

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

Кафедра Систем Управления и Информатики Группа Р3255

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе

Разработка интернет сервиса

для помощи бездомным животным

Автор курсовой работы Федюкович С. А. (подпись)
(фамилия, и. о.)

Руководитель Платунова С. М. (подпись)
(фамилия, и. о.)

“__” _____ 20__ г. Санкт-Петербург, 20__г.

Курсовая работа выполнена с оценкой _____

Дата защиты “__” _____ 20__ г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент _____ Федюкович Семён Андреевич
(Фамилия, И. О.)

Факультет _____ Программной инженерии и компьютерной техники

Кафедра _____ Аппаратно-программных комплексов вычислительной техники _____ Группа _____ Р3255

Дисциплина _____ Программирование интернет приложений

Наименование темы _____ Разработка интернет сервиса для помощи бездомным животным

Задание _____ Спроектировать и разработать серверную часть интернет приложения для помощи бездомным животным, позволяющее как оказать помощь приютам для бездомных животных поблизости, так и разместить объявление о просьбе

Краткие методические указания _____ Node.js — это программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения. MongoDB — это документоориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Содержание пояснительной записки _____ Обоснование выбранных технологий, краткое описание главной логики проекта с примера и полный листинг кода всего приложения.

Рекомендуемая литература _____ Node.js в действии — Кантелон Майк, Хартер Марк, Головайчук TJ, Райлих Натан; Выразительный JavaScript — Marijn Haverbeke; Компьютерные сети: Принципы, технологии, протоколы : Учебник — Виктор Олифер.

Руководитель _____
(Подпись, дата)

Студент _____
(Подпись, дата)

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”**

**ОТЗЫВ
РУКОВОДИТЕЛЯ
О ВЫПОЛНЕНИИ КУРСОВОГО ПРОЕКТА (РАБОТЫ)**

Студент Федюкович Семён Андреевич
(Фамилия, И. О.)
Факультет Программной инженерии и компьютерной техники
Кафедра Аппаратно-программных комплексов вычислительной техники Группа РЗ255
Дисциплина Программирование интернет приложений
Наименование темы Разработка интернет сервиса для помощи бездомным животным

№	Показатели	Оценка			
		5	4	3	0*
1.	Способность к работе с литературными источниками, справочной литературой, Интернет-ресурсами и т. п.				
2.	Использование иностранных источников				
3.	Способность к анализу и обобщению информационного материала				
4.	Владение базовыми знаниями в профессиональной области				
5.	Владение базовыми знаниями в смежных областях				
6.	Владение навыками решения технических задач				
7.	Способность применять знания на практике				
8.	Уровень и корректность использования в работе методов численного моделирования, инженерных расчетов и статистической обработки данных				
9.	Владение навыками использования современных пакетов компьютерных программ и технологий				
10.	Владение навыками оформления отчетных материалов с применением современных пакетов программ				
11.	Качество оформления пояснительной записки (общий уровень грамотности, стиль изложения, качество иллюстраций, корректность цитирования и пр.**)				
12.	Качество оформления презентации				
13.	Владение навыками публичного выступления и межперсональной коммуникации				
14.	Владение навыками планирования и управления временем при выполнении работы				
Итоговая оценка					

* — не оценивается (трудно оценить)

** согласно рекомендациям

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”**

АННОТАЦИЯ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент _____ Федюкович Семён Андреевич
(Фамилия, И. О.)
Факультет _____ Программной инженерии и компьютерной техники
Кафедра _____ Аппаратно-программных комплексов вычислительной техники _____ Группа _____ Р3255
Дисциплина _____ Программирование интернет приложений
Наименование темы _____ Разработка интернет сервиса для помощи бездомным животным

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

1. Цель и задачи работы

Предложены студентом

Цель будущего продукта — эффективное вовлечение людей в помощь бездомным животным. У приютов есть различные задачи, которые может выполнить любой желающий: погулять с собакой, помочь с перевозкой корма, удалённо разместить объявление в сети и прочее. Заказчиком выступает Фонд помощи животным «РЭЙ», который помогает бездомным животным, поддерживая более 30 приютов Москвы, Подмосковья и близлежащих областей. В общей сложности там проживают около 15 000 собак и кошек.

2. Характер работы

Моделирование

3. Содержание работы

Общие сведения; Функциональные требования к разрабатываемой системе; Структура проекта; Архитектура; Технологии; Расчетно-графическая часть; Листинг кода проекта, Экспериментальная часть; Заключение.

4. Выводы

При разработке интернет приложений немаловажную роль в достижении успеха играет выбор правильных технологий, удовлетворяющих специфике поставленной задачи.

Руководитель _____
(Подпись)

Студент _____
(Подпись)

“ ____ ” _____ 20__ г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

ГРАФИК ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент _____ Федюкович Семён Андреевич _____
(Фамилия, И. О.)
Факультет _____ Программной инженерии и компьютерной техники _____
Кафедра _____ Аппаратно-программных комплексов вычислительной техники _____ Группа _____ Р3255 _____
Дисциплина _____ Программирование интернет приложений _____
Наименование темы _____ Разработка интернет сервиса для помощи бездомным животным _____

№	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1	Описание полного функционала приложения	10.11.18	10.11.18	
2	Составление схема базы данных	12.11.18	12.11.18	
3	Реализация основных методов сервиса	14.11.18	15.11.18	
4	Тестирование и поиск ошибок	16.11.18	16.11.18	
5	Оформление проекта	20.12.18	25.12.18	

Руководитель _____
(Подпись, дата)

Студент _____
(Подпись, дата)

Содержание

Введение.....	7
Структура проекта.....	11
Листинг кода проекта.....	20
Экспериментальная часть.....	70
Заключение.....	71
Список литературы.....	72

Введение

Общие сведения

В рамках данной курсовой работы мной была спроектирована и разработана серверная часть интернет приложения для помощи бездомным животным.

На момент написания этой пояснительной записки приложение находится в стадии активной разработки и ещё не готово к релизу. Здесь будет подробно описан уже реализованный функционал, а именно та его часть, над которой велась работа непосредственно мной, а описание же функционала, который пока только в планах приводится только поверхностно для полноты картины. Конечной целью проекта является кроссплатформенное приложение, позволяющее как оказать помощь приютам для бездомных животных поблизости, так и разместить объявление о просьбе (когда используется приют, а не волонтером).

Процесс взаимодействия с приложением предельно прозрачен и прост: у волонтера будет возможность найти задачу исходя из своих ресурсов. Если у человека нет времени он может оказать помощь финансово, или же наоборот — делом. В результате успешного выполнения задач рейтинг волонтера будет подниматься, а самые активные и неравнодушные попадут на доску почета и получают бейджи.

Для удобного поиска задач планируется реализованная фильтрация по местоположению и времени. Более того, каждая компания может привнести свой вклад в развитие проекта, предоставив небольшие бонусы для волонтеров. Для удобного общения приюта и волонтера в приложение будет встроен чат. Если волонтер вдруг забудет о запланированной задаче, приложение уведомит его подходящим ему способом: pushуведомления, сообщение на почту, интеграция с календарем. В случае если исполнитель регулярно не выполняет задачи от приюта, на которые он подписался, он рискует получить временную блокировку на взятие задач. Если у волонтера есть свободное время, то он может отметить свое местоположение и интервал времени для того, чтобы приложение автоматически подобрало для него подходящие задачи.

Цель будущего продукта — эффективное вовлечение людей в помощь бездомным животным. У приютов есть различные задачи, которые может выполнить любой желающий: погулять с собакой, помочь с перевозкой корма, удалённо разместить объявление в сети и прочее.

Заказчиком выступает Фонд помощи животным «РЭЙ», который помогает бездомным животным, поддерживая более 30 приютов Москвы, Подмосковья и близлежащих областей. В общей сложности там проживают около 15 000 собак и кошек.

Функциональные требования к разрабатываемой системе

1. Возможность регистрироваться в системе как приют и как исполнитель (волонтер)
2. У исполнителя должна быть возможность просматривать списки задач: выполненные, будущие и в работе
3. Задача может быть отменена 5
4. Возможность размещать задачи от лица приюта
5. У каждого приюта должен быть физический адрес, для фильтрации по местоположению
6. У каждой задачи должна быть информация: Название, описание, временные рамки, статус, графическое изображение (фотография), данные о животном
7. Карточка задачи должна отражать текущий статус задачи (в работе, выполнена, свободна)
8. Рейтинг исполнителя (достижения: хорошие и плохие)
9. Возможность для приюта оставлять отзывы об исполнителе
10. Временная блокировка за отмену задачи исполнителем (Если исполнитель взял задачу, но впоследствии отменил её, исполнитель будет заблокирован на какое-то время и брать новые задачи для него не будет возможности)
11. Возможность перевода денежных средств на счет приюта (пожертвований)
12. Клиент должен работать на смартфонах и на настольном компьютере (в Интернет браузере)
13. Возможность подавать заявку на то, чтобы забрать животное из приюта домой
14. Возможность для приютов размещать объявления на отдачу питомца в хорошие руки
15. Возможность для приютов скрывать/показывать задачи для публичного просмотра(черновики)
16. Список с рейтингом исполнителей, доска почета, работники месяца
17. Возможность для приютов публиковать задачи по расписанию автоматически (отложенная публикация в заданный час)
18. Возможность для исполнителя указывать свою доступность и готовность помочь в определенное время в определенном месте
19. Ожидается что задача может иметь сложную структуру, т.е. может включать в себя несколько подзадач.
20. Должны работать уведомления приюту и исполнителю об изменении статуса задачи
21. Возможность коммуникации

Архитектура

Верхнеуровневая характеристика приложения: Классическая клиент-серверная архитектура. Коммуникация между клиентом и сервером осуществляется посредством HTTP запросов, инициированных клиентом. Сервер представляет из себя REST API, написанный на Node.js. В качестве постоянного хранилища данных используется нереляционная база данных MongoDB.

Технологии

Здесь даётся краткое описание используемых терминов, технологий и шаблонов проектирования, которые в той или иной степени находят применения в разрабатываемом приложении.

Основную работу на сервере выполняет код, написанный на JavaScript, выполняемый в Node.js. Node.js - это программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и macOS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийноориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Для создания сервера на Node.js в рассматриваемом приложении был использован фреймворк Express.js, выпускаемый под лицензией свободного программного обеспечения с открытым исходным кодом. Express предоставляет все необходимые средства для разработки веб-сервера (маршрутизация, middleware и т.д.). С помощью express мы разворачиваем на сервере REST API или Программный Интерфейс Приложения.

Данные сохраняются в MongoDB. MongoDB — это документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++.

Клиентская часть реализована на JavaScript. JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией языка ECMAScript (стандарт ECMA-262). 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.

Для программирования интерфейса используется библиотека ReactJS. Использование ReactJS обусловлено значительными преимуществами этой библиотеки перед конкурентами (быстродействие, простота, выразительность, приобретаемая за счет специального JSX синтаксиса, позволяющего включать XML-подобную разметку прямо в JavaScript файлы). Кроме того, в дальнейшем планируется портирование приложения на смартфоны, для чего предполагается использование технологии React Native - фреймворка для разработки кроссплатформенных приложений для iOS и Android. Это позволит переиспользовать большую часть кодовой базы приложения с минимальными доработками.

Клиентская часть для работы в браузере настольного ПК или телефона реализована по

принципу Single Page Application – сокращенно SPA, что в переводе на русский язык означает “Приложение одной страницы”. Другими словами SPA – это web-приложение, размещенное на одной web-странице, которая для обеспечения работы загружает весь необходимый код вместе с загрузкой самой страницы. Эта модель отличается от классического интернет приложения, где каждый запрос на сервер требует перезагрузки всей страницы. В SPA не происходит перезагрузка страницы во время работы приложения, а все запросы осуществляются асинхронно посредством технологии AJAX. AJAX (аббревиатура от «Asynchronous Javascript And Xml») – технология обращения к серверу без перезагрузки страницы.

Засчет этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает привычное оконное приложение для настольного ПК.

