МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №1

По дисциплине «Web-программирование»

Хостинг веб-приложения на сервисе Heroku

Выполнил студент группы *M33122*:
Федотенко Николай Владимирович

Проверил:
Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

Цель работы:

Включение имеющейся клиентской части (лабораторных работ прошлого семестра) внутрь нового приложения, которое будет развёрнуто в сервисе облачного хостинга Heroku.

Краткие теоретические сведения:

- **Heroku** – это облачная PaaS-платформа (*платформа как сервис: предоставляет пользователю определённые функции и доступ к ПО, при этом её инфраструктура скрыта),* поддерживающая ряд языков программирования. Была выбрана в качестве сервиса для развёртки лабораторных работ, так как в рамках наших требований она бесплатна, заранее сконфигурирована и проста в использовании.

- **Node.js** – это открытая 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, и даже программировать микроконтроллеры. В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.
  В состав Node.js входит собственный менеджер пакетов **npm**, который предоставляет возможности установки, публикации и поиска пакетов из репозитория *npmjs.com*.

- **TypeScript** – это язык программирования, позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js. TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов (как в традиционных ООП языках), а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость и т. д.

- **MVC** – это архитектурный шаблон (паттерн), разделяющий данные приложения и управляющей логики на 3 отдельных независимо модифицируемых компонента: **модель** (*набор классов, реализующих всю бизнес-логику*), **представление** (*набор классов и шаблонов, отвечающих за взаимодействия с пользователями*) и **контроллер** (*связующее звено между первыми двумя компонентами*).
  Такое разделение Web-приложения на части упрощает структуру приложения за счет более строго разделения его уровней. Логика пользовательского интерфейса располагается в представлении, логика ввода-вывода – в контроллере, а бизнес-логика – в модели. Достигается полное отделение логики работы приложения от представления данных. Разработчик получает полный контроль над формируемым HTML-документом. Облегчается задача выполнения тестирования приложения.

- **REST API** – это веб-сервисы, которые позволяют отправлять запросы к ресурсам по URL-путям. Указывается операция, которую необходимо выполнить, с помощью пути (например, *GET*, *POST*, *PUT*, *DELETE*). Как и в случае с другими API веб-служб, запросы и ответы передаются через HTTP через Интернет, и серверы, принимающие запросы, не зависят от языка запроса (необязательно, чтобы он был определённым языком программирования). Ответы обычно возвращаются в формате JSON или XML. API REST имеют много разных путей (*Endpoints*) с различными параметрами, которые можно настраивать для определения желаемых результатов.

- **NestJS** – это MVC-подобный фреймворк для создания эффективных, масштабируемых серверных приложений Node.js. Он использует JavaScript, построен и полностью поддерживает TypeScript (но при этом позволяет разработчикам писать код на чистом JavaScript) и сочетает в себе элементы объектно-ориентированного программирования (ООП), функционального программирования (ФП), а также функционально-реактивного программирования (ФРП). Под капотом Nest использует надёжные фреймворки HTTP-серверов (по умолчанию **Express**), может быть также настроен и для фреймворка **Fastify**.
  Nest обеспечивает уровень абстракции по сравнению с этими распространенными платформами Node.js (*Express*/*Fastify*), но также предоставляет свои API-интерфейсы непосредственно разработчику. Это даёт разработчикам свободу использовать множество сторонних модулей, доступных для базовой платформы.

Ход выполнения работы:

Данная лабораторная работа выполнена в операционной системе *macOS*.

Используемая IDE: *WebStorm 2021.3* (by JetBrains)

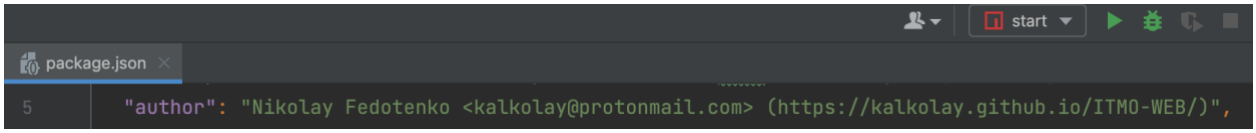
1. Установка необходимого ПО и компонентов:

```
brew install node
npm i @nestjs/cli -g
brew tap heroku/brew && brew install heroku
```

2. Создание приложения NestJS с помощью Nest CLI:

```
nest new itmo-web-2
```

3. Добавление авторства в конфигурационный файл *package.json*:



4. Проверка работоспособности приложения:

```
npm run start # http://localhost:3000 -> GET “Hello, World!”
```

