



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ, ФИЛИАЛ ПЛОВДИВ

ФАКУЛТЕТ ПО ЕЛЕКТРОНИКА И АВТОМАТИКА

---

# ДИПЛОМНА РАБОТА

НА

Борис Василев Тумбев Фак. № 367157

**Специалност:** Компютърни системи и технологии

**Образователно-квалификационна степен:** бакалавър

**Тема:**

Проектиране и разработка на система за разпределение на студентско натоварване.

**Научен ръководител:**

доц. д-р Н. Каканаков

Пловдив 2020

## Задание

Съдържание.....	2
Увод.....	4
Глава I. Обзор - състояние на проблема по литературни данни;.....	5
1.Анализ на темата. Цели и задачи.....	5
1.1Анализ на темата.....	5
1.2. Цели и задачи.....	10
2. Използвани технологии.....	10
2.1 <b>PyCharm</b> .....	10
2.2 <b>GIT</b> .....	11
2.3 Django	
REST.....	12
2.4 Python.....	13
2.5 Java Script.....	15
2.6 React JS.....	17
2.7 JSON.....	17
2.8 AXIOS.....	17
2.9 REDUX.....	17

Глава II. Теоретично решение на поставената задача;	27
1. Цялостна архитектура	27
2. Данни предоставени от информационната система	30
Глава 3 – Описание на софтуерната част;	32
1. Модели	32
2. View-та	32
3. Serualizers	32
4. Reducers	32
Глава 4 – Функционално тестване;	37
Глава 5 – Приложимост на дипломната работа;	41
Глава 6 – Икономическа оценка на резултатите и техническа ефективност;	41
1.Заклучение	41
2.Извод	42
Източници	43
Приложение	44

## **Увод**

С напредване на технологиите се създават решения и системи за организиране на дадени предприятия било то университети или офиси и т.н. За целта се разработват системи които един вид улесняват хората на работното място да комуникират помежду си и да разбират за дейностите на дадената институция.

Такива системи са например: Система за отпуски в която всеки въвежда кога ще отсъства и защо или Университетска система за оценки и програма на занятията, която улеснява и студенти и учители.

Затова предметът на текущата дипломна работа е именно уеб приложение, което е насочено към управление на дейностите в даден университет и подобряване комуникацията между учител и студенти.

## **Глава I**

### **Обзор - състояние на проблема по литературни данни;**

#### **1. Анализ на темата. Цели и задачи.**

В днешно време компютрите са неделима част от нашето ежедневие .Това води до нуждата от разработване на уеб приложения от всякакво естество.

Съществуват уеб приложения от всякакви жанрове – бизнес, образование, финанси, спорт, социални, производителност, музика и аудио, пътешествия и местно съдържание и много други.

Уеб приложения тип "образование" са подходящи за доста хора защото една част от живот ние сме в някаква образователна система, било то училище или университет.

Предметът на текущата дипломна работа е именно уеб приложение , което е насочено към организирането главно на програмата , изпити и оценките в университет. Приложението е предназначено за всеки един студент или учител, желаещ да има дадена информация свързана с него и с университета.

#### **1.1 Анализ на темата.**

Основната цел е разработването на софтуерно решение за събирането на информацията касаеща студенти и учители . Решението се изразява в това да се създаде система която съчетава едновременно информация за студенти и информация за учители. По-добра комуникация между тях чрез лесно изпращане на имейли от учители към студенти с важна информация, която трябва да бъде достъпна по най-бързия начин. И чрез анкети за датите на предстоящите изпити или някаква друга полезна тема.

## 1.2 Поставена задача

Проектиране и реализиране на университетска информационна система под формата на уеб приложение. В днешно време повече хора имат лесен достъп до интернет затова уеб приложение е идеалния и лесен начин за достъп на потребителите до нужната информация. За решаването на дадената задача и постигане на нужните резултати ще използваме следните технологии за проектиране на приложението което да служи за лесен достъп до информация касаеща студенти и учители.