Расчетно-графическая часть

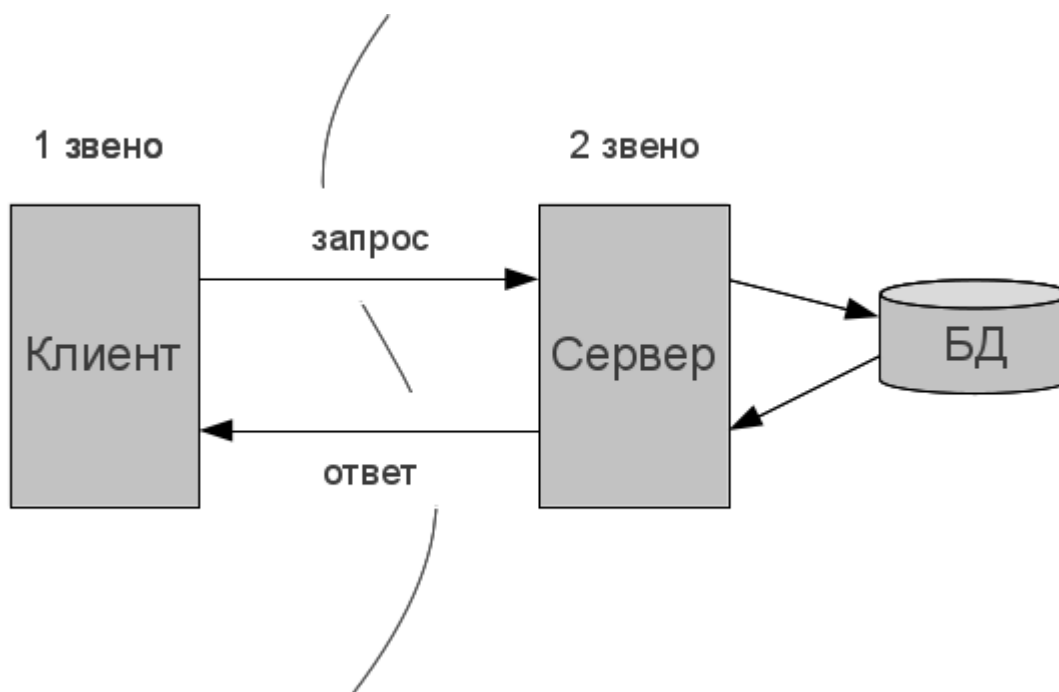


Рис. 1: Схематичное представление клиент-серверного интернет-приложения

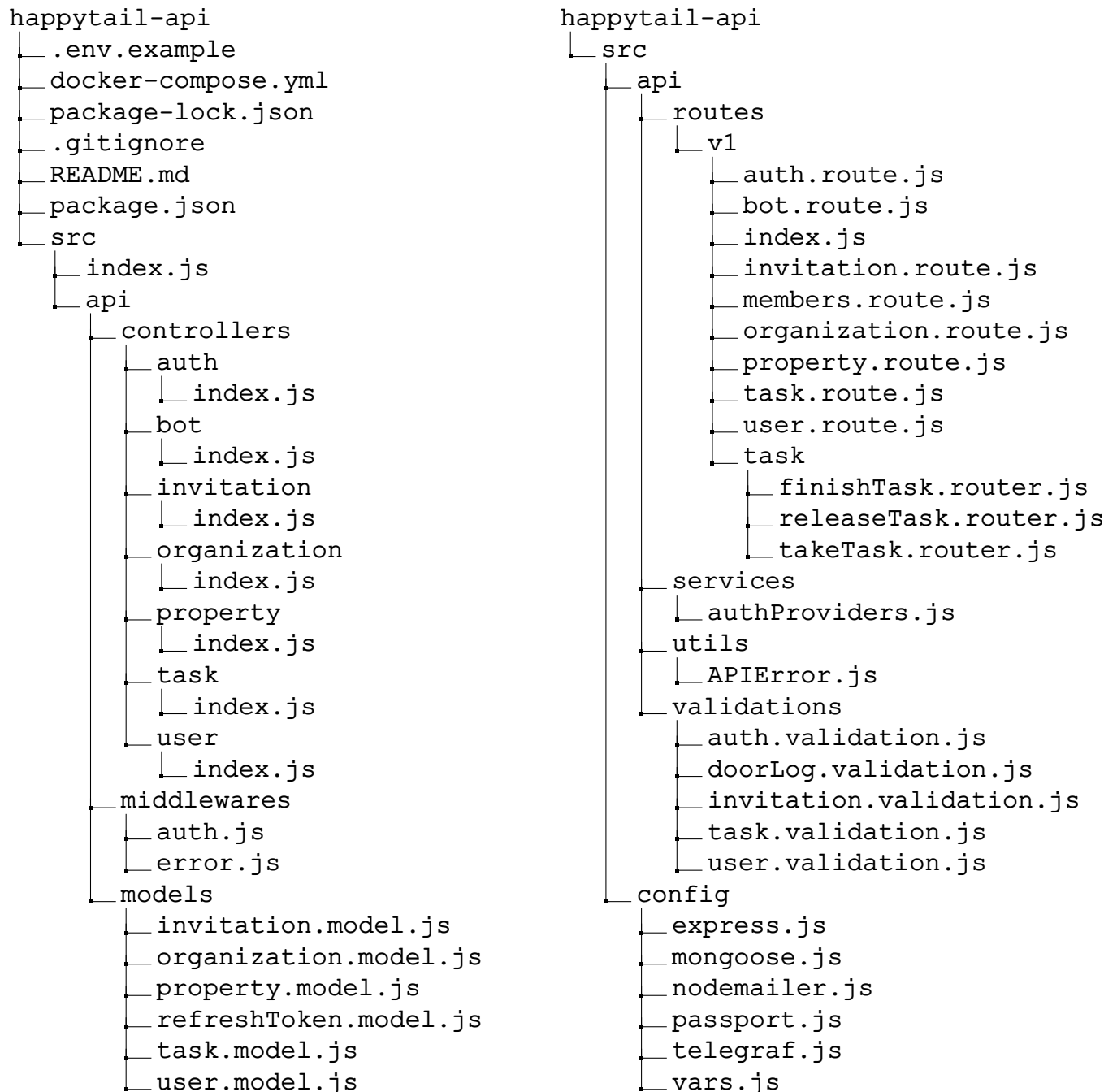


Рис. 2: Схематичное представление большинства современных интернет приложений

Верхний слой — Пользовательский интерфейс, выполняемый в браузере пользователя. Два нижних слоя на Рисунке 2 — Бизнес-логика и Уровень доступа к данным представляют собой собственно серверную часть приложения, реализованную на Node.js и выполняемую на сервере.

Разные части приложения разрабатываются независимо одна от другой и каждая имеет собственный репозиторий для исходных кодов на платформе GitHub: у базы данных свой репозиторий, у серверной части — свой, у клиентской части — свой. Репозиторий с клиентским кодом называется web-client и находится по адресу <https://github.com/redudam/happytail-api.git>. Далее подробно будет рассмотрена реализация серверной части приложения.

Структура проекта



Выше приведено дерево директорий проекта happytail-api. Пройдемся по ключевым директориям. Нас будет интересовать прежде всего директория src поскольку именно в ней располагается исходный код.

Файл src/index.js является входной точкой приложения.

```
/**
 * @module index
 */
'use strict';
Promise = require('bluebird');

const app = require('./config/express');
const {port} = require('./config/vars');
const mongoose = require('./config/mongoose');

mongoose.connect();

app.listen(port, () => console.info(`Server has been started on ${port} port`));
/**
 * Exports express
 * @public
 */
module.exports = app;
```

В данном файле осуществляется подключение модулей для работы с асинхронностью — Bluebird; с сервером — Express; с БД MongoDB — Mongoose. Также загружается модуль, содержащий основные константы — vars:

```
/**
 * @module vars
 */
'use strict';
const path = require('path');

// import .env variables
require('dotenv-safe').load({
  path: path.join(__dirname, '../../.env'),
  sample: path.join(__dirname, '../../.env.example'),
});

module.exports = {
  url: process.env.URL,
  env: process.env.NODE_ENV,
  user: process.env.USER,
  password: process.env.PASSWORD,
  port: process.env.PORT || 3000,
  jwtSecret: process.env.JWT_SECRET,
  jwtExpirationInterval: process.env.JWT_EXPIRATION_MINUTES,
  token: process.env.BOT_TOKEN,
  mongo: {
    uri: process.env.MONGO_URI || 'mongodb://localhost:27017/alarm'
  }
};
```

После производится подключение к базе данных, логика которого описана в данном модуле:

```
/**
 * @module mongoose
 */
'use strict';

const mongoose = require('mongoose');
const {mongo, env} = require('./vars');

// Exit application on error
mongoose.connection.on('error', (err) => {
  console.error(`MongoDB connection error: ${err}`);
  process.exit(-1);
});

mongoose.connection.on('connected', () => {
  console.debug('connected');
});
mongoose.connection.on('disconnected', () => {
  console.debug('disconnected');
});

process.on('SIGINT', function() {
  mongoose.connection.close(function () {
    console.log('Mongoose default connection disconnected through app termination');
    process.exit(0);
  });
});

// print mongoose logs in dev env
if (env === 'development') {
  mongoose.set('debug', true);
}

/**
 * Connect to mongo db
 *
 * @returns {object} Mongoose connection
 * @public
 */
exports.connect = () => {
  mongoose.connect(mongo.uri, {
    reconnectTries: Number.MAX_VALUE, // Never stop trying to reconnect
    reconnectInterval: 500, // Reconnect every 500ms
    keepAlive: 1,
  });
  return mongoose.connection;
};
```

Для удобной отладки работы подключения к БД были обработаны события, происходящие во время подключения, отключения, появления ошибок. Также была добавлена возможность включения и выключения вывода сообщений в консоль. На рисунке 3 представлен схематический рисунок иерархии БД

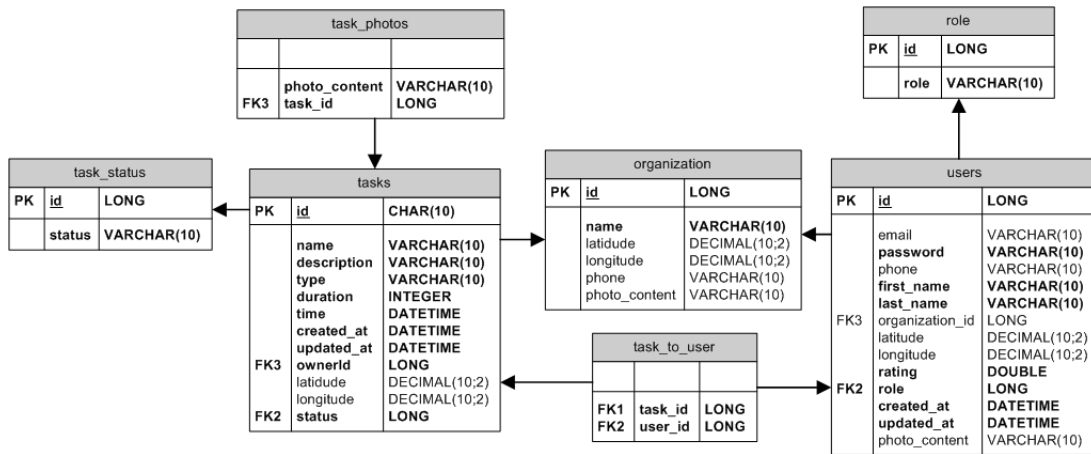


Рис. 3: Схема базы данных

Логика всего сервера основывается на раздлении всех методов на определенные группы, каждой из которых соответствует свой контроллер. Он представляет отдельный Node.js модуль, экспортирующий определенную функцию под конкретный метод. Далее в специальном роутинг файле описываются все методы контроллера, их URL адреса и какую необходимо вызвать функцию у контроллера. Далее рассмотрим конфигурации сервера и конкретные примеры контроллера и роутинг файла:

```

/**
 * @module express
 */
'use strict';

const express = require('express');
const bodyParser = require('body-parser');
const passport = require('passport');
const cors = require('cors');
const helmet = require('helmet');

const routes = require('../api/routes/v1');
const error = require('../api/middlewares/error');
const strategies = require('../passport');
const {token} = require('../config/vars');

/**Express instance
 * @public
 */
const app = express();

// parse body params and attache them to req.body
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// secure apps by setting various HTTP headers
app.use(helmet());

// enable CORS - Cross Origin Resource Sharing
app.use(cors());

// enable authentication
app.use(passport.initialize());
passport.use('jwt', strategies.jwt);

app.use('/v1', routes);
  
```

```
// if error is not an instanceof APIError, convert it.
app.use(error.converter);

// catch 404 and forward to error handler
app.use(error.notFound);

// error handler, send stacktrace only during development
app.use(error.handler);

module.exports = app;
```

Здесь были подключены следующие модули Helmet и CORS — для защиты от некоторых широко известных веб-уязвимостей путем соответствующей настройки заголовков HTTP. Также был задан общий роутинг API, и был подключен модуль для контроля доступов пользователей — JWT (JSON Web Token). Контроль осуществляется путём генерации двух токенов (ключей), аутентифицированному пользователю: access token и refresh token. Первый предназначен для получения доступа к ресурсам приложения и действует только определенное время, а второй для получения нового access токена по истечению его действия. Логика регистрации и выдачи токенов представлена в файле `happytail-api/src/api/controllers/auth/index.js`:

```
/**
 * @module index
 */
'use strict';

const httpStatus = require('http-status');
const User = require('../../../models/user.model');
const Organization = require('../../../models/organization.model');
const RefreshToken = require('../../../models/refreshToken.model');
const Invitation = require('../../../models/invitation.model');
const moment = require('moment-timezone');
const { jwtExpirationInterval } = require('../../../config/vars');

/**
 * Returns a formatted object with tokens
 * @private
 */
function generateTokenResponse(user, accessToken) {
  const tokenType = 'Bearer';
  const refreshToken = RefreshToken.generate(user).token;
  const expiresIn = moment().add(jwtExpirationInterval, 'minutes');
  return {
    tokenType, accessToken, refreshToken, expiresIn,
  };
}

/**
 * Returns jwt token if registration was successful
 * @public
 */
exports.register = async (req, res, next) => {
  try {

    const invitationToken = req.body.inviteToken;
    const email = req.body.email;

    let user = new User(req.body);
```

```

    if(invitationToken){
      const { organizationId } =
        await Invitation.validateInvitation(invitationToken, email);
      user.role = 'organization';
      const organization = await Organization.get(organizationId);
      user.organization = organization.transform();
    }

    user = await (user).save();
    const userTransformed = user.transform();
    const token = generateTokenResponse(user, user.token());
    res.status(HttpStatus.CREATED);
    return res.json({ token, user: userTransformed });
  } catch (error) {
    return next(User.checkDuplicateEmail(error));
  }
};

/**
 * Returns jwt token if valid username and password is provided
 * @public
 */
exports.login = async (req, res, next) => {
  try {
    const { user, accessToken } = await User.findAndGenerateToken(req.body);
    const token = generateTokenResponse(user, accessToken);
    const userTransformed = user.transform();
    return res.json({ token, user: userTransformed });
  } catch (error) {
    return next(error);
  }
};

/**
 * login with an existing user or creates a new one if valid accessToken token
 * Returns jwt token
 * @public
 */
exports.oAuth = async (req, res, next) => {
  try {
    const { user } = req;
    const accessToken = user.token();
    const token = generateTokenResponse(user, accessToken);
    const userTransformed = user.transform();
    return res.json({ token, user: userTransformed });
  } catch (error) {
    return next(error);
  }
};

/**
 * Returns a new jwt when given a valid refresh token
 * @public
 */
exports.refresh = async (req, res, next) => {
  try {
    const { email, refreshToken } = req.body;
    const refreshObject = await RefreshToken.findOneAndRemove({

