

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І.І.МЕЧНИКОВА
ІНСТИТУТ МАТЕМАТИКИ, ЕКОНОМІКИ І МЕХАНІКИ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ
СИСТЕМ

КУРСОВИЙ ПРОЕКТ
з дисципліни
«Мережні інформаційні технології»
на тему:
«Розгортання системи оптимізації навчального процесу
"inStudy" в хмарі Heroku»

студента 6 курсу
спеціальності «Комп'ютерні системи та мережі»
Царюк А.О..

Керівник: Антоненко О.С.

Захищено « » 201 р.

Кількість балів: Оцінка: ECTS

(Підпис керівника)

Члени комісії

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 ПОСТАНОВКА ЗАДАЧИ.....	4
2 АРХИТЕКТУРА И СРЕДСТВА РЕАЛИЗАЦИИ.....	5
2.1 Описание общей архитектуры системы.....	5
2.2 Описание облачной платформы.....	7
3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ.....	10
4 РЕГИСТРАЦИЯ В ОБЛАЧНОЙ ПЛАТФОРМЕ.....	12
5 НАСТРОЙКА ДЕПЛОЯ СИСТЕМЫ.....	15
6 ТЕСТИРОВАНИЕ СИСТЕМЫ.....	18
ВЫВОДЫ.....	21
ЛИТЕРАТУРА.....	22
ПРИЛОЖЕНИЕ А.....	23
ПРИЛОЖЕНИЕ Б.....	24
ПРИЛОЖЕНИЕ В.....	27

ВВЕДЕНИЕ

Heroku — платформа, поддерживающая ряд языков программирования и сторонних сервисов для интеграции. С 2010 года является дочерней компанией Salesforce.com. Heroku, одна из первых облачных платформ, появилась в июне и изначально поддерживала только язык программирования , но на данный момент список поддерживаемых языков также включает в себя и NodeJS. На серверах Heroku используются операционные системы Linux. Heroku предоставляет очень удобное управление приложениями как через визуальный интерфейс на сайте, так и через командный терминал. Используя модель PaaS Heroku позволяет использовать различные дополнения (addons) и языки программирования, затрачивая минимум времени на их интеграцию и настройку их взаимодействия.

Целью курсового проекта является развертывание системы оптимизации процесса образования “inStudy” на платформе Heroku под NodeJS back-end (api server) и VueJS front-end.

1 ПОСТАНОВКА ЗАДАЧИ

Целью курсового проекта является настройка процесса развертывания back-end и fron-end частей системы «inStudy». Система предназначена для оптимизации процесса обучения и требует стабильности в работе, а так же гибкости в настройках.

Выбранный инструмент развертывания должен позволять создавать оптимальную настройку ENV в зависимости от нужд системы и давать возможность динамически изменять характеристики виртуальных машин для достижения максимально успешного сочетания цена-качество.

Так же, необходимо предусмотреть возможности интеграции системы с third party software для получения новых возможностей и реализации нового функционала.

2 АРХИТЕКТУРА И СРЕДСТВА РЕАЛИЗАЦИИ

2.1 Описание общей архитектуры системы

В качестве архитектуры приложения была выбрана трехуровневая (three-tier) клиент-серверная архитектура (рис. 2.1). Такая архитектура на данный момент является очень распространенной, так как разделение сервера приложений и сервера базы данных делает архитектуру устойчивой к сбоям в работе сервера базы данных. Например, если по техническим причинам сервер базы данных не работает или просто не удастся совершить соединение, то приложение все равно будет продолжать свою работу и может информировать пользователей о технических неполадках.

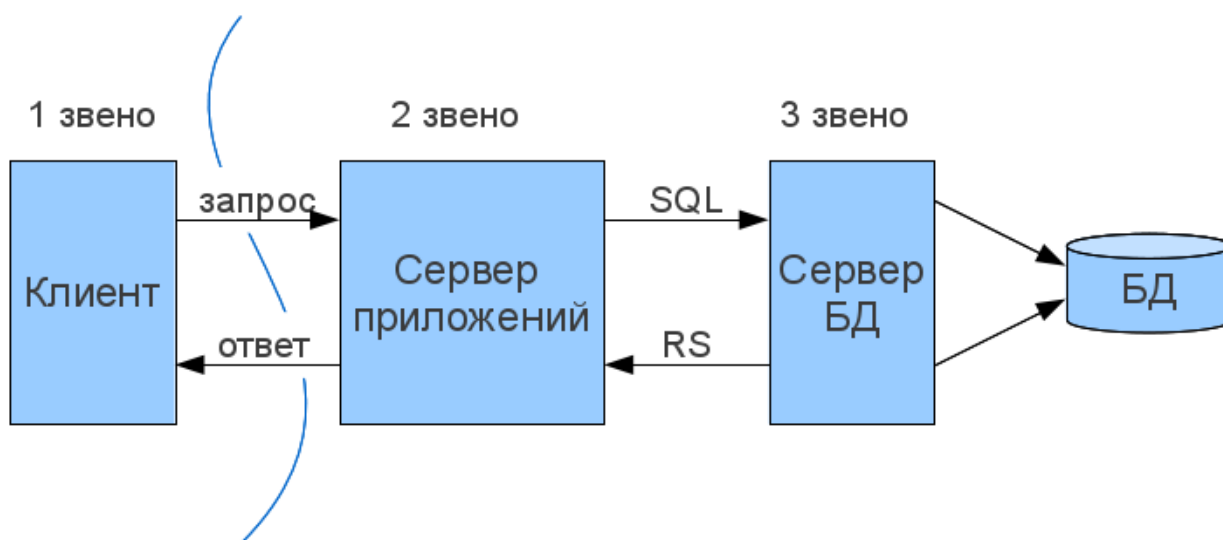


Рисунок 2.1 – Трехзвенная архитектура «клиент-сервер»

Для разработки сервера приложений был выбран язык программирования Javascript с использованием фреймворка VueJS для front-end части приложения и NodeJS – для back-end. Такое сочетание позволит максимально удобно взаимодействовать апи серверу и отдельному SPA клиентскому приложению между собой с высокой скоростью передачи данных. Для реализации back-end используется архитектура паттерна Model-View-Controller (рис. 2.2), что позволяет выстраивать модульную

архитектуру приложения и делать разработку программного обеспечения более структурированной.

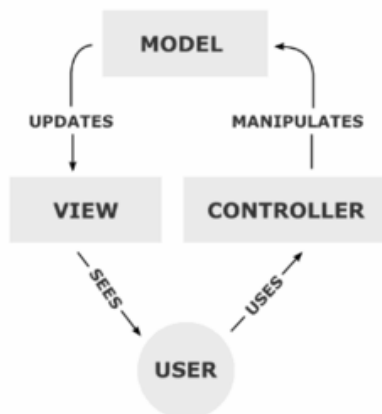


Рисунок 2.2 – Архитектурный паттерн Model-View-Controller (MVC)

Для взаимодействия с БД используется фреймворк BookshelfJS построенный на `querybuilder` `knex`. Такой подход позволяет делать разработку БД последовательной и безопасной. Наличие `seed` и `migrations` в проекте позволяет отслеживать изменения БД и поднимать новый, работоспособный инстанс базы в считанные секунды командами:

```
knex migrate:latest
```

```
knex seed:run
```

Так же, `knex` имеет и команду `rollback`, что позволит безопасно откатиться на любую более старую версию базы. Все миграции сохраняются в отдельную таблицу — `migrations` и содержат хронологию событий.