5. Настройка запуска на произвольном номере порта через переменную окружения:

```
npm i --save @nestjs/config # установим нужные зависимости
```

app.module.ts

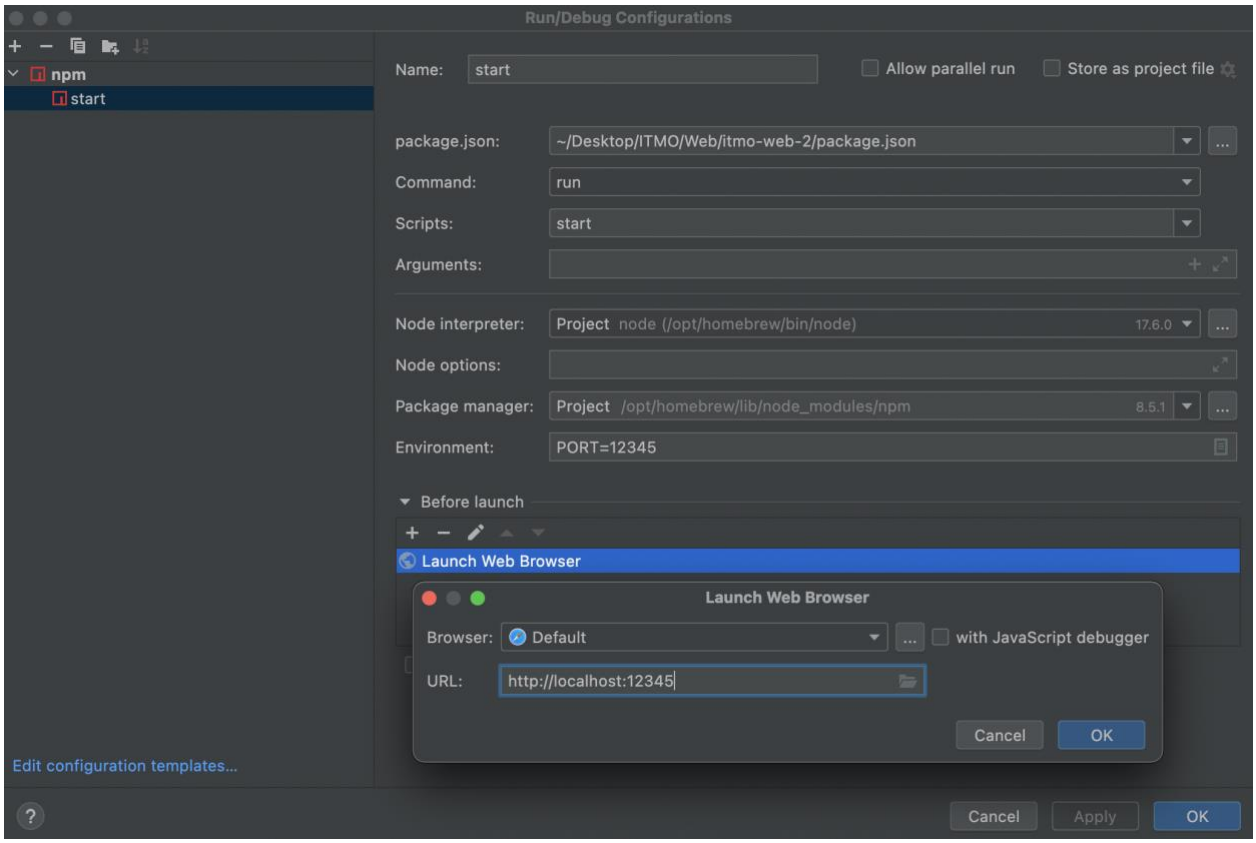
```ts
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module( metadata: {
  imports: [ConfigModule.forRoot()],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

main.ts

```ts
import { NestFactory } from '@nestjs/core';
import { ConfigService } from '@nestjs/config';
import { NestExpressApplication } from '@nestjs/platform-express';
import { join } from 'path';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(
    AppModule
  );

  app.useStaticAssets(join(__dirname, '..', 'public'));

  const port = app.get(ConfigService).get<number>('PORT') || 12345;
  await app.listen(port);
}
bootstrap();
```

- **ConfigModule** – *модуль конфигурации Nest*, загрузка и парсинг *.env* файла;
- **ConfigService** – *сервис конфигурации Nest*, доступ к результатам парсинга.

6. Проверка корректного назначения порта при запуске:

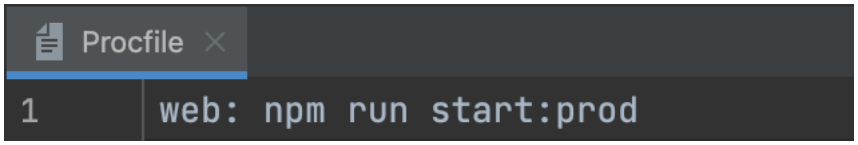


http://localhost:12345 действительно отображает “Hello, World!”

7. Создание инстанса приложения на хостинге Heroku:

