser = JSON.stringify({

userName: user.username,

pass: user.password,

});

return axios.post(`${url}/api/user/FindUser`, newUser, config);

}

authentificateByToken = (token) => {

config.headers['Authentification'] = token

return axios.get(`${url}/api/User/AuthentificateByToken`, config);

}

}