В качестве одной из Баз Данных используется удаленная PostgreSQL . В ней хранится вся информация о пользователях и сущностях в системе. Также для ускорения работы некоторых функций было принято решение перенести часть функционала в Хранимые процедуры базы данных.

Так же система имеет сервис рассылки писем. Для него используется сервис сторонний Sendgrid.

Таким образом система состоит из клиентской части (front-end), серверной (back-end) 2-х баз данных Redis и PostgreSQL и сервиса отправки писем (рис. 2.3).

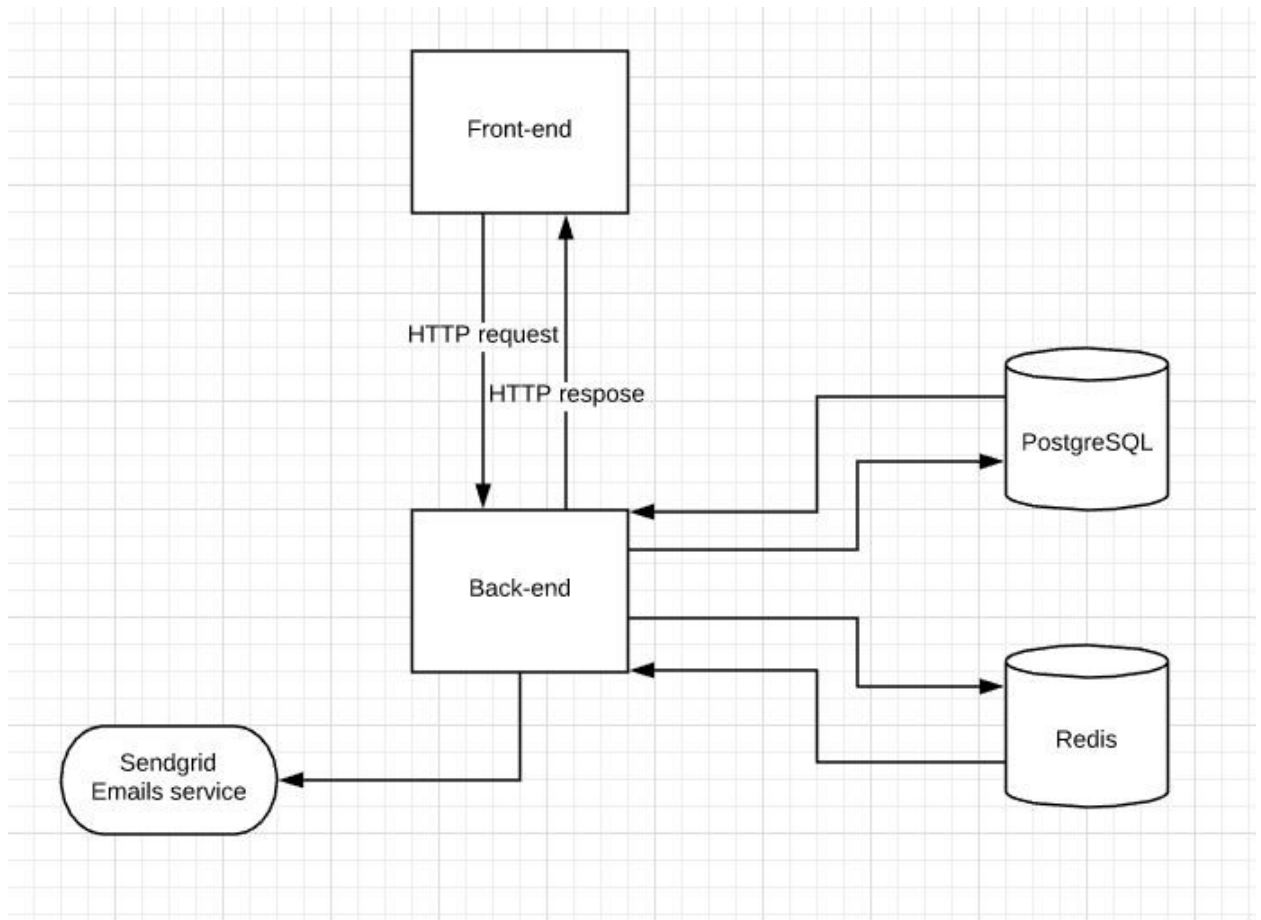


Рисунок 2.3. Схема взаимодействия компонентов системы.

2.2 Описание облачной платформы

Для публикации приложения в облаке использовалась облачная платформа Нероки. Она реализует облачную модель — платформа как сервис (PaaS). Все сервисы Нероки размещены на платформе облачных вычислений Amazon EC2, благодаря чему, в дальнейшем не составит труда выполнить интеграцию с любыми сервисами Amazon. Например: ELB, EC2, EC, VPC, RDS, AS. Эти сервисы, обычно, являются стандартным набором для таких

систем, но для реализации MVP достаточно использование Heroku так как — это более быстрый способ настройки необходимой архитектуры.

Платформа как услуга (PaaS) - это полная среда разработки и развертывания в облаке с ресурсами, которые позволяют вам поставлять все, от простых облачных приложений, до сложных корпоративных приложений с поддержкой облачных вычислений. Приобретение необходимых ресурсов у поставщика облачных услуг по принципу «оплата по мере необходимости» и получение доступ к ним через безопасное подключение к Интернету.

PaaS включает инфраструктурные серверы, хранилища и сети, но также и промежуточное ПО, средства разработки, бизнес-аналитики (BI), системы управления базами данных и многое другое. PaaS разработан для поддержки полного жизненного цикла веб-приложений: создания, тестирования, развертывания, управления и обновления.

Изначально рассматривались 2 сервиса облачных вычислений: Amazon и Heroku. У каждого из них были свои достоинства и недостатки:

Amazon — это веб-сервис, предоставляющий безопасные масштабируемые вычислительные ресурсы в облаке. Он помогает разработчикам, облегчая проведение крупномасштабных вычислений и поддержание инфраструктуры в облаке

Достоинства:

- предоставляет гибкую настройку CPU и памяти;
- очень популярен (имеет удобную SDK);
- имеет детальную документацию.

Недостатки:

- требует при регистрации ввести данные кредитной карты.

Heroku - облачная PaaS-платформа, поддерживающая ряд языков программирования. С 2010 года является дочерней компанией Salesforce.com. Heroku, одна из первых облачных платформ, появилась в июне 2007 года и изначально поддерживала только язык программирования Ruby, но на данный момент список поддерживаемых языков также включает в себя и Node.js.

Достоинства:

- очень проста в эксплуатации;
- создает весь необходимый ENV одной командой;
- очень популярен;
- имеет детальную и доступную для понимания документацию.