```



```

        userEmail: email,
        token: refreshToken,
    });
    const { user, accessToken } =
        await User.findAndGenerateToken({ email, refreshToken });
    const response = generateTokenResponse(user, accessToken);
    return res.json(response);
} catch (error) {
    return next(error);
}
};

```

Основная функция здесь — `generateTokenResponse(user, accessToken)`, которая и производит генерацию токенов доступа и обновления на основе идентификатора пользователя. Таким образом, чтобы узнать, от какого пользователя идёт запрос — достаточно знать только его `accessToken`, который на время действия хранится в БД. Далее рассмотрим роутинг контроллера авторизации `happytail-api/src/api/routes/v1/auth.route.js`:

```

/**
 * @module auth.route
 */
'use strict';
const express = require('express');
const validate = require('express-validation');
const controller = require('../../controllers/auth');
const oAuthLogin = require('../../middlewares/auth').oAuth;
const {
    login,
    register,
    refresh,
} = require('../../validations/auth.validation');

const router = express.Router();

/**
 *
 * @api {post} v1/auth/register Register
 * @apiDescription Register a new user
 * @apiVersion 1.0.0
 * @apiName Register
 * @apiGroup Auth
 * @apiPermission public
 *
 * @apiParam {String} email User's email
 * @apiParam {String{6..128}} password User's password
 *
 * @apiSuccess (Created 201) {String} token.tokenType Access Token's type
 * @apiSuccess (Created 201) {String} token.accessToken Authorization Token
 * @apiSuccess (Created 201) {String} token.refreshToken Token to get a new accessToken
 *                                     after expiration time
 * @apiSuccess (Created 201) {Number} token.expiresIn Access Token's expiration time
 *                                     in milliseconds
 * @apiSuccess (Created 201) {String} token.timezone The server's Timezone
 *
 * @apiSuccess (Created 201) {String} user.id User's id
 * @apiSuccess (Created 201) {String} user.name User's name
 * @apiSuccess (Created 201) {String} user.email User's email
 * @apiSuccess (Created 201) {String} user.role User's role
 * @apiSuccess (Created 201) {Date} user.createdAt Timestamp
 *
 * @apiError (Bad Request 400) ValidationError Some parameters may contain invalid values
 */

```

```

router.route('/register')
    .post(validate(register), controller.register);

/**
 * @api {post} v1/auth/login Login
 * @apiDescription Get an accessToken
 * @apiVersion 1.0.0
 * @apiName Login
 * @apiGroup Auth
 * @apiPermission public
 *
 * @apiParam {String} email User's email
 * @apiParam {String{..128}} password User's password
 *
 * @apiSuccess {String} token.tokenType Access Token's type
 * @apiSuccess {String} token.accessToken Authorization Token
 * @apiSuccess {String} token.refreshToken Token to get a new accessToken
 * after expiration time
 * @apiSuccess {Number} token.expiresIn Access Token's expiration time
 * in milliseconds
 *
 * @apiSuccess {String} user.id User's id
 * @apiSuccess {String} user.name User's name
 * @apiSuccess {String} user.email User's email
 * @apiSuccess {String} user.role User's role
 * @apiSuccess {Date} user.createdAt Timestamp
 *
 * @apiError (Bad Request 400) ValidationError Some parameters may contain invalid values
 * @apiError (Unauthorized 401) Unauthorized Incorrect email or password
 */
router.route('/login')
    .post(validate(login), controller.login);

/**
 * @api {post} v1/auth/refresh-token Refresh Token
 * @apiDescription Refresh expired accessToken
 * @apiVersion 1.0.0
 * @apiName RefreshToken
 * @apiGroup Auth
 * @apiPermission public
 *
 * @apiParam {String} email User's email
 * @apiParam {String} refreshToken Refresh token aquired when user logged in
 *
 * @apiSuccess {String} tokenType Access Token's type
 * @apiSuccess {String} accessToken Authorization Token
 * @apiSuccess {String} refreshToken Token to get a new accessToken after expiration time
 * @apiSuccess {Number} expiresIn Access Token's expiration time in milliseconds
 *
 * @apiError (Bad Request 400) ValidationError Some parameters may contain invalid values
 * @apiError (Unauthorized 401) Unauthorized Incorrect email or refreshToken
 */
router.route('/refresh-token')
    .post(validate(refresh), controller.refresh);

router.route('/vk')
    .post(oAuthLogin('vk'), controller.oAuth);

```

```
module.exports = router;
```

Кроме написания самого роутинга этот файл отвечает за параметры методов, их доступности и прохождения ими валидации по схеме на разные условия, например, что пароль должен минимальную и максимальную длины, также состоять из доступных символов:

```
/*
 *
 * @api {post} v1/auth/register Register
 * @apiDescription Register a new user
 * @apiVersion 1.0.0
 * @apiName Register
 * @apiGroup Auth
 * @apiPermission public
 *
 * @apiParam {String} email User's email
 * @apiParam {String{6..128}} password User's password
 *
 * @apiSuccess (Created 201) {String} token.tokenType Access Token's type
 * @apiSuccess (Created 201) {String} token.accessToken Authorization Token
 * @apiSuccess (Created 201) {String} token.refreshToken Token to get a new accessToken
 * after expiration time
 * @apiSuccess (Created 201) {Number} token.expiresIn Access Token's expiration time
 * in milliseconds
 * @apiSuccess (Created 201) {String} token.timezone The server's Timezone
 *
 * @apiSuccess (Created 201) {String} user.id User's id
 * @apiSuccess (Created 201) {String} user.name User's name
 * @apiSuccess (Created 201) {String} user.email User's email
 * @apiSuccess (Created 201) {String} user.role User's role
 * @apiSuccess (Created 201) {Date} user.createdAt Timestamp
 *
 * @apiError (Bad Request 400) ValidationError Some parameters may contain invalid values
```

Листинг кода проекта

happytail-api/src/index.js

```
/**
 * @module index
 */
'use strict';
Promise = require('bluebird');

const app = require('./config/express');
const {port} = require('./config/vars');
const mongoose = require('./config/mongoose');

mongoose.connect();

app.listen(port, () => console.info(`Server has been started on ${port} port`));
/**
 * Exports express
 * @public
 */
module.exports = app;
happytail-api/src/config/express.js
/**
 * @module express
 */
'use strict';

const express = require('express');
const bodyParser = require('body-parser');
const passport = require('passport');
const cors = require('cors');
const helmet = require('helmet');

const routes = require('../api/routes/v1');
const error = require('../api/middlewares/error');
// const bot = require('../telegraf');
const strategies = require('../passport');
const {token} = require('../config/vars');

/**
 * Express instance
 * @public
 */
const app = express();

// parse body params and attache them to req.body
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// secure apps by setting various HTTP headers
app.use(helmet());

// enable CORS - Cross Origin Resource Sharing
app.use(cors());

// app.options("/*", function(req, res, next){
//     res.header('Access-Control-Allow-Origin', '*');
//     res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,PATCH,DELETE,OPTIONS');
//     res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-Length,
```

```

//      res.send(200);
// });

// enable authentication
app.use(passport.initialize());
passport.use('jwt', strategies.jwt);

app.use('/v1', routes);

// app.use(bot.webhookCallback(`/v1/bot${token}`));

// if error is not an instanceof APIError, convert it.
app.use(error.converter);

// catch 404 and forward to error handler
app.use(error.notFound);

// error handler, send stacktrace only during development
app.use(error.handler);

module.exports = app;
happytail-api/src/config/mongoose.js
/**
 * @module mongoose
 */
'use strict';

const mongoose = require('mongoose');
const {mongo, env} = require('./vars');

// Exit application on error
mongoose.connection.on('error', (err) => {
  console.error(`MongoDB connection error: ${err}`);
  process.exit(-1);
});

mongoose.connection.on('connected', () => {
  console.debug('connected');
});
mongoose.connection.on('disconnected', () => {
  console.debug('disconnected');
});

process.on('SIGINT', function() {
  mongoose.connection.close(function () {
    console.log('Mongoose default connection disconnected through app termination');
    process.exit(0);
  });
});

// print mongoose logs in dev env
if (env === 'development') {
  mongoose.set('debug', true);
}

/**
 * Connect to mongo db

```

```

*
* @returns {object} Mongoose connection
* @public
*/
exports.connect = () => {
  mongoose.connect(mongo.uri, {
    reconnectTries: Number.MAX_VALUE, // Never stop trying to reconnect
    reconnectInterval: 500, // Reconnect every 500ms
    keepAlive: 1,
  });
  return mongoose.connection;
};

```

happytail-api/src/config/nodemailer.js

```

/**
 * @module nodemailer
 * @author Pavel Fediukovich
 */

'use strict';

const nodemailer = require('nodemailer');

const {user, password} = require('./vars');

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: user,
    pass: password
  }
});

const sendEmail = (email, content) => {
  return new Promise((resolve, reject) => {
    transporter.sendMail({
      from: user,
      to: email,
      subject: 'Access to Happy Tail',
      text: content
    }, (error, info) => {
      if (error) {
        reject(error);
      } else {
        resolve(info);
      }
    });
  });
};

```

module.exports = sendEmail;

happytail-api/src/config/passport.js

```

/**
 * @module passport
 */
'use strict';

const JwtStrategy = require('passport-jwt').Strategy;
const { ExtractJwt } = require('passport-jwt');

```

```

const { jwtSecret } = require('./vars');
const authProviders = require('../api/services/authProviders');
const User = require('../api/models/user.model');

const jwtOptions = {
  secretOrKey: jwtSecret,
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
};

const jwt = async (payload, done) => {
  try {
    const user = await User.findById(payload.sub);
    if (user) return done(null, user);
    return done(null, false);
  } catch (error) {
    return done(error, false);
  }
};

const oAuth = service => async ({token, user_id}, done) => {
  try {
    const userData = await authProviders[service](token, user_id);
    const user = await User.oAuthLogin(userData);
    return done(null, user);
  } catch (err) {
    return done(err);
  }
};

exports.jwt = new JwtStrategy(jwtOptions, jwt);
happytail-api/src/config/telegraf.js

/**
 * @module telegraf
 */
'use strict';

const Telegraf = require('telegraf');

const {url, token, env} = require('./vars');
const Property = require('../api/models/property.model');

const TELEGRAM_TOKEN = Property.names.TELEGRAM_TOKEN;

let options = {};

if (env === 'development') {
  const SocksProxyAgent = require('socks-proxy-agent');
  options = {
    telegram: {
      agent: new SocksProxyAgent('socks://127.0.0.1:9050', true)
    }
  };
}

// exports.getBot = async () => {
//   try {
//     const property = await Property.get(TELEGRAM_TOKEN);

```

```

//      return new Telegraf(property.value, options);
//    } catch (error) {
//      if (token) {
//        return new Telegraf(token, options);
//      }
//      error.message = "Telegram token is not provided";
//      throw error;
//    }
//  };
const bot = new Telegraf(token, options);
bot.telegram.setWebhook(`${url}/v1/bot${token}`);

module.exports = bot;

happytail-api/src/config/vars.js

/**
 * @module vars
 */
'use strict';
const path = require('path');

// import .env variables
require('dotenv-safe').load({
  path: path.join(__dirname, '../../.env'),
  sample: path.join(__dirname, '../../.env.example'),
});

module.exports = {
  url: process.env.URL,
  env: process.env.NODE_ENV,
  user: process.env.USER,
  password: process.env.PASSWORD,
  port: process.env.PORT || 3000,
  jwtSecret: process.env.JWT_SECRET,
  jwtExpirationInterval: process.env.JWT_EXPIRATION_MINUTES,
  token: process.env.BOT_TOKEN,
  mongo: {
    uri: process.env.MONGO_URI || 'mongodb://localhost:27017/alarm'
  }
};

happytail-api/src/controllers/auth/index.js

/**
 * @module index
 */
'use strict';

const httpStatus = require('http-status');
const User = require('../..../models/user.model');
const Organization = require('../..../models/organization.model');
const RefreshToken = require('../..../models/refreshToken.model');
const Invitation = require('../..../models/invitation.model');
const moment = require('moment-timezone');
const { jwtExpirationInterval } = require('../..../config/vars');

/**
 * Returns a formatted object with tokens
 * @private
 */