## 2.Използвани технологии

### 2.1 PyCharm

PyCharm е мощна интегрирана среда за разработка (на английски: integrated development environment, IDE) на софтуерни проблеми. Използва се за разработка на конзолни и графични потребителски интерфейс приложения. Предоставя по-лесен достъп до базата данни с която се работи, по-лесно използване на GIT предоставя лесен начин за merge на бранчове, ако съществуват конфликти. Вграден дебъгер и много допълнителни улесняващи работата модули.

### 2.2 GIT

**Git** е децентрализирана система за контрол на версиите на файлове. Създадена е от Линус Торвалдс за управление на разработката на Linux. Поради нуждата да се контролира огромната база от код на Linux ядрото, основна цел при разработката на Git е била бързината. Координатор на разработката на Git е Джунио Хамано.

Всяка локална Git директория е хранилище с пълна история и възможности за следене на версиите. Това прави Git независим от мрежови връзки към централен сървър.

Системи като GIT спомагат работата на много програмисти върху един и същ проект да комуникират един с друг и да се следи работата на всеки един от тях без да се объркват проекта и кода в него.

Това се постига чрез работа върху отделни branches тоест отделни части на даден код които в последствие се събират в една обща част.

## 2.3 Django REST

Django е базиран на Python web framework, който позволява MVC(Model View Controler) архитектурен модел който при Django се нарича MTV(Model Template View) , но и също така позволява работата с REST (Representational State Transfer) архитектурния модел, който е изпозван в тази дипломна работа. REST е стил софтуерна архитектура за реализация на уеб услуги. Основната идея е да се определи системен ресурс, който се променя в резултат на взаимодействието между доставчика на услуги и потребителя. Архитектурният модел REST включва взаимодействията между сървър и клиент, осъществени по време на трансфера на данни. Концепцията беше въведена за пръв път от Рой Филдинг през 2000 г. като част от неговата докторска дисертация. Филдинг е един от основните автори на HTTP протокола, под който се изпълняват REST имплементациите в повечето случаи.

Архитектурата REST е разработена успоредно с HTTP 1.1. Въпреки това, REST е обща архитектура, която може да бъде реализирана в други среди, а не само под HTTP. World Wide Web представлява най – голямото осъществяване на архитектурния стил на REST. REST – стилът обикновено се състои от клиенти и сървъри. Клиентите инициират заявки към сървърите; сървърите преработват заявките и връщат подходящи отговори. Заявките и отговорите са създадени през прехвърляне на образа на ресурси. Ресурсът може да бъде всякаква ясна и смислена концепция, която може да бъде адресирана. Представяне (анг. representation) на ресурс обикновено е документ, който намира сегашното възнамерявано състояние на ресурса.

Клиентът започва да изпраща заявки, когато е готов да направи преходът към ново състояние. Докато една или повече заявки са неизпълнени за клиента се смята, че е в преход. Представянето на всяко приложение се състои от линкове, които могат да бъдат използвани следващия път, когато клиентът избере да направи нови официални промени.

Архитектурният стил на „REST“ прилага шест ограничителни условия, като същевременно дава свобода за дизайна и имплементацията на индивидуалните компоненти:

### Клиент-сървър

Единният интерфейс разделя клиента и сървъра. Това означава, например, че клиента не се грижи за складирането на данни. Тази задача остава изцяло за сървъра, като по този начин се подобрява портативността на клиентския код (може да се използва в различни среди). Сървърът няма връзка с потребителския интерфейс и по този начин е по-семпъл и лесен за премащабиране. Клиентът и сървърът могат да бъдат заменени или развивани независимо един от друг, стига това да не налага промяна на единния интерфейс помежду им.

## Без статус на сесията (*Stateless*)

Следващото условие е на сървъра да не се запазват статуси на сесиите. Всяка заявка от клиента, съдържа в себе си нужната информация за нейната обработка, статуси на сесии се запазват единствено при клиента.