Недостатки:

- не имеет гибкой настройки вычислительных мощностей;
- использует все компоненты Amazon;
- дороже в использовании (в отличие от Amazon)

Из двух сервисов был выбран Heroku так как он является более быстрым в развертывании и не требует кредитной карты. Heroku можно использовать с Debit картой, что значительно упрощает процесс оплаты сервиса. Для MVP версии приложения, предоставляемых возможностей Heroku будет достаточно. В дальнейшем планируется интеграция с продуктами Amazon.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Система делиться на две части: front-end и back-end. Каждая часть является изолированной друг от друга и является отдельным Web приложением. Front-end использует фреймворк VueJS, который стал широкоиспользуемым благодаря своей легковесности и технологии Shadow DOM, которая позволяет на лету пересобирать DOM и дополнить его новым контекстом с ориентацией на каждое новое событие в коде. Front-end собирается с помощью библиотеки webpack делиться на чанки. Это позволяет организовать Lazy-loadin процесс и увеличить скорость загрузки приложения. Таким образом Web интерфейс будет состоять из «собираемой» статики, которая должна отдаваться клиенту простым HTTP сервером.

Вторая часть приложения — back-end (api server). Эта часть отвечает за взаимодействие пользователя с разными api системы, базой данных и другими third-party сервисами. На back-end реализована аутентификация путем Web JWT, которая позволяет определять пользователя в течении каждого запроса к back-end и верифицировать его права доступа. Так же, серверное приложение имеет публичное api и static пути.

На back-end, так же, реализована отправка E-mails уведомлений. Для отправки используется сервис SendGrid. Так, же в дальнейшем необходима будет настройка Cache Db для хранения сессий и временной информации. Такая БД должна взаимодействовать с оперативной памятью. Такой подход обеспечит высокую скорость записи и чтения контекстных данных.

Heroku предлагает интеграцию с системой SendGrid, и инфраструктурой PostgreSQL и Redis, что будет полезным в дальнейшем использовании.

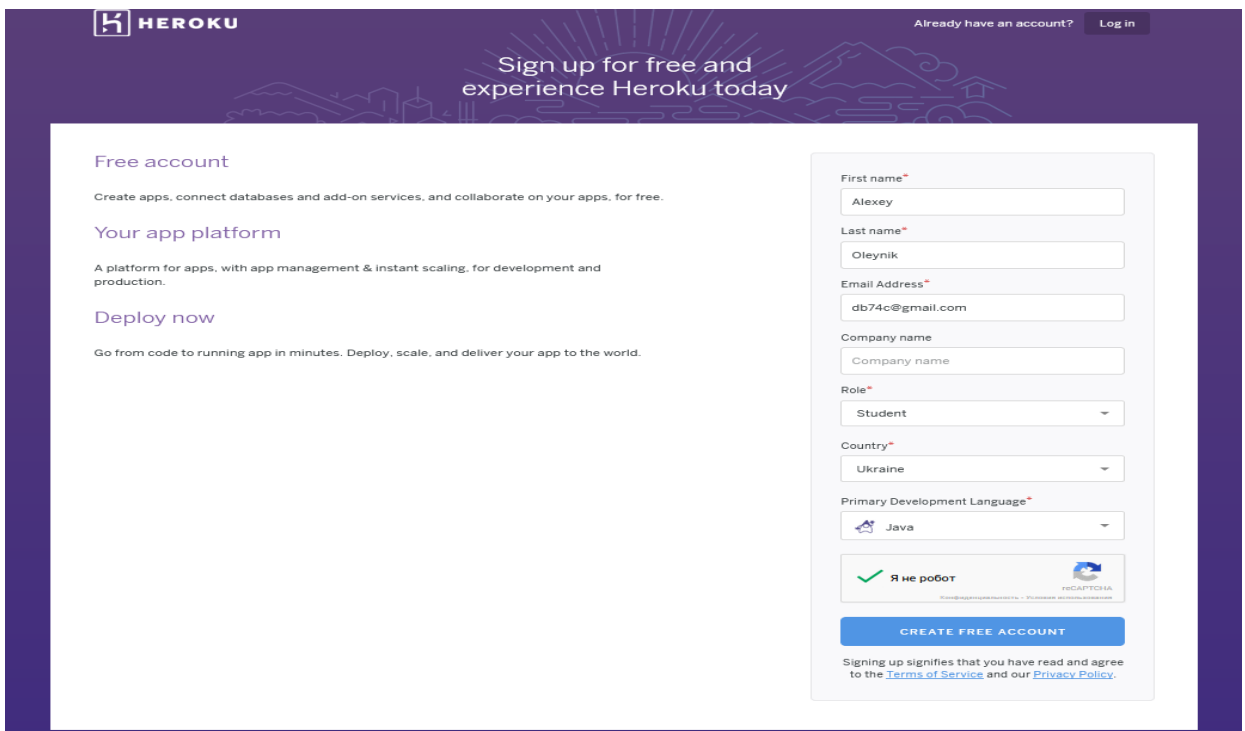
Таким образом, решено использовать 2 Dino Heroku, один для front-end, а другой для back-end. На back-end создать интеграцию с сервисом SendGrid и инфраструктурой PostgreSQL. Для front-end части необходимо будет настроить gits subtree для статического сервера, которое будет пересобиаться в процессе деплоя.

4 РЕГИСТРАЦИЯ В ОБЛАЧНОЙ ПЛАТФОРМЕ

Прежде всего, необходимо зарегистрироваться в облачной платформе. Для этого необходимо указать свои данные в форме регистрации на Heroku.

Далее – регистрация в облачной платформе. Она состоит из трех шагов:

- указание персональных данных;
- подтверждение своего аккаунта через email;
- указание и подтверждение пароля.



The screenshot shows the Heroku website's sign-up page. The header features the Heroku logo and a navigation bar with links for 'Already have an account?' and 'Log in'. The main heading is 'Sign up for free and experience Heroku today'. The 'Free account' section describes the platform's capabilities: 'Create apps, connect databases and add-on services, and collaborate on your apps, for free.' Below this, the 'Your app platform' section states: 'A platform for apps, with app management & instant scaling, for development and production.' The 'Deploy now' section encourages users to 'Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.' The registration form on the right includes fields for 'First name*' (filled with 'Alexey'), 'Last name*' (filled with 'Oleynik'), 'Email Address*' (filled with 'db74c@gmail.com'), 'Company name' (with a sub-field for 'Company name'), 'Role*' (a dropdown menu with 'Student' selected), 'Country*' (a dropdown menu with 'Ukraine' selected), and 'Primary Development Language*' (a dropdown menu with 'Java' selected). A CAPTCHA section with a green checkmark and the text 'Я не робот' is also present. At the bottom of the form is a blue button labeled 'CREATE FREE ACCOUNT'. Below the button, a small disclaimer states: 'Signing up signifies that you have read and agree to the [Terms of Service](#) and our [Privacy Policy](#).'