```



```

function generateTokenResponse(user, accessToken) {
  const tokenType = 'Bearer';
  const refreshToken = RefreshToken.generate(user).token;
  const expiresIn = moment().add(jwtExpirationInterval, 'minutes');
  return {
    tokenType, accessToken, refreshToken, expiresIn,
  };
}

/**
 * Returns jwt token if registration was successful
 * @public
 */
exports.register = async (req, res, next) => {
  try {

    const invitationToken = req.body.inviteToken;
    const email = req.body.email;

    let user = new User(req.body);

    if(invitationToken){
      const { organizationId } = await Invitation.validateInvitation(invitationToken, email);
      user.role = 'organization';
      const organization = await Organization.get(organizationId);
      user.organization = organization.transform();
    }

    user = await (user).save();
    const userTransformed = user.transform();
    const token = generateTokenResponse(user, user.token());
    res.status(HttpStatus.CREATED);
    return res.json({ token, user: userTransformed });
  } catch (error) {
    return next(User.checkDuplicateEmail(error));
  }
};

/**
 * Returns jwt token if valid username and password is provided
 * @public
 */
exports.login = async (req, res, next) => {
  try {
    const { user, accessToken } = await User.findAndGenerateToken(req.body);
    const token = generateTokenResponse(user, accessToken);
    const userTransformed = user.transform();
    return res.json({ token, user: userTransformed });
  } catch (error) {
    return next(error);
  }
};

/**
 * login with an existing user or creates a new one if valid accessToken token
 * Returns jwt token
 * @public
 */
exports.oauth = async (req, res, next) => {
  try {

```

```

        const { user } = req;
        const accessToken = user.token();
        const token = generateTokenResponse(user, accessToken);
        const userTransformed = user.transform();
        return res.json({ token, user: userTransformed });
    } catch (error) {
        return next(error);
    }
};

/**
 * Returns a new jwt when given a valid refresh token
 * @public
 */
exports.refresh = async (req, res, next) => {
    try {
        const { email, refreshToken } = req.body;
        const refreshObject = await RefreshToken.findOneAndRemove({
            userEmail: email,
            token: refreshToken,
        });
        const { user, accessToken } = await User.findAndGenerateToken({ email, refreshObject
        const response = generateTokenResponse(user, accessToken);
        return res.json(response);
    } catch (error) {
        return next(error);
    }
};

```

happytail-api/src/controllers/bot/index.js

```

/**
 * @module index
 */
'use strict';
const Markup = require('telegraf/markup');

const bot = require('.../config/telegraf');

const User = require('.../models/user.model');

const Property = require('.../models/property.model');

exports.handle = (req, res) => {
    bot.handleUpdate(req.body);
    res.status(200);
    res.end();
};

bot.start(async (ctx) => {
    try {
        const telegramId = ctx.from.id;
        const user = await User.findByIdByTelegramId(telegramId);
        ctx.reply(`Привет, ${user.name}`, Markup
            .keyboard(['/on', '/off'])
            .oneTime()
            .resize()
            .extra());
    } catch (error) {

```

```

        ctx.reply("Sorry, you are not authorized");
    }
});

// bot.hears('Оповещения', ctx => {
//     ctx.reply(Markup
//         .keyboard(['Включить', 'Выключить'])
//         .oneTime()
//         .resize()
//         .extra());
// });

bot.command('/on', async ctx => {
    try {
        let alarmEnabled = await Property.get('ALARM_ENABLED');

        if (!alarmEnabled)
        {
            alarmEnabled = new Property({name:'ALARM_ENABLED'});
        }

        alarmEnabled.value = 'true';
        await alarmEnabled.save();

        await ctx.reply("Alarm has been enabled");
    } catch (e) {
    }
});

bot.command('/off', async ctx => {
    try {
        let alarmEnabled = await Property.get('ALARM_ENABLED');

        if (!alarmEnabled)
        {
            alarmEnabled = new Property({name:'ALARM_ENABLED'});
        }

        alarmEnabled.value = 'false';
        await alarmEnabled.save();

        await ctx.reply("Alarm has been disabled");
    } catch (e) {
    }
});

```

happytail-api/src/controllers/invitation/index.js

```

/**
 * @module index
 */
'use strict';

const Invitation = require('../../models/invitation.model');
// const sendEmail = require('../../config/nodemailer');

/**
 * Create invitation
 * @public
 */
exports.create = async (req, res, next) => {

```

```

    try {
      const user = req.user;
      const { email, organizationId } = req.body;

      const inviteObject = await Invitation.generate(user, email, organizationId);

      // const fullUrl = `${req.protocol}://${req.get('host')}/register/${inviteObject.token}`;
      const fullUrl = `${req.protocol}://95.213.28.116:5000/signup/${inviteObject.token}`;
      // await sendEmail(email, fullUrl);
      return res.json(inviteObject);
    } catch (error) {
      return next(error);
    }
  };
};

```

happytail-api/src/controllers/organization/index.js

```

/**
 * @module index.js
 */

'use strict';

const httpStatus = require('http-status');
const Organization = require('../../models/organization.model');
const User = require('../../models/user.model');
const { omit } = require('lodash');
const { handler: errorHandler } = require('../../middlewares/error');

/**
 * Load task and append to req.
 * @public
 */
exports.load = async (req, res, next, id) => {
  try {
    const organization = await Organization.get(id);
    req.locals = { organization };
    return next();
  } catch (error) {
    return errorHandler(error, req, res);
  }
};

/**
 * Load user and append to req.
 * @public
 */
exports.get = (req, res) => res.json(req.locals.organization.transform());

/**
 * Load user and append to req.
 * @public
 */
exports.getMembers = async (req, res, next) => {
  const { organization } = req.locals;
  try {
    const members = await User.find({organization});

    res.json(members.map(member => member.transform()));
  } catch (e) {

```

```

        next(e);
    }
};

/**
 * Create new user
 * @public
 */
exports.create = async (req, res, next) => {
    try {
        const organization = new Organization(req.body);
        const created = await organization.save();
        res.status(HttpStatus.CREATED);
        res.json(created.transform());
    } catch (error) {
        next(error);
    }
};

/**
 * Replace existing user
 * @public
 */
exports.replace = async (req, res, next) => {
    try {
        const { organization } = req.locals;

        const newOrganization = new Organization(req.body);
        const newOrganizationObject = omit(newOrganization.toObject(), '_id');

        await organization.
            update(newOrganizationObject, { override: true, upsert: true });
        const savedOrganization = await Organization.get(organization._id);

        res.json(savedOrganization.transform());
    } catch (error) {
        next(error);
    }
};

/**
 * Update existing user
 * @public
 */
exports.update = (req, res, next) => {
    const { organization } = req.locals;

    Object.assign(organization, req.body);

    organization.save()
        .then(saved => res.json(saved.transform()))
        .catch(err => {
            next(err);
        });
};

/**
 * Get user list
 * @public
 */

```

```

exports.list = async (req, res, next) => {
  try {
    const organizations = await Organization.list(req.query);
    res.json(organizations.map(org=>org.transform()));
  } catch (error) {
    next(error);
  }
};

```

```

/**
 * Delete user
 * @public
 */
exports.remove = (req, res, next) => {
  const { organization } = req.locals;

  organization.remove()
    .then(() => res.status(httpStatus.NO_CONTENT).end())
    .catch(e => next(e));
};

```

happytail-api/src/controllers/property/index.js

```

/**
 * @module index
 */

const httpStatus = require('http-status');

const Property = require('../../models/property.model');

/**
 * Create new property
 * @public
 */
exports.create = async (req, res, next) => {
  try {
    const property = new Property(req.body);
    const savedProperty = await property.save();
    res.status(httpStatus.CREATED);
    res.json(savedProperty);
  } catch (e) {
    next(e);
  }
};

exports.list = async (req, res, next) => {
  try {
    const properties = await Property.list();
    res.json(properties);
  } catch (e) {
    next(e);
  }
};

```

happytail-api/src/controllers/task/index.js

```

/**
 * @module index
 */

```

```

'use strict';

const httpStatus = require('http-status');
const {omit} = require('lodash');
const reject = require('lodash/reject');
const Task = require('../../models/task.model');
const User = require('../../models/user.model');
const Org = require('../../models/organization.model');
const {handler: errorHandler} = require('../../middlewares/error');

/**
 * Load task and append to req.
 * @public
 */
exports.load = async (req, res, next, id) => {
  try {
    const task = await Task.get(id);
    req.locals = {task};
    return next();
  } catch (error) {
    return errorHandler(error, req, res);
  }
};

/**
 * Load user and append to req.
 * @public
 */
exports.get = (req, res) => res.json(req.locals.task.transform());

/**
 * Create new user
 * @public
 */
exports.create = async (req, res, next) => {
  try {
    const {user} = req;
    const loggedUserId = req.user._id;
    let task = new Task(req.body);
    task.organization = user.organization;
    task.ownerId = loggedUserId;
    task = await task.save();
    const organization = await Org.findOne({id: user.organization.id});
    organization.taskStats.all += 1;
    organization.taskStats.active += 1;
    await organization.save();
    res.status(httpStatus.CREATED);
    res.json(task.transform());
  } catch (error) {
    next(error);
  }
};

/**
 * Replace existing user
 * @public
 */
exports.replace = async (req, res, next) => {

```

```

    try {
      const {task, user} = req.locals;

      if (task.ownerId !== user._id) {
        const err = new Error();
        err.stack = httpStatus.FORBIDDEN;
        throw err;
      }
      const newTask = new Task(req.body);

      await task.update(newTask, {override: true, upsert: true});
      const savedTask = await Task.findById(task._id);

      res.json(savedTask.transform());
    } catch (error) {
      next(error);
    }
  };

  /**
   * Update existing user
   * @public
   */
  exports.update = (req, res, next) => {
    const {task, user} = req.locals;

    if (task.ownerId !== user._id) {
      const err = new Error();
      err.stack = httpStatus.FORBIDDEN;
      throw err;
    }

    const updatedTask = req.body;
    Object.assign(task, updatedTask);

    task.save()
      .then(savedTask => res.json(savedTask.transform()))
      .catch(next);
  };

  /**
   * Get user list
   * @public
   */
  exports.list = async (req, res, next) => {
    try {
      const tasks = await Task.list(req.query);
      const transformedTasks = tasks.map(task => task.transform());
      res.json(transformedTasks);
    } catch (error) {
      next(error);
    }
  };

  exports.take = async (req, res, next) => {
    const {user} = req;
    const {task} = req.locals;
    try {
      if (task.status !== 'available') {
        const err = new Error('Task is not available');

```



```

        err.stack = httpStatus.BAD_REQUEST;
        throw err;
    }

    if (!task.hasManyAssignee) {
        task.status = 'assigned';
        await task.save();
    }
    user.tasks.push(task);
    await user.save();
    res.sendStatus(httpStatus.ACCEPTED);
} catch (e) {
    next(e);
}
};

exports.release = async (req, res, next) => {
    const {user} = req;
    const {task} = req.locals;
    try {
        if (task.status !== 'assigned' && !task.hasManyAssignee) {
            const err = new Error('Task is not assigned');
            err.stack = httpStatus.BAD_REQUEST;
            throw err;
        }

        task.status = 'available';
        await task.save();
        user.tasks = reject(user.tasks, item => item._id === task._id);
        await user.save();
        res.sendStatus(httpStatus.ACCEPTED)
    } catch (e) {
        next(e);
    }
};

exports.finish = async (req, res, next) => {
    const {user} = req;
    const {task} = req.locals;
    try {
        if (task.status !== 'assigned' && !task.hasManyAssignee) {
            const err = new Error('Task is not assigned');
            err.status = httpStatus.BAD_REQUEST;
            throw err;
        }

        if (!task.hasManyAssignee) {
            task.status = 'done';
            await task.save();
        }
        const userTask = user.tasks.find(item => item._id === task._id);
        userTask.status = 'done';
        await user.save();
        const organization = await Org.findOne({id: user.organization.id});
        organization.taskStats.done += 1;
        organization.taskStats.active -= 1;
        const organizationTask = organization.tasks.find(item => item._id === task._id);
        organizationTask.status = 'done';
        await organizationTask.save();
        await organization.save();
    }
};

```

```

        res.sendStatus(httpStatus.ACCEPTED)
    } catch (e) {
        next(e);
    }
};

/**
 * Delete user
 * @public
 */
exports.remove = (req, res, next) => {
    const {task, user} = req.locals;

    if (task.ownerId !== user._id) {
        const err = new Error();
        err.stack = httpStatus.FORBIDDEN;
        throw err;
    }

    task.remove()
        .then(() => res.status(httpStatus.NO_CONTENT).end())
        .catch(e => next(e));
};

happytail-api/src/controllers/user/index.js

/**
 * @module index
 */
'use strict';

const httpStatus = require('http-status');
const { omit } = require('lodash');
const User = require('../../models/user.model');
const { handler: errorHandler } = require('../../middlewares/error');

/**
 * Load user and append to req.
 * @public
 */
exports.load = async (req, res, next, id) => {
    try {
        const user = await User.get(id);
        req.locals = { user };
        return next();
    } catch (error) {
        return errorHandler(error, req, res);
    }
};

/**
 * Get user
 * @public
 */
exports.get = (req, res) => res.json(req.locals.user.transform());

/**
 * Get logged in user info
 * @public
 */