## Кеширане

Клиентът има право да кешира (запазва) информация, получена в отговор от сървъра, за да я преизползва при последващи заявки. За тази цел сървърът трябва имплицитно или експлицитно да е посочил дали информацията в отговора може да се кешира, за да се избегнат случаи, в които клиентът получава грешна информация при бъдещи заявки. При правилно управление и използване на кеширането могат частично или напълно да се елиминират ненужни взаимодействия между клиента и сървъра, като по този начин се подобрява бързината и производителността.

## Многослойна система

Обикновено клиентът не знае дали е свързан с крайния сървър или със сървър-посредник. Сървърите-посредници подобряват ефективността, като увеличават капацитета за обработване на заявки и предоставят споделени кешове. Също така те допринасят да подобряването на сигурността.

## Код при поискване (незадължително)

Сървърът може временно да разшири функционалността, изпращайки код, който се изпълнява директно при клиента. Например клиентски скриптове, написани на JavaScript или компилирани компоненти като Java applets.

## Единен интерфейс

Единният интерфейс между клиента и сървъра разделя и опростява архитектурата. По този начин всеки компонент може да се развива самостоятелно.

Единственото условие на REST архитектурата, което не е задължително е „Код по поискване“. Всяко приложение (услуга), изпълняващо на гореописаните условия, може да се нарече „RESTful“. Ако нарушава дори едно от условията, то не може да бъде считано за „RESTful“.

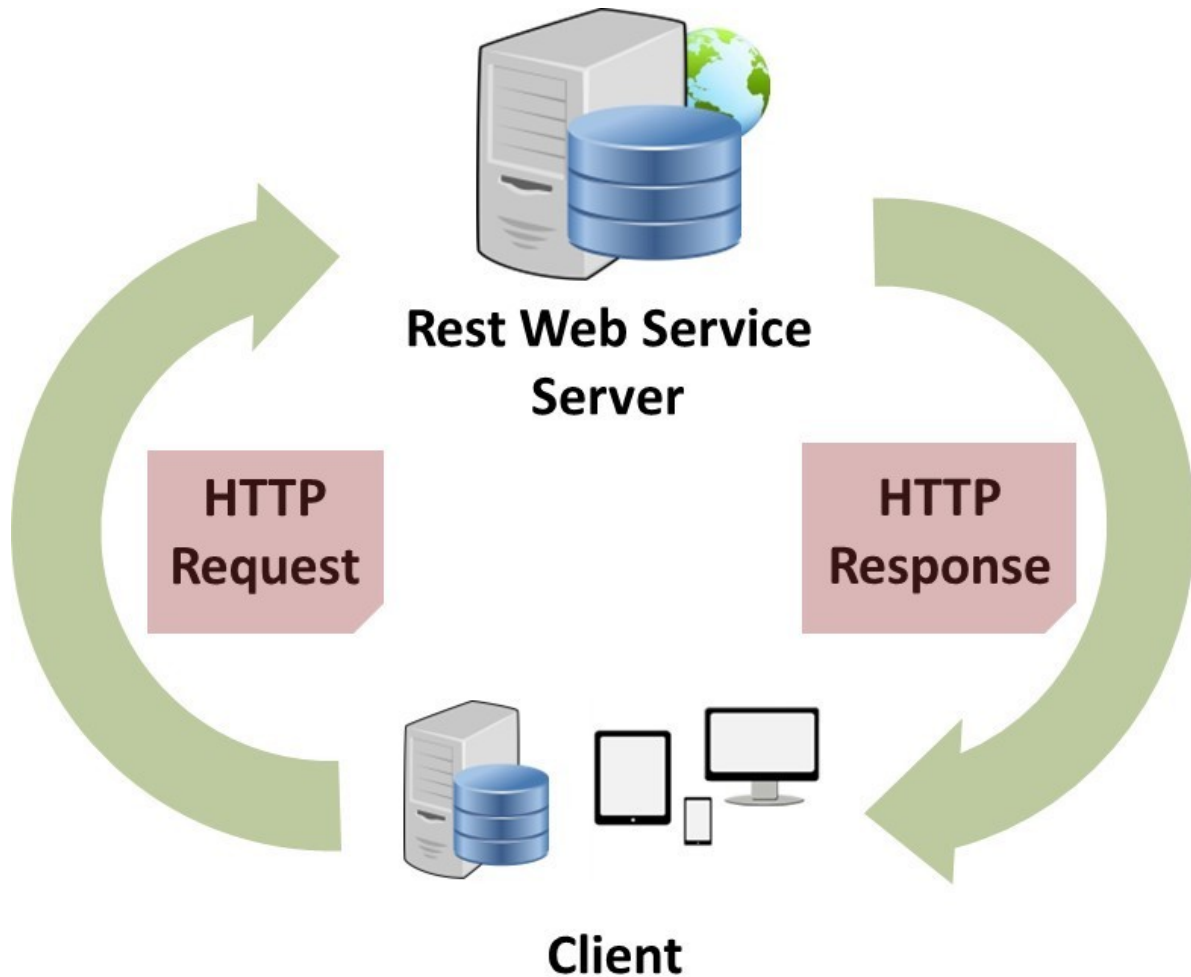
Всяка разпространена хипермедийна система, съответстваща на архитектурния стил на „REST“ притежава нужната производителност, мащабируемост, опростеност, гъвкавост, видимост, портативност и надеждност.