Рисунок 4.1 – Первый этап регистрации

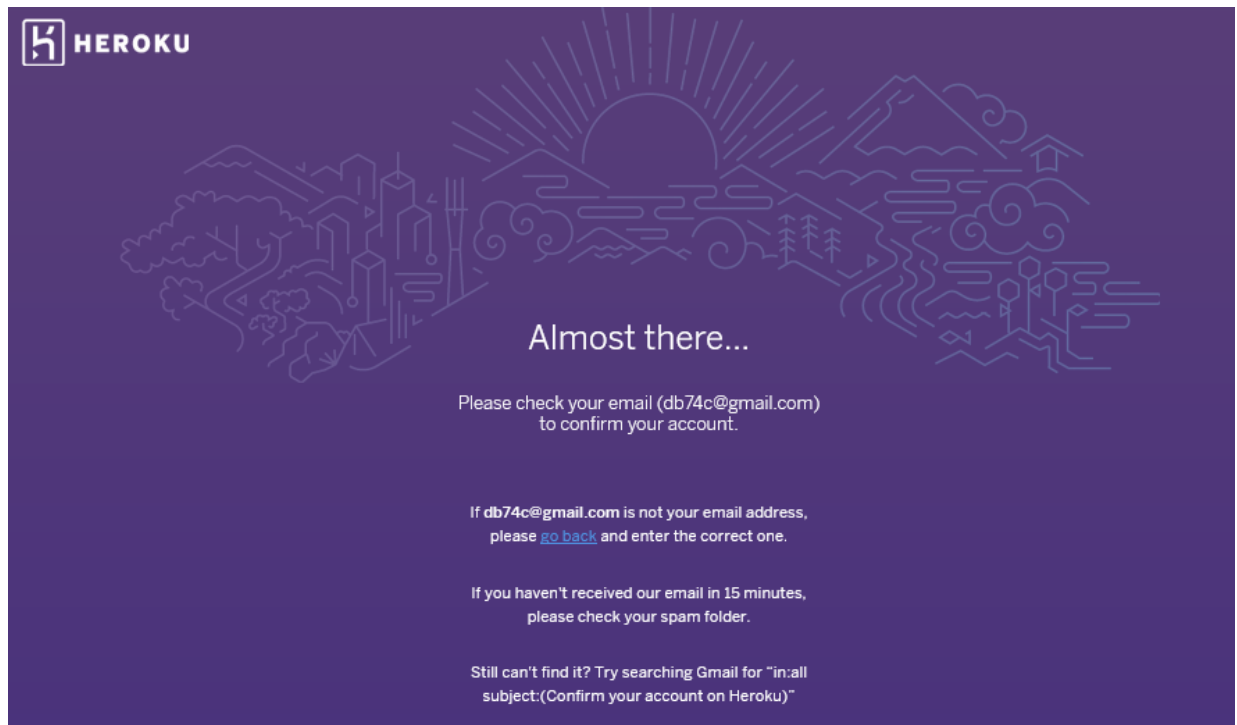


Рисунок 4.2 – Второй этап регистрации

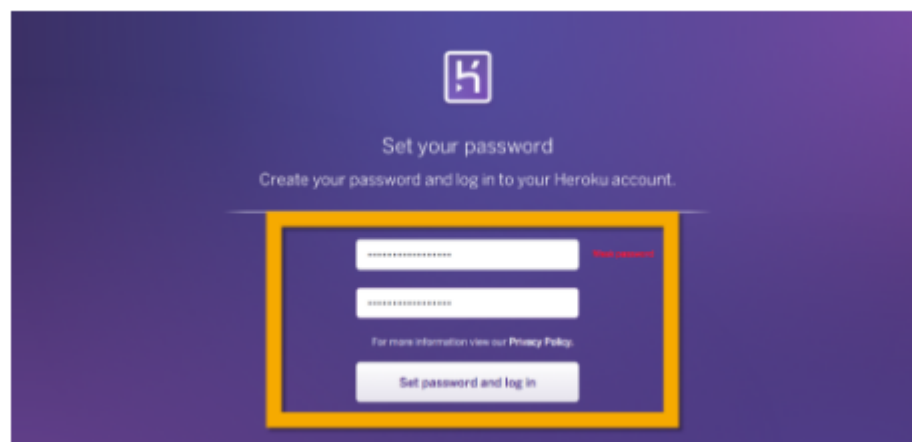


Рисунок 4.3 – Третий этап регистрации

После завершения регистрации, загружается стартовая страница портала:

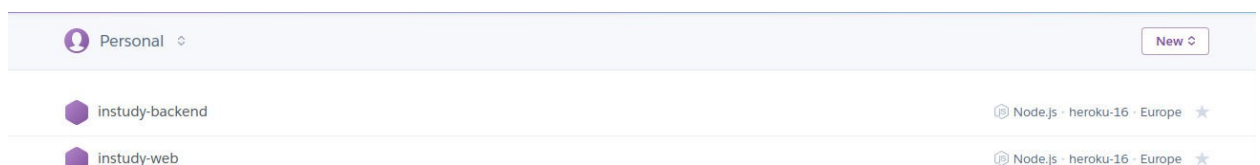


Рисунок 4.4 – Начальная страница Heroku

После Завершения Регистрации необходимо настроить все Add-ons. Это можно сделать в соответствующем разделе в Heroku-app (рис. 4.5).

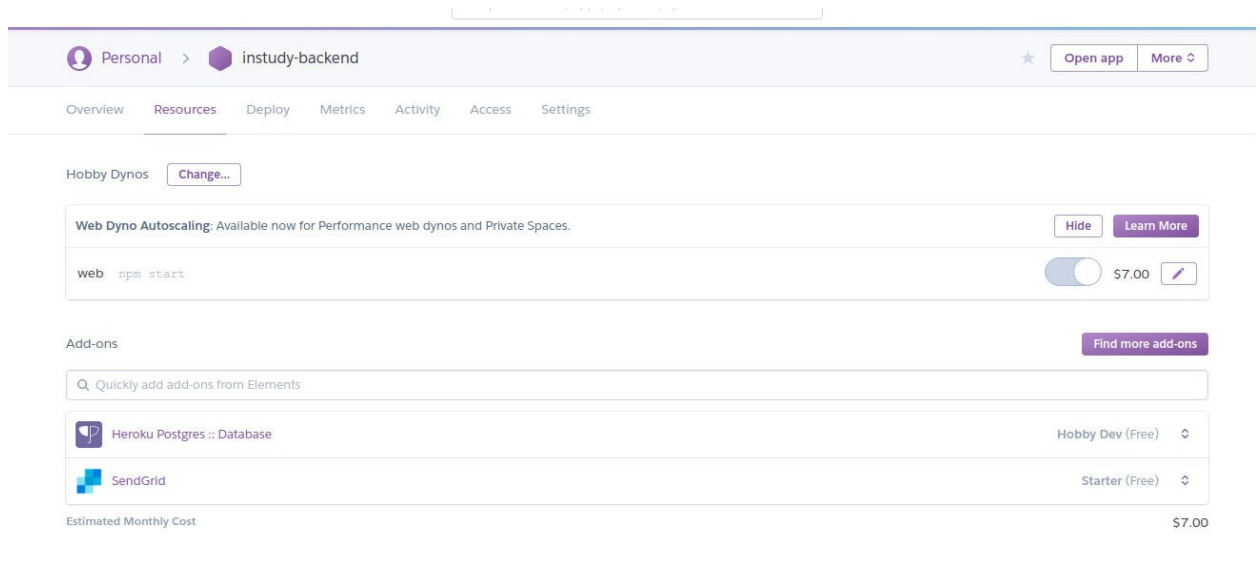


Рисунок 4.5 – Страница настройки Add-ons с настроенным SendGrid и PostgreSQL.