```
heroku apps:create itmo-web-fedotenko --region eu
```

8. Указание входной точки для Heroku:



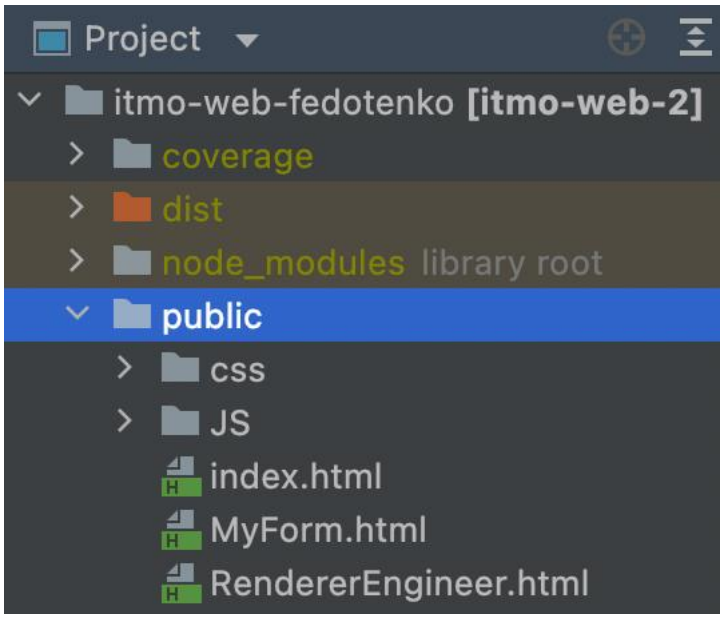
9. Загрузка кода приложения на сам хостинг:

```
heroku git:remote
```

```
git add . && git commit -m "Initial commit"
git push heroku master
```

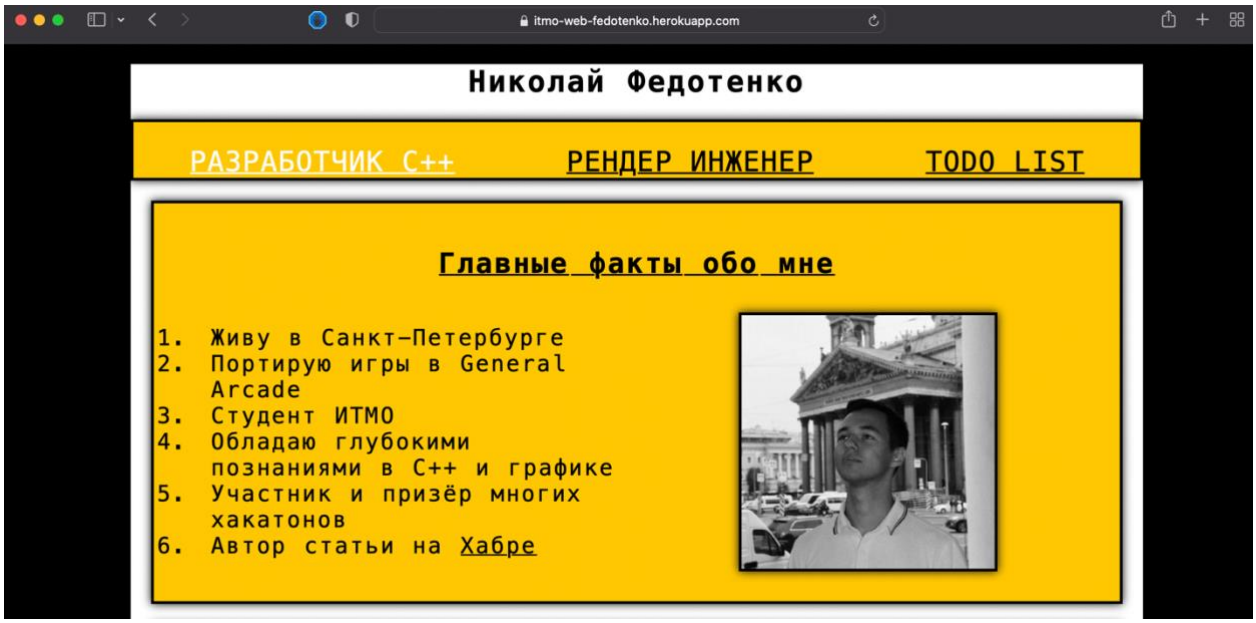
```
heroku apps:open # https://itmo-web-fedotenko.herokuapp.com
```

10. Размещение статических ресурсов:




```
const app = await NestFactory.create<NestExpressApplication>(
  AppModule
);

app.useStaticAssets(join(__dirname, '..', 'public'));
```

Проверка результата (https://itmo-web-fedotenko.herokuapp.com/index.html):



Вывод:

Я научился пользоваться сервисом облачного хостинга Heroku (с помощью браузера и Heroku CLI) и осуществлять развёртку приложений со включением имеющейся клиентской части. Я также ознакомился с программной платформой Node.js и её фреймворком NestJS.