RESTful уеб API (също наричано RESTful уеб service) е уеб приложение, което използва принципите на HTTP и REST. Представлява колекция от ресурси със четири дефинирани аспекта:

- Основният „URL“ за уеб приложенията като <http://example.com/resources/>



- Internet media типът на данните поддържани от уеб приложенията. Това най-често е JSON, но може да бъде всеки друг валиден Интернет медиен тип, като се има предвид, че е валиден хипертекст стандарт.
- Операции поддържани от уеб приложениято използвайки HTTP методи (примерно: GET, PUT, POST, или DELETE).
- Приложенията трябва да се задвижват от хипертекст.



## 2.4 Python

**Python** е интерпретируем, интерактивен, обектно-ориентиран език за програмиране, създаден от Guido van Rossum в началото на 90-те години. Кръстен е на телевизионното шоу на BBC „*Monty Python's Flying Circus*“. Често бива сравняван с Tcl, Perl, Scheme, Java и Ruby.

„Python“ предлага добра структура и поддръжка за разработка на големи приложения. Той притежава вградени сложни типове данни като гъвкави масиви и речници, за които биха били необходими дни, за да се напишат ефикасно на C.

Python позволява разделянето на една програма на модули, които могат да се използват отново в други програми. Също така притежава голям набор от стандартни модули, които да се използват като основа на програмите. Съществуват и вградени модули, които обезпечават такива неща като файлов вход/изход (I/O), различни системни функции, сокети (*sockets*), програмни интерфейси към GUI-библиотеки като Tk, както и много други.

Тъй като Python е език, който се интерпретира, се спестява значително време за разработка, тъй като не са необходими компилиране и свързване (*linking*) за тестването на дадено приложение. Освен това, бидейки интерпретируем език с идеология сходна с тази на Java, приложение, написано на него, е сравнително лесно преносимо на множеството от останали платформи (или операционни системи).

Програмите, написани на Python, са доста компактни и четими, като често те са и по-кратки от еквивалентните им, написани на C/C++. Това е така, тъй като:

- наличните сложни типове данни позволяват изразяването на сложни действия с един-единствен оператор;
- групирането на изразите се извършва чрез отстъп, вместо чрез начални и крайни скоби или някакви други ключови думи (друг език, използващ такъв начин на подредба, е Haskell);
- не са необходими декларации на променливи или аргументи.
- Python съдържа прости конструкции, характерни за функционалния стил на програмиране, които му придават допълнителна гъвкавост

Всеки модул на Python се компилира преди изпълнение до код за съответната виртуална машина. Този код се записва за повторна употреба като **.pyc** файл.

Програмите написани на Python представляват съвкупност от файлове с изходен код. При първото си изпълнение този код се компилира до байткод, а при всяко следващо се използва кеширана версия. Байткодът се изпълнява от интерпретатор на Python.