Heroku PostgreSQL – это база данных SQL которая размещена в облаке. Для получения доступом к ней необходимо нажать на значек addon и перейти в нстройки. Там будет вся необходимая информация по настройке данного дополнения

Sendgrid – это сторонний сервис предназначенный для отправки различных e-mail писем. Система его использует ля отправки приглашений и уведомлений. Для получения доступов необходимо перейти на вкладку с настройками и найти нужные настройки из всех существующих ENV.

5 НАСТРОЙКА ДЕПЛОЯ СИСТЕМЫ

Система состоит из двух репозиториях `front-end` и `back-end` которые никак не связаны друг с другом. Клиентская часть имеет свое собственное `subtree` предназначенное для хранения статики и релизных версий приложения (рис. 5.1).

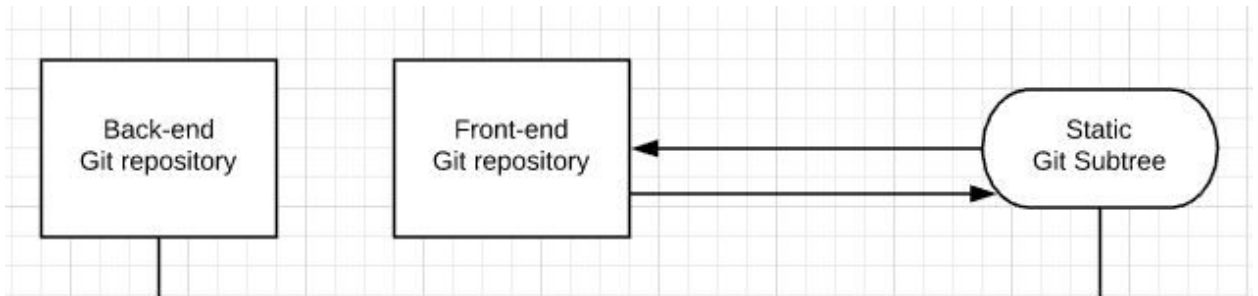


Рисунок 1. Схема взаимодействия репозиториях и их `subtree`

Для взаимодействия с `Heroku` необходимо локально установить `Heroku CLI` и авторизоваться в нем используя следующую команду:

```
$ heroku login
```

Далее будет необходимо ввести свой логин и пароль для `Heroku`.

Следующим шагом необходимо добавить новый `remote` в локальный `git`. Используя следующую команду был добавлен удаленный репозиторий `Heroku`, с которого будет собираться будущее `back-end` приложение:

```
$ heroku git:clone -a instudy-backend
```

Аналогично для `front-end`:

```
$ heroku git:clone -a instudy-web
```

Так же, для `Web` части приложения необходимо создать новый `Http` сервер для отдачи статических файлов. Для добавления результирующего кода в репозиторий создадим новое `git subtree`:

```
$ git subtree add --prefix dist heroku master
```

Таким образом было создана связь между репозиториями приложения и удаленными репозиториями Heroku. Каждый репозиторий связан со своим App (рис. 5.2)

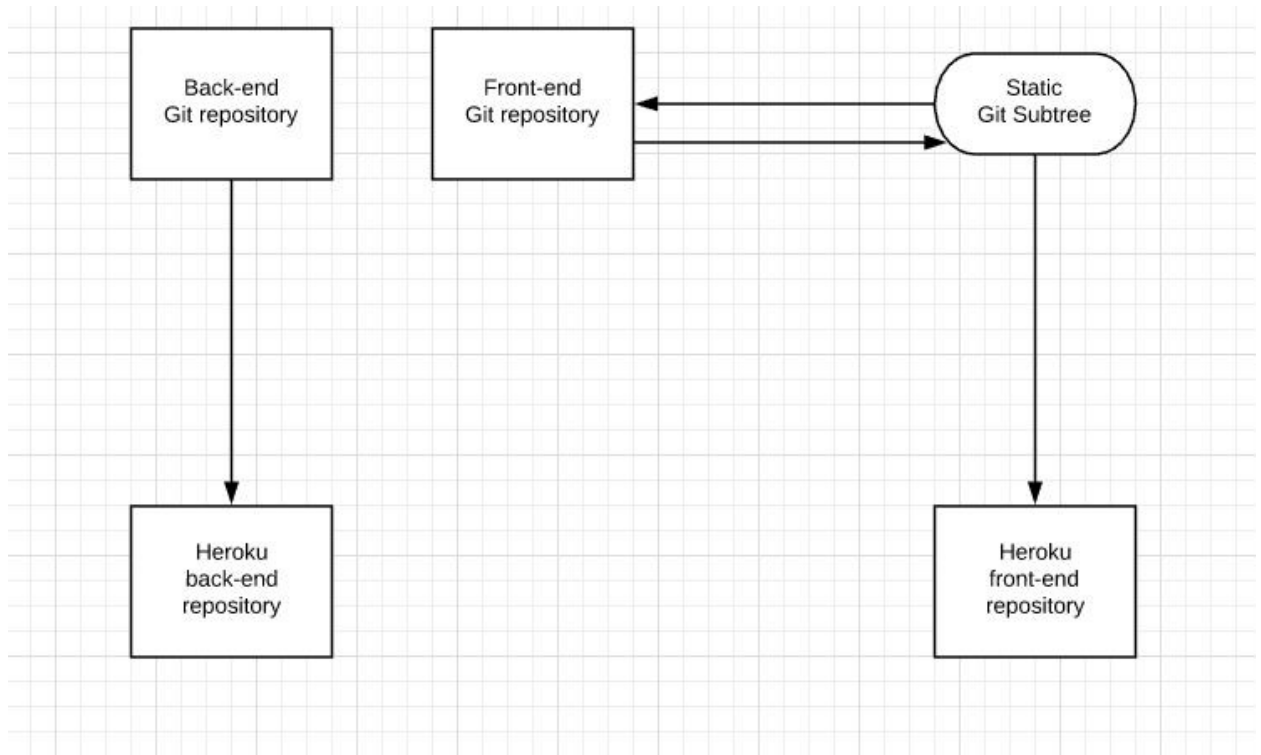


Рисунок 5.2 Схема связи репозитория Heroku и приложения.

В папке dist хранятся все статические файлы собранные webpack в чанки. Так же, в этой папке хранятся сервер статики. Для дальнейшо деплоя создаем коммит в репозиторий:

```

$ git add .
$ git commit -m «initial commit»
$ git push
$ git push heroku master
$ git subtree push --prefix dist heroku master
  
```

В дальнейшем, для удобства, последнюю команду добавим в файл package.json и назовем ее «deploy». Таким образом деплой новых изменений можно запускать простой командой :

```

$ npm run deploy
  
```


Конфигурационные ENV для доступа к PostgreSQL и SendGrid можно получить из раздела Setting. Для удобства они будут перенесены в отдельный .json файл. Это позволит в дальнейшем не использовать .sh скрипт для экспортирования переменных под каждое окружение, а использовать только одну переменную окружения: NODE_ENV.