```

```

exports.loggedIn = (req, res) => res.json(req.user.transform());

/**
 * Create new user
 * @public
 */
exports.create = async (req, res, next) => {
  try {
    const user = new User(req.body);
    const savedUser = await user.save();
    res.status(HttpStatus.CREATED);
    res.json(savedUser.transform());
  } catch (error) {
    next(User.checkDuplicateEmail(error));
  }
};

/**
 * Replace existing user
 * @public
 */
exports.replace = async (req, res, next) => {
  try {
    const { user } = req.locals;
    const newUser = new User(req.body);
    const omitRole = user.role !== 'admin' ? 'role' : '';
    const newUserObject = omit(newUser.toObject(), '_id', omitRole);

    await user.update(newUserObject, { override: true, upsert: true });
    const savedUser = await User.findById(user._id);

    res.json(savedUser.transform());
  } catch (error) {
    next(User.checkDuplicateEmail(error));
  }
};

/**
 * Update existing user
 * @public
 */
exports.update = (req, res, next) => {
  const omitRole = req.locals.user.role !== 'admin' ? 'role' : '';
  const updatedUser = omit(req.body, omitRole);
  const user = Object.assign(req.locals.user, updatedUser);

  user.save()
    .then(savedUser => res.json(savedUser.transform()))
    .catch(e => next(User.checkDuplicateEmail(e)));
};

/**
 * Get user list
 * @public
 */
exports.list = async (req, res, next) => {
  try {
    const users = await User.list(req.query);
    const transformedUsers = users.map(user => user.transform());
    res.json(transformedUsers);
  }
};

```

```

    } catch (error) {
      next(error);
    }
  };

/**
 * Delete user
 * @public
 */
exports.remove = (req, res, next) => {
  const { user } = req.locals;

  user.remove()
    .then(() => res.status(httpStatus.NO_CONTENT).end())
    .catch(e => next(e));
};

```

happytail-api/src/middlewares/auth.js

```

/**
 * @module auth
 */
'use strict';
const httpStatus = require('http-status');
const passport = require('passport');
const User = require('../models/user.model');
const APIError = require('../utils/APIError');

const ADMIN = 'admin';
const ORGANIZATION = 'organization';
const USER = 'user';
const LOGGED_USER = '_loggedUser';

const handleJWT = (req, res, next, roles) =>
  async (err, user, info) => {
    const error = err || info;
    const logIn = Promise.promisify(req.logIn);
    const apiError = new APIError({
      message: error ? error.message : 'Unauthorized',
      status: httpStatus.UNAUTHORIZED,
      stack: error ? error.stack : undefined,
    });

    try {
      if (error || !user) throw error;
      await logIn(user, { session: false });
    } catch (e) {
      return next(apiError);
    }

    if (roles === LOGGED_USER) {
      if (user.role !== 'admin' &&
        req.params.userId !== user._id.toString()) {
        apiError.status = httpStatus.FORBIDDEN;
        apiError.message = 'Forbidden';
        return next(apiError);
      }
    }
  } else if (!roles.includes(user.role)) {
    apiError.status = httpStatus.FORBIDDEN;
    apiError.message = 'Forbidden';
    return next(apiError);
  }

```

```

    } else if (err || !user) {
      return next(apiError);
    }

    req.user = user;

    return next();
  };

exports.ADMIN = ADMIN;
exports.ORGANIZATION = ORGANIZATION;
exports.USER = USER;
exports.LOGGED_USER = LOGGED_USER;

exports.authorize = (roles = User.roles) => (req, res, next) =>
  passport.authenticate(
    'jwt', { session: false },
    handleJWT(req, res, next, roles),
  )(req, res, next);

exports.oAuth = service =>
  passport.authenticate(service, { session: false });
happytail-api/src/middlewares/error.js
/**
 * @module error
 */
'use strict';

const httpStatus = require('http-status');
const expressValidation = require('express-validation');
const APIError = require('../utils/APIError');
const { env } = require('../config/vars');

/**
 * Error handler. Send stacktrace only during development
 * @public
 */
const handler = (err, req, res, next) => {
  const response = {
    code: err.status,
    message: err.message || httpStatus[err.status],
    errors: err.errors,
    stack: err.stack,
  };

  if (env !== 'development') {
    delete response.stack;
  }

  res.status(err.status);
  res.json(response);
  res.end();
};
exports.handler = handler;

/**
 * If error is not an instanceof APIError, convert it.
 * @public
 */

```

```

exports.converter = (err, req, res, next) => {
  let convertedError = err;

  if (err instanceof expressValidation.ValidationError) {
    convertedError = new APIError({
      message: 'Validation Error',
      errors: err.errors,
      status: err.status,
      stack: err.stack,
    });
  } else if (!(err instanceof APIError)) {
    convertedError = new APIError({
      message: err.message,
      status: err.status,
      stack: err.stack,
    });
  }

  return handler(convertedError, req, res);
};

/**
 * Catch 404 and forward to error handler
 * @public
 */
exports.notFound = (req, res, next) => {
  const err = new APIError({
    message: 'Not found',
    status: httpStatus.NOT_FOUND,
  });
  return handler(err, req, res);
};

```

happytail-api/src/models/invitation.model.js

```

/**
 * @module invitation.model
 */
'use strict';

const mongoose = require('mongoose');
const httpStatus = require('http-status');
const crypto = require('crypto');
const moment = require('moment-timezone');

const APIError = require('../utils/APIError');

/**
 * Invitation Schema
 * @private
 */
const invitationSchema = new mongoose.Schema({
  token: {
    type: String,
    required: true,
    index: true,
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
});

```

```

    },
    organizationId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Organization',
      required: true,
    },
    email: {
      type: String,
      match: /^\\S+@\\S+\\.\\S+$/,
      unique: true,
      trim: true,
      lowercase: true,
    },
    expires: {type: Date},
  });

invitationSchema.statics = {

  /**
   * Get inviteObject
   *
   * @param {ObjectId} id - The objectId of user.
   * @returns {Promise<User, APIError>}
   */
  async validateInvitation(token, email) {
    try {
      let tokenObject =
        await this.findOneAndRemove({token: token, email: email}).exec();

      if (tokenObject) {
        const {expires} = tokenObject;
        if (moment().isAfter(expires)) {
          throw new APIError({
            message: 'Invitation token has been expired',
            status: httpStatus.BAD_REQUEST,
          });
        }
        return tokenObject;
      }

      throw new APIError({
        message: 'Invitation token does not exist',
        status: httpStatus.NOT_FOUND,
      });
    } catch (error) {
      throw error;
    }
  },

  /**
   * Generate an invitation and saves it into the database
   *
   * @param {User} user
   * @param {email} email
   * @returns {Invitation}
   */
  generate(user, email, organizationId) {
    const userId = user._id;
    const token = `${crypto.randomBytes(60).toString('hex')}`;

```

```

        const expires = moment().add(60, 'days').toDate();
        const tokenObject = new Invitation({
            token, userId, email, organizationId, expires,
        });
        tokenObject.save();
        return tokenObject;
    },

};

/**
 * @typedef Invitation
 */
const Invitation = mongoose.model('Invitation', invitationSchema);
module.exports = Invitation;

happytail-api/src/models/organization.model.js

/**
 * @module organization
 */

'use strict';

const mongoose = require('mongoose');
const httpStatus = require('http-status');
const Task = require('../task.model');

const APIError = require('../utils/APIError');

const type = ['shelter', 'grooming', 'pet_clinic'];

/**
 * Organization Schema
 * @private
 */
const organizationSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
    },
    type: {
        type: String,
        enum: type,
        required: true,
        default: 'shelter'
    },
    description: {
        type: String,
        trim: true,
    },
    latitude: {
        type: Number,
    },
    longitude: {
        type: Number,
    },
    phone: {
        type: String,
        trim: true,
    },

```



```

    },
    taskStats: {
      all: {
        type: Number,
        default: 0
      },
      active: {
        type: Number,
        default: 0
      },
      done: {
        type: Number,
        default: 0
      }
    }
  }
});

organizationSchema.pre('save', async function save(next) {
  try {
    return next();
  } catch (error) {
    return next(error);
  }
});

/**
 * Statics
 */
organizationSchema.statics = {

  type,

  /**
   * Get task
   *
   * @param {ObjectId} id - The objectId of user.
   * @returns {Promise<User, APIError>}
   */
  async get(id) {
    try {
      let organization;

      if (mongoose.Types.ObjectId.isValid(id)) {
        organization = await this.findById(id).exec();
      }
      if (organization) {
        return organization;
      }

      throw new APIError({
        message: 'Organization does not exist',
        status: httpStatus.NOT_FOUND,
      });
    } catch (error) {
      throw error;
    }
  },

  /**

```

```

    * List tasks in descending order of 'createdAt' timestamp.
    *
    * @param {number} skip - Number of tasks to be skipped.
    * @param {number} limit - Limit number of tasks to be returned.
    * @returns {Promise<Task[]>}
    */
    list({
      page = 1, perPage = 30,
    }) {

      return this.find()
        .sort({createdAt: -1})
        .skip(perPage * (page - 1))
        .limit(perPage)
        .exec();
    },
  );

organizationSchema.method({
  transform() {
    const transformed = {};
    const fields = ['id',
      'title',
      'type',
      'description',
      'location',
      'phone',
      'taskStats'];

    fields.forEach((field) => {
      transformed[field] = this[field];
    });

    return transformed;
  },
})

module.exports = mongoose.model('Organization', organizationSchema);
happytail-api/src/models/property.model.js
/**
 * @module property.model
 */
'use strict';

const mongoose = require('mongoose');

const APIError = require('../utils/APIError');

const names = {TELEGRAM_TOKEN: 'TELEGRAM_TOKEN'};

const propertySchema = mongoose.Schema({
  name: {
    type: String,
    require: true,
    unique: true
  },
  value: {
    type: String,

```

```

        require: true
    }
});

propertySchema.statics = {
    names,

    async get(name) {
        try {
            const property = await this.findOne({name: name}).exec();
            if (property) {
                return property;
            }
            return null;
        } catch (error) {
            throw error;
        }
    },

    list() {
        return this.find({}).exec();
    }
};

/**
 * @typedef Property
 */
module.exports = mongoose.model('Property', propertySchema);
happytail-api/src/models/refreshToken.model.js

/**
 * @module refreshToken.model
 */
'use strict';

const mongoose = require('mongoose');
const crypto = require('crypto');
const moment = require('moment-timezone');

/**
 * Refresh Token Schema
 * @private
 */
const refreshTokenSchema = new mongoose.Schema({
    token: {
        type: String,
        required: true,
        index: true,
    },
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true,
    },
    userEmail: {
        type: String,
        ref: 'User',
        required: true,
    },
    expires: { type: Date },

```

```

});

refreshTokenSchema.statics = {

  /**
   * Generate a refresh token object and saves it into the database
   *
   * @param {User} user
   * @returns {RefreshToken}
   */
  generate(user) {
    const userId = user._id;
    const userEmail = user.email;
    const token = `${userId}.${crypto.randomBytes(40).toString('hex')}`;
    const expires = moment().add(30, 'days').toDate();
    const tokenObject = new RefreshToken({
      token, userId, userEmail, expires,
    });
    tokenObject.save();
    return tokenObject;
  },

};

/**
 * @typedef RefreshToken
 */
const RefreshToken = mongoose.model('RefreshToken', refreshTokenSchema);
module.exports = RefreshToken;

happytail-api/src/models/task.model.js

/**
 * @module task.model
 */
'use strict';
const mongoose = require('mongoose');
const httpStatus = require('http-status');
const {omitBy, isNil} = require('lodash');
const moment = require('moment-timezone');

const APIError = require('../utils/APIError');

/**
 * Task Status
 */
const status = ['available', 'in_progress', 'hidden', 'done', 'deleted', 'assigned'];

/**
 * Task Priority
 */
const priority = ['low', 'medium', 'high', 'hot', 'extra'];

/**
 * Task type
 */
const type = ['auto', 'animals', 'remote', 'donate', 'other'];

/**

```

```

* Model Schema
* @private
*/
const taskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    trim: true,
  },
  description: {
    type: String,
    required: true,
    trim: true,
  },
  latitude: {
    type: Number,
  },
  longitude: {
    type: Number,
  },
  status: {
    type: String,
    enum: status,
    default: 'available',
    required: true,
  },
  priority: {
    type: String,
    enum: priority,
    default: 'medium',
    required: true,
  },
  type: {
    type: String,
    enum: type,
    default: 'other',
    required: true,
  },
  ownerId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'user',
    required: true,
  },
  organization: {
    type: Object,
    ref: 'organization',
  },
  date: {
    type: Date,
  },
  duration: {
    type: Number,
  },
  hasManyAssignee: {
    type: Boolean
  },
}, {
  timestamps: true,