- Строго типизиран (*strong typing*) – При несъответствие между типовете е необходимо изрично конвертиране.
- Динамично типизиран (*dynamic typing*) – Типовете на данните се определят по време на изпълнението. Работи на принципа *duck typing* – Оценява типа на обектите според техните свойства.
- Използва *garbage collector* – вътрешната реализация на езика се грижи за управлението на паметта.
- Блоковете се формират посредством отстъп. Като разграничител между програмните фрагменти използва нов ред.

Ето няколко причини да работим с него:

- Езикът притежава приятен и лесен за научаване синтаксис(може би най-лесния от всички други езици);
- Широко приложение. Позволява разработването на сървърна back-end логика, уеб , мобилни приложения с Kivu, десктоп приложения с PyQTи т.н.
- Особено подходящ за начинаещи;
- Наличие на огромен брой от технологични рамки, библиотеки и инструменти за разработка на езика;
  - Популярен и гъвкав;

## 2.5 Java Script

**JavaScript** (чете се *джаваскрипт*) е интерпретируем език за програмиране, разпространяван с повечето Уеб браузъри. Поддържа обектно-ориентиран и функционален стил на програмиране. Създаден е в Netscape през 1995 г. Най-често се прилага към HTML-а на Интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. Javascript е стандартизиран под името EcmaScript.

JavaScript е програмен език, който позволява динамична промяна на поведението на браузъра в рамките на дадена HTML страницата. JavaScript се зарежда, интерпретира и изпълнява от уеб браузъра, който му осигурява достъп до Обектния модел на браузъра. JavaScript функции могат да се свържат със събития на страницата (например: движение/натискане на мишката, клавиатурата или елемент от страницата, и други потребителски действия). JavaScript е най-широко разпространеният език за програмиране в Интернет. Прието е JavaScript програмите да се наричат скриптове.

### Разлики с Java

Освен съвпадението в част от името, двата езика нямат кой знае какви прилики, дори са разработени от различни корпорации (Java е дело на Sun, а JavaScript е разработка на Netscape). Java е популярен език за програмиране не само на Интернет приложения, но и на самостоятелни програми за различни платформи. Интернет приложенията на Java се наричат аплети. Те са файлове с разширение .class и се вмъкват в HTML документа между таговете <applet> и </applet>.

## Възможности

JavaScript може да влияе на почти всяка част от брауъра. Брауъра изпълнява JavaScript кода в цикъла на събития т.е. като резултат от действия на потребителя или събития в брауъра (например document.onLoad).

Основни задачи в повечето JavaScript приложения са:

- Зареждане на данни чрез AJAX.
- Ефекти с изображения и HTML елементи: скриване/показване, пренареждане, влачене, слайд шоу, анимация и много други.
- Управление на прозорци и рамки.
- Разпознаване на възможностите на брауъра.
- Използване на камерата и микрофона.
- Създаване на 3D графики WebGL.
- По-добър и гъвкав потребителски интерфейс

Какво не може да се прави с помощта на JavaScript:

- Не може да се записва информация на потребителския компютър или отдалечения сървър.
- Не може да се запазва информация директно в отдалечена база данни.
- Не може да се стартират локални приложения.

## 2.5 React JS

React е библиотека на Java Script за създаване на потребителски интерфейс. Поддържа се от Facebook.

Реакт може да се използва както за създаване единична страница(**single-page application (SPA)**) е уеб приложение или сайт, при който динамично се презаписва текущата страница вместо да се презарежда напълно нова страница. при СПА или всички HTML, JS, CSS се зарежда при първоначалното зареждане или необходимия ресурс се добавя динамично. Страницата не се презарежда в никой момент) или мобилно приложение. Реакт само има грицата да предаде данните на DOM (**Document Object Model** е интерфейс, който основава XML или HTML документи като дървовидни структури, където всеки възел е обект представляващ част от документа) за това за създаване на приложения с Реакт са необходими допълнителни библиотеки за управления на състояние и на пътищата като: Redux, React Router, Axios.