Деплой back-end приложения осуществляется аналогично:

```
$ git add .  
$ git commit -m «initial commit»  
$ git push  
$ git push heroku master
```

Таким образом, был настроен удобный и простой в использовании деплой двух приложений в облако путем использования PaaS решения Heroku.

6 ТЕСТИРОВАНИЕ СИСТЕМЫ

Благодаря настроенной инфраструктуре и админки Heroku можно видеть логи деплоя (рис 6.1) и все необходимые метрики (рис 6.2).

На графиках видны показатели описывающие текущую загрузку приложения. На данный момент загруженность минимальная.

Стоит отметить, что через меню настроек метрик можно настроить множество других метрик и отображать наиболее необходимую информацию.

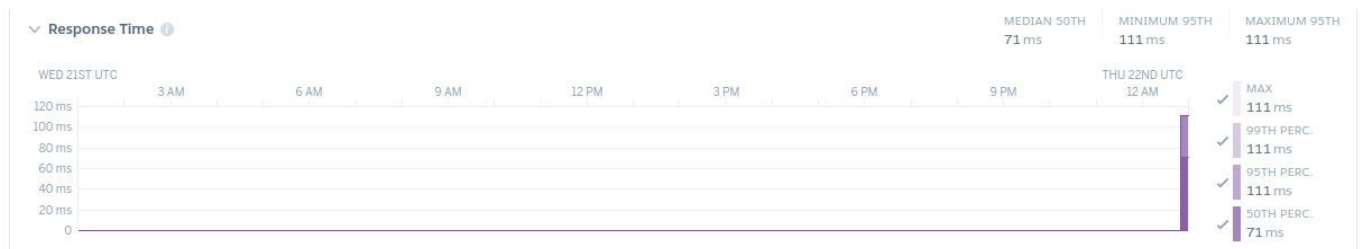


Рисунок 6.1 – График времени ответа сервера.

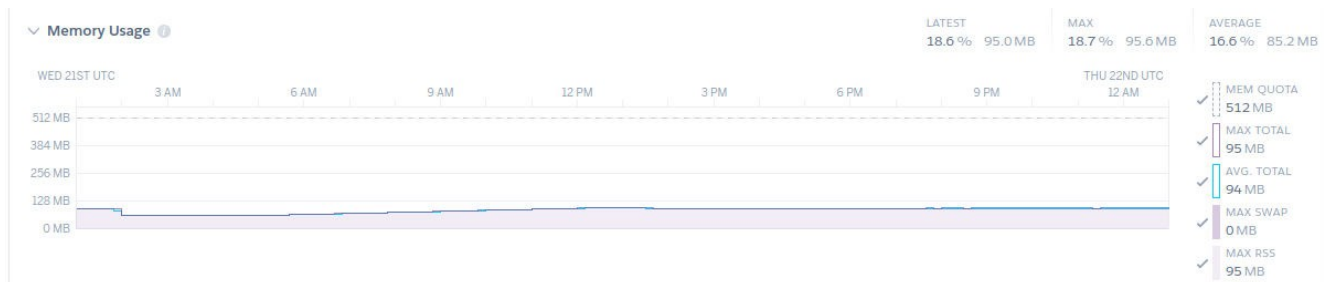


Рисунок 6.2 – График использования оперативной памяти.

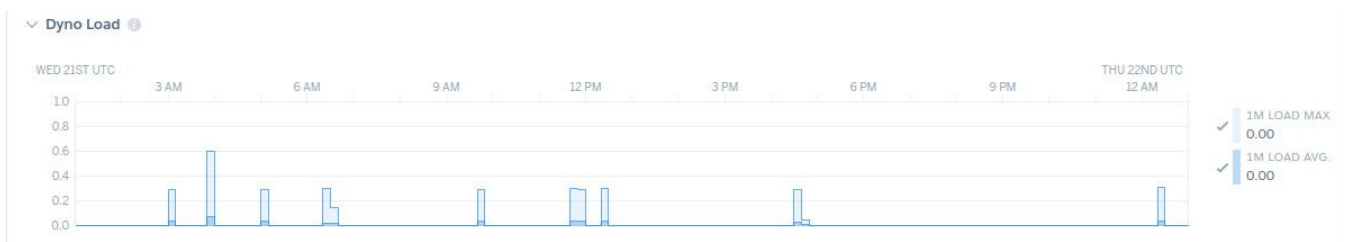


Рисунок 6.3 – График использования CPU.

Каждый процесс деплоя сопровождается логированием, его можно просмотреть на странице логов (рис 6.4). На этой странице, так же, можно увидеть все возникшие ошибки во время заливки.

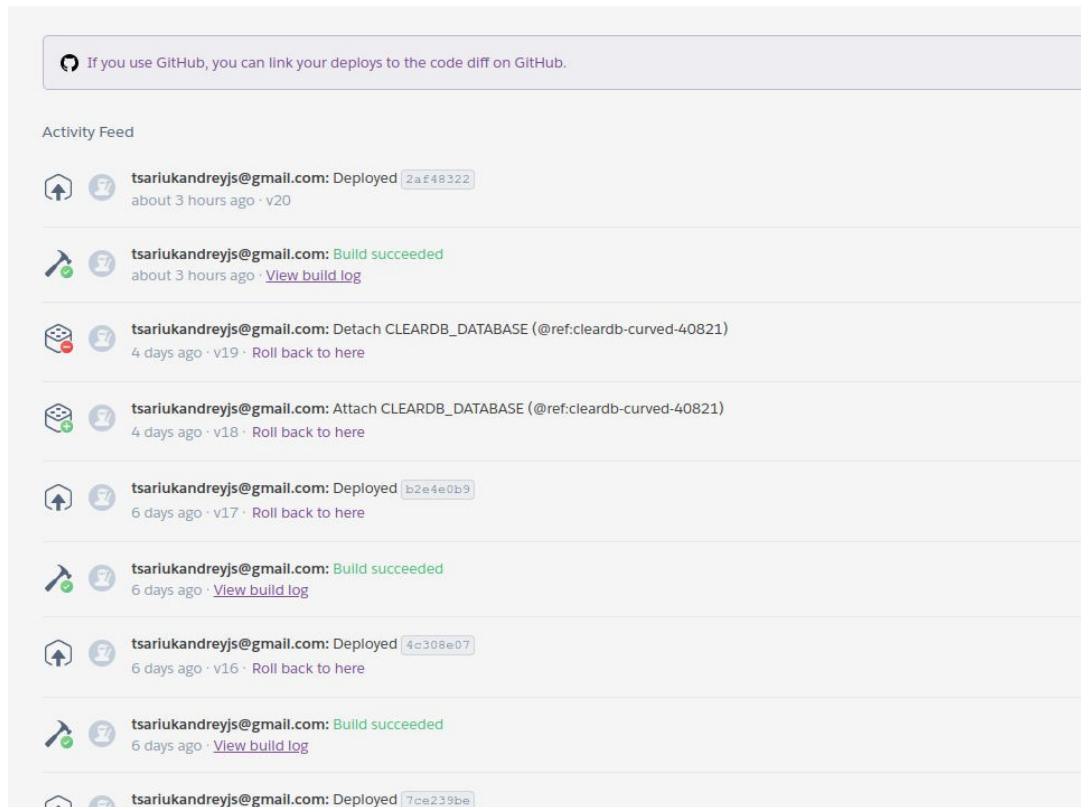


Рисунок 6.4 – Страница логов процесса деплоя.