```

```

});

/**
 * Methods
 */
taskSchema.method({
  transform() {
    const transformed = {};
    const fields = ['id', 'title', 'latitude', 'longitude', 'type', 'priority', 'status', 'ownerId',
      'updatedAt', 'createdAt', 'date', 'duration', 'hasManyAssignee', 'organization'];

    fields.forEach((field) => {
      transformed[field] = this[field];
    });

    return transformed;
  },
});

taskSchema.pre('save', async function save(next) {
  try {

    return next();
  } catch (error) {
    return next(error);
  }
});

taskSchema.index({ title: "text" });

/**
 * Statics
 */
taskSchema.statics = {

  priority,
  status,
  type,

  /**
   * Get task
   *
   * @param {ObjectId} id - The objectId of user.
   * @returns {Promise<User, APIError>}
   */
  async get(id) {
    try {
      let task;

      if (mongoose.Types.ObjectId.isValid(id)) {
        task = await this.findById(id).exec();
      }
      if (task) {
        return task;
      }

      throw new APIError({
        message: 'Task does not exist',
        status: httpStatus.NOT_FOUND,
      });
    }
  }
};

```

```

    } catch (error) {
      throw error;
    }
  },

  /**
   * List tasks in descending order of 'createdAt' timestamp.
   *
   * @param {number} skip - Number of tasks to be skipped.
   * @param {number} limit - Limit number of tasks to be returned.
   * @returns {Promise<Task[]>}
   */
  list({
    page = 1, perPage = 30, title, priority
  }) {

    const textSearch = title ? {$text: {$search: title}} : {};

    const statusFilter = {status:{
      $nin: [ 'hidden', 'deleted' ]
    }};

    const filters = [textSearch, statusFilter];
    if (priority) {
      filters.push({priority: {$in: priority}});
    }
    return this.find().and(filters)
      .sort({createdAt: -1})
      .skip(perPage * (page - 1))
      .limit(perPage)
      .exec();
  },

};

/**
 * @typedef Task
 */
module.exports = mongoose.model('Task', taskSchema);
happytail-api/src/models/user.model.js

/**
 * @module user.model
 */
'use strict';
const mongoose = require('mongoose');
const httpStatus = require('http-status');
const {omitBy, isNil} = require('lodash');
const bcrypt = require('bcryptjs');
const moment = require('moment-timezone');
const jwt = require('jwt-simple');
const uuidv4 = require('uuid/v4');

const APIError = require('../utils/APIError');
const {env, jwtSecret, jwtExpirationInterval} = require('../../config/vars');

/**
 * User Roles
 */

```

```

const roles = ['user', 'admin', 'organization'];

/**
 * User Schema
 * @private
 */
const userSchema = new mongoose.Schema({
  email: {
    type: String,
    match: /^[^S+@\S+\.\S+$/ ,
    // required: true,
    unique: true,
    trim: true,
    lowercase: true,
  },
  organization: {
    type: Object,
    ref: 'organization',
  },
  latitude: {
    type: Number,
  },
  longitude: {
    type: Number,
  },
  rating: {
    type: Number,
  },
  telegramId: {
    type: String,
  },
  password: {
    type: String,
    required: true,
    minlength: 6,
    maxlength: 128,
  },
  firstName: {
    type: String,
    maxlength: 128,
    trim: true,
  },
  lastName: {
    type: String,
    maxlength: 128,
    index: true,
    trim: true,
  },
  role: {
    type: String,
    enum: roles,
    default: 'user',
  },
  picture: {
    type: String,
    trim: true,
  },
  notifications: {
    telegram: {
      type: Boolean,

```



```

    },
    phone: {
      type: String,
      trim: true,
    },
    tasks: {
      type: Array,
      default: []
    },
    taskStats: {
      all: {
        type: Number,
      },
      undone: {
        type: Number,
      },
      done: {
        type: Number,
      }
    },
    services: {
      vk: String,
    },
  }, {
    timestamps: true,
  });

/**
 * Add your
 * - pre-save hooks
 * - validations
 * - virtuals
 */
userSchema.pre('save', async function save(next) {
  try {
    const {tasks} = this;
    if (!this.isModified('password')) {
      if (tasks) {
        tasks.forEach(task => {
          if (task.status === 'assigned') {
            this.taskStats.undone += 1;
          } else if (task.status === 'done') {
            this.taskStats.done += 1;
          }
        });
      }

      this.taskStats.all = tasks.length || 0;

      return next();
    }
  }

  const rounds = env === 'test' ? 1 : 10;

  const hash = await bcrypt.hash(this.password, rounds);
  this.password = hash;

  return next();

```

```

    } catch (error) {
        return next(error);
    }
});

/**
 * Methods
 */
userSchema.method({
  transform() {
    const transformed = {};
    const fields = ['id',
      'firstName',
      'lastName',
      'picture',
      'role',
      'createdAt',
      'updatedAt',
      'email',
      'organization',
      'phone',
      'tasks',
      'taskStats'];

    fields.forEach((field) => {
      transformed[field] = this[field];
    });

    return transformed;
  },

  token() {
    const payload = {
      exp: moment().add(jwtExpirationInterval, 'minutes').unix(),
      iat: moment().unix(),
      sub: this._id,
    };
    return jwt.encode(payload, jwtSecret);
  },

  async passwordMatches(password) {
    return bcrypt.compare(password, this.password);
  },
});

/**
 * Statics
 */
userSchema.statics = {

  roles,

  /**
   * Find user by telegramId
   *
   * @param {String} telegramId - The telegramId of user.
   * @returns {Promise<User, APIError>}
   */
  async findByTelegramId(telegramId) {
    try {

```

```

        const user = await this.findOne({telegramId: telegramId}).exec();
        if (user) {
            return user;
        }

        throw new APIError({
            message: 'User does not exist',
            status: httpStatus.NOT_FOUND,
        });
    } catch (error) {
        throw error;
    }
},

/**
 * Get user
 *
 * @param {ObjectId} id - The objectId of user.
 * @returns {Promise<User, APIError>}
 */
async get(id) {
    try {
        let user;

        if (mongoose.Types.ObjectId.isValid(id)) {
            user = await this.findById(id).exec();
        }
        if (user) {
            return user;
        }

        throw new APIError({
            message: 'User does not exist',
            status: httpStatus.NOT_FOUND,
        });
    } catch (error) {
        throw error;
    }
},

/**
 * Find user by email and tries to generate a JWT token
 *
 * @param {ObjectId} id - The objectId of user.
 * @returns {Promise<User, APIError>}
 */
async findAndGenerateToken(options) {
    const {email, password, refreshObject} = options;
    if (!email) throw new APIError({message: 'An email is required to generate a token'});

    const user = await this.findOne({email}).exec();
    const err = {
        status: httpStatus.UNAUTHORIZED,
        isPublic: true,
    };
    if (password) {
        if (user && await user.passwordMatches(password)) {
            return {user, accessToken: user.token()};
        }
    }
}

```

```

        err.message = 'Incorrect email or password';
    } else if (refreshObject && refreshObject.userEmail === email) {
        return {user, accessToken: user.token()};
    } else {
        err.message = 'Incorrect email or refreshToken';
    }
    throw new APIError(err);
},

/**
 * List users in descending order of 'createdAt' timestamp.
 *
 * @param {number} skip - Number of users to be skipped.
 * @param {number} limit - Limit number of users to be returned.
 * @returns {Promise<User[]>}
 */
list({
    page = 1, perPage = 30, firstName, email, role, organizationId
}) {
    const options = omitBy({firstName, email, role, organizationId}, isNil);

    return this.find(options)
        .sort({createdAt: -1})
        .skip(perPage * (page - 1))
        .limit(perPage)
        .exec();
},

/**
 * Return new validation error
 * if error is a mongoose duplicate key error
 *
 * @param {Error} error
 * @returns {Error|APIError}
 */
checkDuplicateEmail(error) {
    if (error.name === 'MongoError' && error.code === 11000) {
        return new APIError({
            message: 'Validation Error',
            errors: [{
                field: 'email',
                location: 'body',
                messages: ['"email" already exists'],
            }],
            status: httpStatus.CONFLICT,
            isPublic: true,
            stack: error.stack,
        });
    }
    return error;
},

async oAuthLogin({
    service, user_id, last_name, first_name,
}) {
    const user = await this.findOne({$or: [{[`services.${service}`]: user_id}, {email}]});
    if (user) {
        user.services[service] = user_id;
        if (!user.last_name) user.lastName = last_name;
        if (!user.first_name) user.firstName = first_name;
    }
}

```

```

        return user.save();
    }
    const password = uuidv4();
    return this.create({
        services: {[service]: user_id}, password, lastName: last_name, firstName: first_n
    });
},
};

/**
 * @typedef User
 */
module.exports = mongoose.model('User', userSchema);
happytail-api/src/routes/v1/tasks/finishTask.router.js

/**
 * @module taskAction.router
 */

'use strict';

const express = require('express');
const controller = require('../../../controllers/task/index');
const { authorize, USER, ORGANIZATION } = require('../../../middlewares/auth');

const router = express.Router({mergeParams: true});

router.post('/', authorize([USER, ORGANIZATION]), controller.finish)

module.exports = router;
;
happytail-api/src/routes/v1/tasks/releaseTask.router.js

/**
 * @module taskAction.router
 */

'use strict';

const express = require('express');
const controller = require('../../../controllers/task/index');
const { authorize, USER } = require('../../../middlewares/auth');

const router = express.Router({mergeParams: true});

router.post('/', authorize(USER), controller.release);

module.exports = router;
happytail-api/src/routes/v1/tasks/taskTask.router.js

/**
 * @module taskAction.router
 */

'use strict';

const express = require('express');
const controller = require('../../../controllers/task/index');
const { authorize, USER } = require('../../../middlewares/auth');

```

```

const router = express.Router({mergeParams: true});

router.post('/', authorize(USER), controller.take);

module.exports = router;
happytail-api/src/routes/v1/auth.route.js

/**
 * @module auth.route
 */
'use strict';
const express = require('express');
const validate = require('express-validation');
const controller = require('../../../controllers/auth');
const oAuthLogin = require('../../../middlewares/auth').oAuth;
const {
  login,
  register,
  refresh,
} = require('../../../validations/auth.validation');

const router = express.Router();

/**
 * @api {post} v1/auth/register Register
 * @apiDescription Register a new user
 * @apiVersion 1.0.0
 * @apiName Register
 * @apiGroup Auth
 * @apiPermission public
 *
 * @apiParam {String} email User's email
 * @apiParam {String{6..128}} password User's password
 *
 * @apiSuccess (Created 201) {String} token.tokenType Access Token's type
 * @apiSuccess (Created 201) {String} token.accessToken Authorization Token
 * @apiSuccess (Created 201) {String} token.refreshToken Token to get a new accessToken
 * after expiration time
 * @apiSuccess (Created 201) {Number} token.expiresIn Access Token's expiration time
 * in milliseconds
 * @apiSuccess (Created 201) {String} token.timezone The server's Timezone
 *
 * @apiSuccess (Created 201) {String} user.id User's id
 * @apiSuccess (Created 201) {String} user.name User's name
 * @apiSuccess (Created 201) {String} user.email User's email
 * @apiSuccess (Created 201) {String} user.role User's role
 * @apiSuccess (Created 201) {Date} user.createdAt Timestamp
 *
 * @apiError (Bad Request 400)
 * ValidationError Some parameters may contain invalid values
 */
router.route('/register')
  .post(validate(register), controller.register);

/**
 * @api {post} v1/auth/login Login
 * @apiDescription Get an accessToken
 * @apiVersion 1.0.0
 * @apiName Login

```

```

* @apiGroup Auth
* @apiPermission public
*
* @apiParam {String} email User's email
* @apiParam {String{..128}} password User's password
*
* @apiSuccess {String} token.tokenType Access Token's type
* @apiSuccess {String} token.accessToken Authorization Token
* @apiSuccess {String} token.refreshToken Token to get a new accessToken
after expiration time
* @apiSuccess {Number} token.expiresIn Access Token's expiration time
in milliseconds
*
* @apiSuccess {String} user.id User's id
* @apiSuccess {String} user.name User's name
* @apiSuccess {String} user.email User's email
* @apiSuccess {String} user.role User's role
* @apiSuccess {Date} user.createdAt Timestamp
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Incorrect email or password
*/
router.route('/login')
  .post(validate(login), controller.login);

/**
* @api {post} v1/auth/refresh-token Refresh Token
* @apiDescription Refresh expired accessToken
* @apiVersion 1.0.0
* @apiName RefreshToken
* @apiGroup Auth
* @apiPermission public
*
* @apiParam {String} email User's email
* @apiParam {String} refreshToken Refresh token aquired when user logged in
*
* @apiSuccess {String} tokenType Access Token's type
* @apiSuccess {String} accessToken Authorization Token
* @apiSuccess {String} refreshToken Token to get a new accessToken after expiration time
* @apiSuccess {Number} expiresIn Access Token's expiration time in milliseconds
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Incorrect email or refreshToken
*/
router.route('/refresh-token')
  .post(validate(refresh), controller.refresh);

router.route('/vk')