Так же, логи приложения можно посмотреть выполнив команду в терминале, она вернет все недавние логи приложения, ошибки и информацию о них:

```
$ heroku logs
```

Нажав на кнопку “Open App” можно перейти на ссылку на новое, зашруженное приложение и проверить его работоспособность. В новом браузерном окне откроется приложение с формой авторизации. (рис 6.5) Введя все необходимые данные мы проверим работу api server, БД, и web части приложения (рис. 6.6).

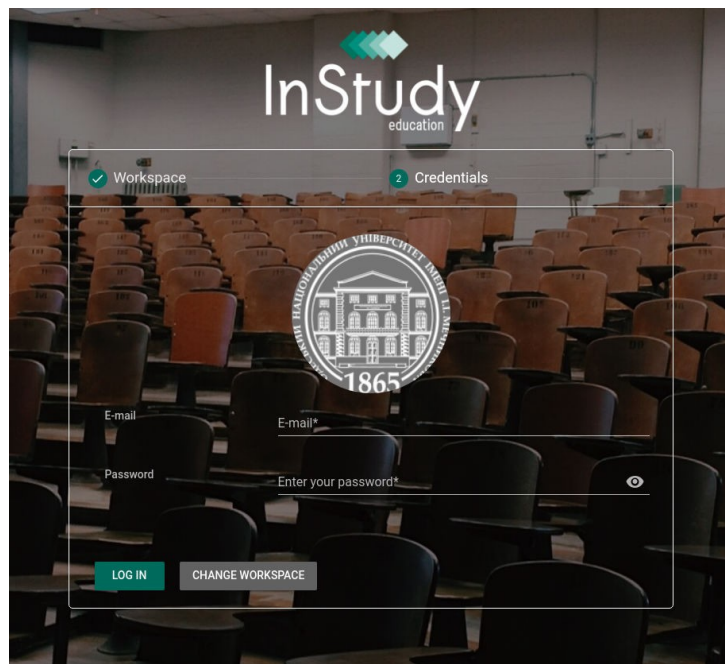


Рисунок 6.5 – Страница авторизации приложения.

Moks						
Dashboard	Users	Students	Subjects	Books	Files	Invites
Search by E-mail						
<input type="checkbox"/>	id	↑ E-mail	User	Token	Created	
<input type="checkbox"/>	2	testcortkx1q@gmail.com	undefined undefined	0A67Ae08k8GZeiSFhZNIAWZB	2018-03-05T22:45:25.720Z	
<input type="checkbox"/>	1	testcortkx1q@gmail.com	undefined undefined	0A679ZFfx7H0IKLLzKDSNg72t	2018-03-05T22:41:07.664Z	
						Rows per page: 5

Рисунок 6.6 – Страница списка приглашений в систему.

ВЫВОДЫ

Облачные вычисления тесно вошли в нашу жизнь и ежедневное использование компьютера, подключенного к сети Интернет не обходится без них. Новые технологии принесли много нового, в частности, перемены на ИТ-рынке и появление новых услуг, сервисов и платформ, которые являются основополагающими для новых бизнес-моделей. Развитие глобальной сети, а также рост потребления контента, связанный также с бурным ростом мобильной техники, потребовал от производителей создания новых масштабируемых и гибких систем, которые бы позволили наилучшим образом подстраиваться под растущие запросы, а также предоставили новые пути доставки контента и построения инфраструктур.

Основные преимущества таких систем, такие как масштабируемость, мультитенантность, эластичность, а также, что очень важно, оплата за использование выдвигают их на первый план.

Компания Salesforce предлагает свою облачную платформу – Heroku. Она включает в себя поддержку множества дополнений, таких как хранилище данных, веб-сайты, виртуальные машины, мобильные и облачные сервисы.

Таким образом, создание и разворачивание веб-приложений при помощи Heroku предоставляет разработчику много возможностей и преимуществ. Примером некоторых возможностей является разработанная для данного курсового проекта интернет-игра «Ships». Благодаря возможностям Heroku поставленная задача была выполнена.

ЛИТЕРАТУРА

1. Т. Коннолли, К. Бегг, А. Страчан Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – 2-е изд., испр. и доп. – М. : Вильямс, 2001. – 1111 с.
2. PaaS [Электронный ресурс] – Режим доступа:
<https://aws.amazon.com/ru/types-of-cloud-computing/>
3. Heroku Documentation[Электронный ресурс] – Режим доступа:
[:/devcenter.heroku.com/categories/reference](https://devcenter.heroku.com/categories/reference)
4. EAV/CR[Электронный ресурс] – Режим доступа:
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC61391/>

ПРИЛОЖЕНИЕ А

СЕРВЕР СТАТИКИ WEB ПРИЛОЖЕНИЯ

```
const express = require('express')
const path = require('path')
const serveStatic = require('serve-static')
app = express()
app.use(serveStatic(__dirname))
const port = process.env.PORT || 5000
app.listen(port);
app.use((req, res) => {
  res.sendFile('./index.html', { root: __dirname });
});
console.log('server started ' + port)
```

ПРИЛОЖЕНИЕ Б

СКРИПТЫ СОЗДАНИЯ БД

```

"use strict";

exports.up = function (knex) {
  return knex.schema
    .createTable("workspaces", function (table) {
      table.increments("id").primary();
      table
        .string("name")
        .nullable()
        .unique();
      table
        .timestamp("created_at")
        .defaultTo(knex.fn.now())
        .nullable();
      table
        .timestamp("updated_at")
        .defaultTo(knex.fn.now())
        .nullable();
    })
    .createTable("users", function (table) {
      table.increments("id").primary();
      table
        .string("email")
        .nullable()
        .unique();
      table
        .string("password");
      table
        .bigInteger("workspace_id")
        .nullable()
        .references("workspaces.id")
        .onDelete("CASCADE");
      table
        .timestamp("created_at")
        .defaultTo(knex.fn.now())
        .nullable();
      table
        .timestamp("updated_at")
        .defaultTo(knex.fn.now())
        .nullable();
    })
    .createTable("roles", function (table) {
      table.increments("id").primary();
      table
        .bigInteger("user_id")

```



```

    .nullable()
    .references("users.id")
    .onDelete("CASCADE");

table.bigInteger("role_id").nullable();
table.string("role_type").nullable();

table.unique(["user_id", "role_id", "role_type"]);
table
    .timestamp("created_at")
    .defaultTo(knex.fn.now())
    .nullable();
table
    .timestamp("updated_at")
    .defaultTo(knex.fn.now())
    .nullable();
})

.createTable("students", function (table) {
    table.increments("id").primary();
})
.createTable("lectors", function (table) {
    table.increments("id").primary();
})
.createTable("admins", function (table) {
    table.increments("id").primary();
})
.createTable("profiles", function (table) {
    table.increments("id").primary();
    table
        .bigInteger("user_id")
        .nullable()
        .references("users.id")
        .onDelete("CASCADE")
        .unique();
    table.string("gender");
    table.string("fname");
    table.string("lname");
    table.string("personal_phone");
    table.string("work_phone");
    table.string("personal_email");
    table.string("work_email");
    table
        .timestamp("created_at")
        .defaultTo(knex.fn.now())
        .nullable();
    table
        .timestamp("updated_at")
        .defaultTo(knex.fn.now())
        .nullable();

```

```

    })
    .createTable("settings", function (table) {
      table.increments("id").primary();
      table
        .bigInteger("user_id")
        .notNullable()
        .references("users.id")
        .onDelete("CASCADE");
      table
        .string("language")
        .notNullable()
        .defaultTo("eng");
      table
        .timestamp("created_at")
        .defaultTo(knex.fn.now())
        .notNullable();
      table
        .timestamp("updated_at")
        .defaultTo(knex.fn.now())
        .notNullable();
    });
  };

```

```

exports.down = function (knex) {
  return knex.schema
    .dropTable("workspaces")
    .dropTable("roles")
    .dropTable("students")
    .dropTable("lectors")
    .dropTable("admins")
    .dropTable("profiles")
    .dropTable("settings")
    .dropTable("users");
};

```

ПРИЛОЖЕНИЕ В

СТРАНИЦА АВТОРИЗАЦИИ

```

<template>

<v-layout class="background background-1" dark>
  <div class="background-muted">
    <div class="logo-container">
      
    </div>
    <div class="login-layout">
      <v-stepper v-model="stepper" class="rounded transparent" dark>
        <v-stepper-header class="header-border">
          <v-stepper-step step="1" :complete="stepper > 1"><span class="stepper-title">Workspace</span>
        </v-stepper-step>
        <v-divider></v-divider>
        <v-stepper-step step="2" :complete="stepper > 2"><span class="stepper-
title">Credentials</span></v-stepper-step>
        <v-divider> </v-divider>

      </v-stepper-header>
      <v-stepper-items>
        <v-stepper-content step="1">
          <v-form v-model="validWorkspace" class="transparent static-height" v-
on:submit.prevent="submitWorkspace">
            <v-card class="elevation-0 transparent">
              <v-text-field label="Workspace" v-model="workspace_name" required></v-text-field>
            </v-card>
            <v-btn :disabled="!!!workspace_name" color="primary" type="submit">Continue</v-btn>
          </v-form>
        </v-stepper-content>
        <v-stepper-content step="2">
          <div>
            <v-avatar :tile="false" :size="200" class="workspace-logo grey darken-4" color="secondry">
              <img :class="{ 'visible-logo': stepper > 1 }" class="grey darken-1"
:src="workspace.avatar.publicPath" alt="">
            </v-avatar>
          </div>
          <SignInForm ref="SignInForm" @submit="signIn()"> </SignInForm>
          <v-btn color="primary" @click="signIn()">Log In</v-btn>
          <v-btn color="secondary" @click="changeWorkspace()">Change Workspace</v-btn>
        </v-stepper-content>
      </v-stepper-items>
    </v-stepper>
  </div>
</div>

```

```
</v-layout>
</template>
```

```
<script>
import { Api, ApiService } from "../../services";
import jwt_decode from "jwt-decode";
import SignInForm from "./SignInForm.vue";
export default {
  name: "hello",
  data() {
    return {
      loader: null,
      loading3: false,
      stepper: 0,
      workspace_name: "",
      validWorkspace: false,
      workspace: {
        avatar: {}
      }
    };
  },
  components: {
    SignInForm
  },
  watch: {
    loader() {
      const l = this.loader;
      this[l] = !this[l];
      setTimeout(() => (this[l] = false), 2000);
      // @click.native="loader = 'loading3' :disabled='loading3'
      this.loader = null;
    }
  },
  methods: {
    submitWorkspace() {
      console.log("Workspace", this.workspace_name);
      if (this.workspace_name)
        ApiService.Auth.initializeWorkspace(this.workspace_name).then(res => {
          this.workspace = res;
          console.log(this.workspace);
          this.stepper = 2;
        });
    },
    changeWorkspace() {
      this.stepper = 1;
    },
    signIn() {
      console.log("fooooo");
      let signInData = this.$refs.SignInForm.form();
      if (signInData) {
```

```

ApiService.Auth.signIn(
Object.assign(
  {},
  {
workspace_id: this.workspace.id
  },
  signInData
)
).then(res => {
  console.log("res", res);
  let isAdmin = res.user.short_roles.indexOf("admins") !== -1;
  let isStudent = res.user.short_roles.indexOf("students") !== -1;
  let isLector = res.user.short_roles.indexOf("lectors") !== -1;
  switch (true) {
  case isAdmin:
    return this.$router.push(`/${this.workspace.name}/admin/users`);
    break;
  case isLector:
    return this.$router.push(`/${this.workspace.name}/lectors/users`);
    break;
  case isStudent:
    return this.$router.push(`/${this.workspace.name}/student/users`);

  default:
    return this.$router.push(`/sign-in`);
    break;
  }
});
}
}
}
}
};
</script>

```

```

<!-- Add "scoped" attribute to limit CSS to this component only -->

```

```

<style>
.logo-container {
text-align: center;
}
.logo {
max-width: 300px;
}

```

```

.project-logo {
font-size: 4rem;
color: white;
margin: 0 auto;
display: block;
}

```

```
.background {
background-position: center;
background-repeat: no-repeat;
background-size: cover;
}

.background-1 {
background-image: url(/static/images/in-Study-background-1.jpeg);
}

.background-2 {
background-image: url(/static/images/in-Study-background-2.jpeg);
}

.background-3 {
background-image: url(/static/images/in-Study-background-3.jpeg);
}

.background-4 {
background-image: url(/static/images/in-Study-background-4.jpeg);
}

.bacground-muted {
background-color: rgba(0, 0, 0, 0.5);
width: 100%;
}

.login-layout {
padding-left: 30%;
padding-right: 30%;
padding-bottom: 50px;
}

.rounded {
border-radius: 5px;
border: 1px solid white !important;
}

.workspace-logo {
margin-top: 1vh !important;
margin: 0 auto;
display: block !important;
visibility: hidden;
}

.visible-logo {
visibility: visible;
}

.static-height {
```

```
/* height: 70vh; */  
}  
  
.stepper-title {  
font-size: 2vh;  
}  
  
.header-border {  
border-bottom: 1px solid white;  
}  
  
@media (max-width: 1024px) {  
.login-layout {  
padding-top: 10vh;  
padding-left: 10px;  
padding-right: 10px;  
}  
}  
</style>
```