```

```

        .post(oAuthLogin('vk'), controller.oAuth);

/**
 * TODO: POST /v1/auth/reset-password
 */

module.exports = router;
happytail-api/src/routes/v1/bot.route.js

/**
 * @module bot.route
 */
'use strict';

const express = require('express');

const controller = require('../../controllers/bot');

const router = express.Router();

router.route('/')
    .post(controller.handle);

module.exports = router;
happytail-api/src/routes/v1/index.js

/**
 * @module index
 */
'use strict';

const express = require('express');

const userRoutes = require('./user.route');
const invitationRoutes = require('./invitation.route');
const authRoutes = require('./auth.route');
const propertyRoutes = require('./property.route');
const taskRoutes = require('./task.route');
const organizationRoutes = require('./organization.route');
// const botRoutes = require('./bot.route');

const {token} = require('../../../../config/vars');

const router = express.Router();

router.use('/users', userRoutes);
router.use('/invitations', invitationRoutes);
router.use('/auth', authRoutes);
router.use('/properties', propertyRoutes);
router.use('/tasks', taskRoutes);
router.use('/organizations', organizationRoutes);
// router.use(`/bot${token}`, botRoutes);

module.exports = router;
happytail-api/src/routes/v1/invitation.route.js

/**
 * @module invitation.route
 */
'use strict';

```



```

const express = require('express');
const validate = require('express-validation');

const controller = require('../controllers/invitation');

const {authorize, ADMIN, ORGANIZATION} =
    require('../middlewares/auth');
const {createInvitation} =
    require('../validations/invitation.validation');

const router = express.Router();

router
    .route('/')
    /**
     * @api {post} v1/invitations User Profile
     * @apiDescription Invite a user
     * @apiVersion 1.0.0
     * @apiName InviteUser
     * @apiGroup User
     * @apiPermission admin
     *
     * @apiParam {String} email User's email
     *
     * @apiSuccess {String} inviteToken Authorization Token
     * @apiSuccess {String} email Email of invited user
     * @apiSuccess {Number} expiresIn Invite Token's expiration time in miliseconds
     *
     * @apiError (Bad Request 400)
     * ValidationError Some parameters may contain invalid values
     * @apiError (Unauthorized 401)
     * Unauthorized Only authenticated users can invite user
     */
    .post(authorize([ADMIN, ORGANIZATION]),
        validate(createInvitation), controller.create);

module.exports = router;

happytail-api/src/routes/v1/members.route.js

/**
 * @module members.route
 */

'use strict';

const express = require('express');
const controller = require('../controllers/organization');
const router = express.Router({mergeParams: true});

router.get('/', controller.getMembers);

module.exports = router;

happytail-api/src/routes/v1/organization.route.js

/**
 * @module user.route
 */

```

```

'use strict';
const express = require('express');
const controller = require('../../controllers/organization');
const { authorize, ADMIN, LOGGED_USER } =
    require('../../middlewares/auth');

const router = express.Router();

/**
 * Load user when API with userId route parameter is hit
 */
router.param('organizationId', controller.load);

router
    .route('/')
    .get(controller.list)
    .post(authorize(ADMIN), controller.create);

router
    .route('/:organizationId')
    .get(controller.get)
    .put(authorize(ADMIN), controller.replace)
    .patch(authorize(ADMIN), controller.update)
    .delete(authorize(ADMIN), controller.remove);

router.use('/:organizationId/members',
    require('./members.route'));

module.exports = router;
happytail-api/src/routes/v1/property.route.js

/**
 * @module property.route
 */
'use strict';

const express = require('express');

const controller = require('../../controllers/property');

const router = express.Router();

router.route('/')
    .post(controller.create)
    .get(controller.list);

module.exports = router;
happytail-api/src/routes/v1/task.route.js

/**
 * @module task.route
 */
'use strict';

const express = require('express');
const validate = require('express-validation');
const controller = require('../../controllers/task');

```

```

const { authorize, ADMIN, ORGANIZATION, USER } =
  require('../middlewares/auth');
const {
  listTasks,
  createTask,
  replaceTask,
  updateTask,
} = require('../validations/task.validation');

const router = express.Router();

/**
 * Load user when API with userId route parameter is hit
 */
router.param('taskId', controller.load);

router
  .route('/')
  /**
   * @api {get} v1/tasks List tasks
   * @apiDescription Get a list of tasks
   * @apiVersion 1.0.0
   * @apiName ListTasks
   * @apiGroup Task
   * @apiPermission all
   *
   * @apiParam {Number{1-}} [page=1] List page
   * @apiParam {Number{1-100}} [perPage=1] Tasks per page
   * @apiParam {String} [title] Task title
   *
   * @apiSuccess {Object[]} tasks List of tasks.
   *
   * @apiError (Bad Request 400)
   * ValidationError Some parameters may contain invalid values
   */
  .get(validate(listTasks), controller.list)
  /**
   * @api {post} v1/tasks Create Task
   * @apiDescription Create a new task
   * @apiVersion 1.0.0
   * @apiName CreateTask
   * @apiGroup Task
   * @apiPermission organization, admin
   *
   * @apiHeader {String} Authorization User's access token
   *
   * @apiParam {String} title Task's title
   * @apiParam {String{6..128}} description Task's description
   * @apiParam {String{..128}} [name] User's name
   * @apiParam {String=user,admin} [role] User's role
   *
   * @apiSuccess (Created 201) {String} id User's id
   * @apiSuccess (Created 201) {String} name User's name
   * @apiSuccess (Created 201) {String} email User's email
   * @apiSuccess (Created 201) {String} role User's role
   * @apiSuccess (Created 201) {Date} createdAt Timestamp
   *
   * @apiError (Bad Request 400)
   * ValidationError Some parameters may contain invalid values
   * @apiError (Unauthorized 401)

```

```

* Unauthorized      Only authenticated users can create the data
* @apiError (Forbidden 403)
*   Forbidden      Only admins can create the data
*/
.post(authorize([ORGANIZATION, ADMIN]),
validate(createTask), controller.create);

```

router

```

.route('/:taskId')
/**
 * @api {get} v1/tasks/:id Get Task
 * @apiDescription Get task information
 * @apiVersion 1.0.0
 * @apiName GetTask
 * @apiGroup Task
 * @apiPermission all
 *
 * @apiSuccess {String} id      User's id
 * @apiSuccess {String} name    User's name
 * @apiSuccess {String} email   User's email
 * @apiSuccess {String} role    User's role
 * @apiSuccess {Date}   createdAt Timestamp
 *
 * @apiError (Unauthorized 401)
 * Unauthorized Only authenticated users can access the data
 * @apiError (Forbidden 403)
 * Forbidden    Only user with same id or admins can access the data
 * @apiError (Not Found 404)
 * NotFound     User does not exist
 */
.get(controller.get)
/**
 * @api {put} v1/users/:id Replace User
 * @apiDescription Replace the whole user document with a new one
 * @apiVersion 1.0.0
 * @apiName ReplaceUser
 * @apiGroup User
 * @apiPermission user
 *
 * @apiHeader {String} Athorization User's access token
 *
 * @apiParam {String} email      User's email
 * @apiParam {String{6..128}} password User's password
 * @apiParam {String{..128}} [name] User's name
 * @apiParam {String=user,admin} [role] User's role
 * (You must be an admin to change the user's role)
 *
 * @apiSuccess {String} id      User's id
 * @apiSuccess {String} name    User's name
 * @apiSuccess {String} email   User's email
 * @apiSuccess {String} role    User's role
 * @apiSuccess {Date}   createdAt Timestamp
 *
 * @apiError (Bad Request 400)
 * ValidationError Some parameters may contain invalid values
 * @apiError (Unauthorized 401)
 * Unauthorized Only authenticated users can modify the data
 * @apiError (Forbidden 403)
 * Forbidden    Only user with same id or admins can modify the data
 */

```

```

* @apiError (Not Found 404)
* NotFound      User does not exist
*/
.put(authorize([ORGANIZATION, ADMIN]),
validate(replaceTask), controller.replace)
/**
* @api {patch} v1/users/:id Update User
* @apiDescription Update some fields of a user document
* @apiVersion 1.0.0
* @apiName UpdateUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Athorization  User's access token
*
* @apiParam {String}      email      User's email
* @apiParam {String{6..128}} password User's password
* @apiParam {String{..128}} [name]    User's name
* @apiParam {String=user,admin} [role]  User's role
* (You must be an admin to change the user's role)
*
* @apiSuccess {String} id      User's id
* @apiSuccess {String} name    User's name
* @apiSuccess {String} email   User's email
* @apiSuccess {String} role    User's role
* @apiSuccess {Date}  createdAt Timestamp
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can modify the data
* @apiError (Forbidden 403)
* Forbidden Only user with same id or admins can modify the data
* @apiError (Not Found 404)
* NotFound      User does not exist
*/
.patch(authorize(USER), validate(updateTask), controller.update)
/**
* @api {patch} v1/users/:id Delete User
* @apiDescription Delete a user
* @apiVersion 1.0.0
* @apiName DeleteUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Athorization  User's access token
*
* @apiSuccess (No Content 204) Successfully deleted
*
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can delete the data
* @apiError (Forbidden 403)
* Forbidden Only user with same id or admins can delete the data
* @apiError (Not Found 404)
* NotFound      User does not exist
*/
.delete(authorize(ORGANIZATION), controller.remove);

```

```

router.use('/:taskId/take', require('./task/takeTask.router'));
router.use('/:taskId/release', require('./task/releaseTask.router'));

```

```

router.use('/:taskId/finish', require('./task/finishTask.router'));

module.exports = router;
happytail-api/src/routes/v1/user.route.js
/**
 * @module user.route
 */
'use strict';
const express = require('express');
const validate = require('express-validation');
const controller = require('../../../controllers/user');
const { authorize, ADMIN, LOGGED_USER } = require('../../../middlewares/auth');
const {
  listUsers,
  createUser,
  replaceUser,
  updateUser,
} = require('../../../validations/user.validation');

const router = express.Router();

/**
 * Load user when API with userId route parameter is hit
 */
router.param('userId', controller.load);

router
  .route('/')
  /**
   * @api {get} v1/users List Users
   * @apiDescription Get a list of users
   * @apiVersion 1.0.0
   * @apiName ListUsers
   * @apiGroup User
   * @apiPermission admin
   *
   * @apiHeader {String} Athorization User's access token
   *
   * @apiParam {Number{1-}} [page=1] List page
   * @apiParam {Number{1-100}} [perPage=1] Users per page
   * @apiParam {String} [name] User's name
   * @apiParam {String} [email] User's email
   * @apiParam {String=user,admin} [role] User's role
   *
   * @apiSuccess {Object[]} users List of users.
   *
   * @apiError (Unauthorized 401)
   * Unauthorized Only authenticated users can access the data
   * @apiError (Forbidden 403)
   * Forbidden Only admins can access the data
   */
  .get(authorize(ADMIN), validate(listUsers), controller.list)
  /**
   * @api {post} v1/users Create User
   * @apiDescription Create a new user
   * @apiVersion 1.0.0
   * @apiName CreateUser
   * @apiGroup User

```

```

* @apiPermission admin
*
* @apiHeader {String} Authorization User's access token
*
* @apiParam {String} email User's email
* @apiParam {String{6..128}} password User's password
* @apiParam {String{..128}} [name] User's name
* @apiParam {String=user,admin} [role] User's role
*
* @apiSuccess (Created 201) {String} id User's id
* @apiSuccess (Created 201) {String} name User's name
* @apiSuccess (Created 201) {String} email User's email
* @apiSuccess (Created 201) {String} role User's role
* @apiSuccess (Created 201) {Date} createdAt Timestamp
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can create the data
* @apiError (Forbidden 403)
* Forbidden Only admins can create the data
*/
.post(authorize(ADMIN), validate(createUser), controller.create);

```

router

```

.route('/profile')
/**
* @api {get} v1/users/profile User Profile
* @apiDescription Get logged in user profile information
* @apiVersion 1.0.0
* @apiName UserProfile
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Authorization User's access token
*
* @apiSuccess {String} id User's id
* @apiSuccess {String} name User's name
* @apiSuccess {String} email User's email
* @apiSuccess {String} role User's role
* @apiSuccess {Date} createdAt Timestamp
*
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated Users can access the data
*/
.get(authorize(), controller.loggedIn);

```

router

```

.route('/:userId')
/**
* @api {get} v1/users/:id Get User
* @apiDescription Get user information
* @apiVersion 1.0.0
* @apiName GetUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Authorization User's access token

```

```

*
* @apiSuccess {String} id          User's id
* @apiSuccess {String} name        User's name
* @apiSuccess {String} email       User's email
* @apiSuccess {String} role        User's role
* @apiSuccess {Date}   createdAt   Timestamp
*
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can access the data
* @apiError (Forbidden 403)
* Forbidden Only user with same id or admins can access the data
* @apiError (Not Found 404)
* NotFound User does not exist
*/
.get(authorize(LOGGED_USER), controller.get)
/**
* @api {put} v1/users/:id Replace User
* @apiDescription Replace the whole user document with a new one
* @apiVersion 1.0.0
* @apiName ReplaceUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Authorization User's access token
*
* @apiParam {String} email User's email
* @apiParam {String{6..128}} password User's password
* @apiParam {String{..128}} [name] User's name
* @apiParam {String=user,admin} [role] User's role
* (You must be an admin to change the user's role)
*
* @apiSuccess {String} id          User's id
* @apiSuccess {String} name        User's name
* @apiSuccess {String} email       User's email
* @apiSuccess {String} role        User's role
* @apiSuccess {Date}   createdAt   Timestamp
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can modify the data
* @apiError (Forbidden 403)
* Forbidden Only user with same id or admins can modify the data
* @apiError (Not Found 404)   NotFound User does not exist
*/
.put(authorize(LOGGED_USER), validate(replaceUser), controller.replace)
/**
* @api {patch} v1/users/:id Update User
* @apiDescription Update some fields of a user document
* @apiVersion 1.0.0
* @apiName UpdateUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Authorization User's access token
*
* @apiParam {String} email User's email
* @apiParam {String{6..128}} password User's password
* @apiParam {String{..128}} [name] User's name
* @apiParam {String=user,admin} [role] User's role

```



```

* (You must be an admin to change the user's role)
*
* @apiSuccess {String} id          User's id
* @apiSuccess {String} name        User's name
* @apiSuccess {String} email       User's email
* @apiSuccess {String} role        User's role
* @apiSuccess {Date}   createdAt   Timestamp
*
* @apiError (Bad Request 400)
* ValidationError Some parameters may contain invalid values
* @apiError (Unauthorized 401)
* Unauthorized Only authenticated users can modify the data
* @apiError (Forbidden 403)
* Forbidden Only user with same id or admins can modify the data
* @apiError (Not Found 404)
* NotFound User does not exist
*/
.patch(authorize(LOGGED_USER), validate(updateUser), controller.update)
/**
* @api {patch} v1/users/:id Delete User
* @apiDescription Delete a user
* @apiVersion 1.0.0
* @apiName DeleteUser
* @apiGroup User
* @apiPermission user
*
* @apiHeader {String} Athorization User's access token
*
* @apiSuccess (No Content 204) Successfully deleted
*
* @apiError (Unauthorized 401) Unauthorized
* Only authenticated users can delete the data
* @apiError (Forbidden 403) Forbidden
* Only user with same id or admins can delete the data
* @apiError (Not Found 404) NotFound
* User does not exist
*/
.delete(authorize(LOGGED_USER), controller.remove);

```

```
module.exports = router;
```

```
happytail-api/src/services/authProviders.js
```

```
/* eslint-disable camelcase */
```

```
const axios = require('axios');
```

```

exports.vk = async (access_token, user_id) => {
  const url = 'https://api.vk.com/method/getProfileInfo';
  const params = {access_token};
  const response = await axios.get(url, {params});
  const {
    first_name, last_name,
  } = response.data;
  return {
    service: 'vk',
    user_id,
    first_name,
    last_name,
  };
};

```

```

};

happytail-api/src/utils/APIError.js

/**
 * @module APIError
 */
'use strict';

const httpStatus = require('http-status');

/**
 * @extends Error
 */
class ExtendableError extends Error {
  constructor({
    message, errors, status, isPublic, stack,
  }) {
    super(message);
    this.name = this.constructor.name;
    this.message = message;
    this.errors = errors;
    this.status = status;
    this.isPublic = isPublic;
    this.isOperational = true; // This is required since
    //bluebird 4 doesn't append it anymore.
    this.stack = stack;
    // Error.captureStackTrace(this, this.constructor.name);
  }
}

/**
 * Class representing an API error.
 * @extends ExtendableError
 */
class APIError extends ExtendableError {
  /**
   * Creates an API error.
   * @param {string} message - Error message.
   * @param {number} status - HTTP status code of error.
   * @param {boolean} isPublic - Whether the
   * message should be visible to user or not.
   */
  constructor({
    message,
    errors,
    stack,
    status =
      httpStatus.INTERNAL_SERVER_ERROR,
    isPublic = false,
  }) {
    super({
      message, errors, status, isPublic, stack,
    });
  }
}

module.exports = APIError;

happytail-api/src/validations/auth.validation.js

/**

```

```

    * @module auth.validation
    */
    'use strict';

    const Joi = require('joi');

    module.exports = {
        // POST /v1/auth/register
        register: {
            body: {
                email: Joi.string().email().required(),
                password: Joi.string().required().min(6)
                    .max(128),
                inviteToken: Joi.string(),
            },
        },

        // POST /v1/auth/login
        login: {
            body: {
                email: Joi.string().email().required(),
                password: Joi.string().required().max(128),
            },
        },

        // POST /v1/auth/refresh
        refresh: {
            body: {
                email: Joi.string().email().required(),
                refreshToken: Joi.string().required(),
            },
        },
    };

```

happytail-api/src/validations/doorLog.validation.js

```

/**
 * @module doorLog.validation
 */
    'use strict';

    const Joi = require('joi');
    const DoorLog = require('../models/doorLog.model');

    module.exports = {
        // GET /v1/doorLog
        listDoorLogs: {
            query: {
                page: Joi.number().min(1),
                perPage: Joi.number().min(1).max(100),
                state: Joi.string().valid(DoorLog.states)
            },
        },

        // POST /v1/doorLog
        createDoorLog: {
            body: {
                state: Joi.string().valid(DoorLog.states)
            },
        },
    };

```

happytail-api/src/validations/invitation.validation.js

```

/**
 * @module invitation.validation
 */
'use strict';

const Joi = require('joi');

module.exports = {
  // POST /v1/invitations/
  createInvitation: {
    body: {
      email: Joi.string().email().required(),
    }
  }
};

happytail-api/src/validations/task.validation.js

/**
 * @module task.validation
 */
'use strict';

const Joi = require('joi');
const Task = require('../models/task.model');

module.exports = {

  // GET /v1/tasks
  listTasks: {
    query: {
      page: Joi.number().min(1),
      perPage: Joi.number().min(1).max(100),
      title: Joi.string(),
    },
  },

  // POST /v1/tasks
  createTask: {
    body: {
      title: Joi.string().required(),
      description: Joi.string().required(),
      location : Joi.object().keys({
        type: Joi.string().valid('Point').required(),
        coordinates: Joi.array().items(Joi.number()).min(2).max(2).required()
      }),
      status : Joi.string().valid(Task.status),
      priority : Joi.string().valid(Task.priority),
      type : Joi.string().valid(Task.type),
      date : Joi.date(),
      duration : Joi.number().min(1),
    },
  },

  // PUT /v1/tasks/:taskId
  replaceTask: {
    body: {
      title: Joi.string().required(),
      description: Joi.string().required(),
      location : Joi.object().keys({
        type: Joi.string().valid('Point').required(),

```

```

        coordinates: Joi.array().items(Joi.number()).min(2).max(2).required()
    )),
    status : Joi.string().valid(Task.status),
    priority : Joi.string().valid(Task.priority),
    type : Joi.string().valid(Task.type),
    date : Joi.date().required(),
    duration : Joi.number().min(1).required(),
  },
  params: {
    taskId: Joi.string().regex(/^[a-zA-F0-9]{24}$/).required(),
  },
},
// PATCH /v1/tasks/:taskId
updateTask: {
  body: {
    title: Joi.string().required(),
    description: Joi.string().required(),
    location : Joi.object().keys({
      type: Joi.string().valid('Point').required(),
      coordinates: Joi.array().items(Joi.number()).min(2).max(2).required()
    )),
    status : Joi.string().valid(Task.status),
    priority : Joi.string().valid(Task.priority),
    type : Joi.string().valid(Task.type),
    date : Joi.date().required(),
    duration : Joi.number().min(1).required(),
  },
  params: {
    taskId: Joi.string().regex(/^[a-zA-F0-9]{24}$/).required(),
  },
},
};

```

happytail-api/src/validations/user.validation.js

```

/**
 * @module user.validation
 */
'use strict';

const Joi = require('joi');
const User = require('../models/user.model');

module.exports = {

  // GET /v1/users
  listUsers: {
    query: {
      page: Joi.number().min(1),
      perPage: Joi.number().min(1).max(100),
      firstName: Joi.string(),
      email: Joi.string(),
      role: Joi.string().valid(User.roles),
    },
  },

  // POST /v1/users
  createUser: {
    body: {
      email: Joi.string().email().required(),

```

```

        password: Joi.string().min(6).max(128).required(),
        lastName: Joi.string().max(128),
        firstName: Joi.string().max(128),
        role: Joi.string().valid(User.roles),
      },
    },
  },
  // PUT /v1/users/:userId
  replaceUser: {
    body: {
      email: Joi.string().email().required(),
      password: Joi.string().min(6).max(128).required(),
      lastName: Joi.string().max(128),
      firstName: Joi.string().max(128),
      role: Joi.string().valid(User.roles),
    },
    params: {
      userId: Joi.string().regex(/^[a-zA-F0-9]{24}$/).required(),
    },
  },
  // PATCH /v1/users/:userId
  updateUser: {
    body: {
      email: Joi.string().email(),
      password: Joi.string().min(6).max(128),
      lastName: Joi.string().max(128),
      firstName: Joi.string().max(128),
      role: Joi.string().valid(User.roles),
    },
    params: {
      userId: Joi.string().regex(/^[a-zA-F0-9]{24}$/).required(),
    },
  },
},
};

```

Экспериментальная часть

Установка зависимостей проекта осуществляется с помощью менеджера пакетов npm (Node Package Manager). Для этого необходимо выполнить команду

```
npm install
```

из корневой директории проекта.

Npm также предоставляет возможность выполнять различные команды, для сборки или например для запуска тестов. Список всех доступных команд можно посмотреть в файле конфигурации package.json. Приведём его содержимое, поскольку этот файл чрезвычайно важен, и представляет особый интерес.

```

{
  "name": "happy-tail-web-ui",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@team-griffin/react-file-input": "^1.0.0",
    "axios": "^0.18.0",
    "bootstrap": "^4.1.3",
    "react": "^16.6.1",
    "react-dom": "^16.6.1",

```

```

"react-file-input": "^0.2.5",
"react-router-dom": "^4.3.1",
"react-scripts": "2.1.1",
"reactstrap": "^6.5.0"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject",
  "json-server": "./node_modules/.bin/json-server --watch db.json --port 3004"
},
"eslintConfig": {
  "extends": "react-app"
},
"browserslist": [
  ">0.2%",
  "not dead",
  "not ie <= 11",
  "not op_mini all"
],
"devDependencies": {
  "json-server": "^0.14.0"
}
}

```

Именно из этого файла `npm` получает информацию о том, какие зависимости необходимо установить для корректной работы приложения. Список необходимых пакетов с указанием точной версии (по принципу семантического версионирования) расположен в разделе `dependencies`. Команда `npm install`, скачает нужные версии всех модулей, перечисленных в этом разделе, из репозитория `npm` и установит их в директорию `node_modules` в корне проекта. В разделе `scripts` перечислены все доступные команды. Особого внимания заслуживает команда `build` которая запускает процесс сборки проекта и готовит его для запуска в работу. Сборка осуществляется с помощью `npm` модуля под названием `Webpack`, о котором следует рассказать отдельно.

`Webpack` это статический модульный сборщик для приложений на JavaScript. Подобные инструменты позволяют разработчикам упаковывать, компилировать и в целом организовывать все ресурсы, необходимые для проекта. Можно использовать не только сторонние библиотеки, но и собственные файлы. Подобная модульная система позволяет добиться лучшей организации проекта, так как получается, что он разбит на небольшие модули.

`Webpack` на данный момент является одним из самых мощных подобных инструментов. Он имеет открытый исходный код и позволяет решать множество разных задач.

Заключение

В ходе работы над данным проектом я усвоил на практике очень важные и бесценные знания в области программирования интернет приложений. Несмотря на то, что проект ещё находится в стадии активной разработки, даже эта малая часть полностью раскрывает процесс создания программ. Мною был получен ценнейший опыт использования современных технологий разработки применительно к реальной задаче.

Выводы

При разработке интернет приложений немаловажную роль в достижении успеха играет выбор правильных технологий, удовлетворяющих специфике поставленной задачи.

Не последним фактором при выборе технологии также является её популярность и актуальность, ведь чем шире сообщество людей, использующих ту или иную технологию, тем больше обучающих материалов по теме, и проще получить помощь или ответ на какой-либо технический вопрос.

На примере данного приложения я убедился в том, что выбранный стек технологий в полной мере справляется со всеми целями, которые были поставлены. Связка Node.js и MongoDB на сервере показывают себя хорошо, обеспечивая стабильную работу и высокий уровень быстродействия. На клиенте же применены React.js очень выгодно отличается от более старых технологий таких как jQuery во многом потому, что React.js позволяет абстрагироваться от работы с DOM (Document Object Model) напрямую, тем самым сводя к минимуму вероятность ошибки и повышая производительность.

Список литературы

1. Node.js в действии — Кантелон Майк, Хартер Марк, Головайчук TJ, Райлих Натан
2. Выразительный JavaScript — Marijn Haverbeke
3. Компьютерные сети: Принципы, технологии, протоколы : Учебник — Виктор Олифер
4. <https://nodejs.org/en/docs/> — Официальная Документация по Node.js
5. <http://expressjs.com/ru/api.html/> - Документация Express
6. <https://mongoosejs.com/docs/api.html> — Документация по Mongoose
7. <https://stackoverflow.com/> — Вопросы и ответы
8. <https://habr.com/post/340146/> — JSON Web Tokens
9. <https://developer.mozilla.org/en-US/> — Документация по JavaScript и Browser API
10. <https://refactoring.guru/ru/design-patterns/catalog> — Шаблоны проектирования
11. <https://expressjs.com/ru/advanced/best-practice-security.html> — Лучшие практические методы для рабочей среды
12. <https://github.com/getify/You-Dont-Know-JS> — Книга по JavaScript
13. <https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azurearchitecture/common-web-application-architectures> — Современные архитектуры интернет приложений
14. <https://www.w3.org/standards/webdesign/> — Web Стандарты