Реакт използва Virtual DOM, създава се кеш който изчислява разликите между първоначалното състояние и следващото състояние и актуализира ДОМ дървото ефективно. Така все едно се актуализира цялата страница а всъщност се обновява подкомпонентите които са били променени.

Реакт също има hooks(техника която променя поведението на дадено приложение като пресича извикване на функции,събития или съобщения преминали през софтуерни компоненти. Код който управлява тези прекъсвания при извикване на функции, събития или съобщения се наричат Hooks), които позволяват изпълняването на код в даден момент от зареждането на даден компонент.

- `ShouldComponentUpdate` - спира презареждането на даден компонент ако не е необходимо като самата функция върне `False`
- `componentDidMount` – се извиква когато компонента е зареден и създаден. Използва се обикновено, когато трябва да се вземат данни от API
- `render` – е най-важният метод и е задължителен във всеки компонент. Извиква се всеки път когато състоянието на компонента се промени, което трябва да бъде отразено на потребителския интерфейс.

Реакт използва **JSX**(JavaScript XML), който е надграждане на JS синтаксиса

## 2.5 JSON

JSON, или JavaScript Object Notation, е текстово базиран отворен стандарт създаден за човешки четим обмен на данни. Произлиза от скриптовия език JavaScript, за да представя прости структури от данни и асоциативни масиви, наречени обекти. Въпреки своята връзка с JavaScript, това е езиково независима спецификация, с анализатори, които могат да преобразуват много други езици в JSON.

Форматът на JSON първоначално е бил създаден от Дъглас Крокфорд (Douglas Crockford) и е описан в RFC 4627. Официалният Интернет медия тип за JSON е `application/json`. Разширението на файловете написани на JSON е `.json`.

Форматът на JSON често е използван за сериализация и предаване на структурирани данни през Интернет връзка. Използва се главно, за да предаде данни между сървър и Интернет приложение, изпълнявайки функциите на алтернатива на XML.

Базовите типове данни на JSON са:

- `Number` (число с плаваща запетая, `double precision floating-point format` в JavaScript)
- `String` (низ от символи с Unicode кодиране, затворени в двойни кавички, като „специалните“ символи се представят с т.нар. `escaping` – символни последователности, започващи със символа `„\“`)
- `Boolean` (`true` или `false`)
- `Array` (наредена поредица от стойности, разделени със запетая и затворени в квадратни скоби; стойностите не е задължително да бъдат от един и същ тип)

- Object (неподредена колекция от двойки ключ:стойност, символът „:“ разделя ключът и стойността, разделени със запетая и затворени в къдрави скоби; ключовете трябва да са string-ове и да са различни един от друг)
- null (empty)

Всяко незначимо бяло пространство може да бъде добавено около „структурните символи“ (като скоби „{} []“, двоеточие ':' и запетая ',').

Следващият пример показва представянето на обект, който описва човек в JSON. Обектът има string полета за първо и последно име, Number поле за години, Object, който представя адресът на човека и Array от телефонни номера представени като Object.

## 2.6 AXIOS

Axios е JS библиотека, която прави HTTP заявки от браузера. Главната и цел е да подобри .fetch() метода с който се правят HTTP заявки в чистия JS. При .fetch() трябва допълнително да използваш метода .json() за да вземеш реалната дата. (fetch(`\${baseUrl}/drivers`).then(res => res.json())), но при Axios не е така (axios.get(`\${baseUrl}/drivers`).then(res => res)).

Някои от основните свойства на axios са:

- XMLHttpRequests от браузера
- HTTP заявки от node.js
- Използва Promises (Promise обектът представлява евентуалният завършек (или неуспех) на една асинхронна операция и нейната получена стойност.)
- Трансформира request и response данните

## 2.6 REDUX

Е библиотека на JS за управление на състоянието на дадено приложение

Управление на състоянието(данни), състоянието са данни който се променят. Състоянието определя какво да се покаже на потребителския интерфейс. Като цяла има три аспекта на данните , които трябва да контролираме:

- Взимане и запазване на данни
- Придаване на данни към ПИ

- Променяне на данни

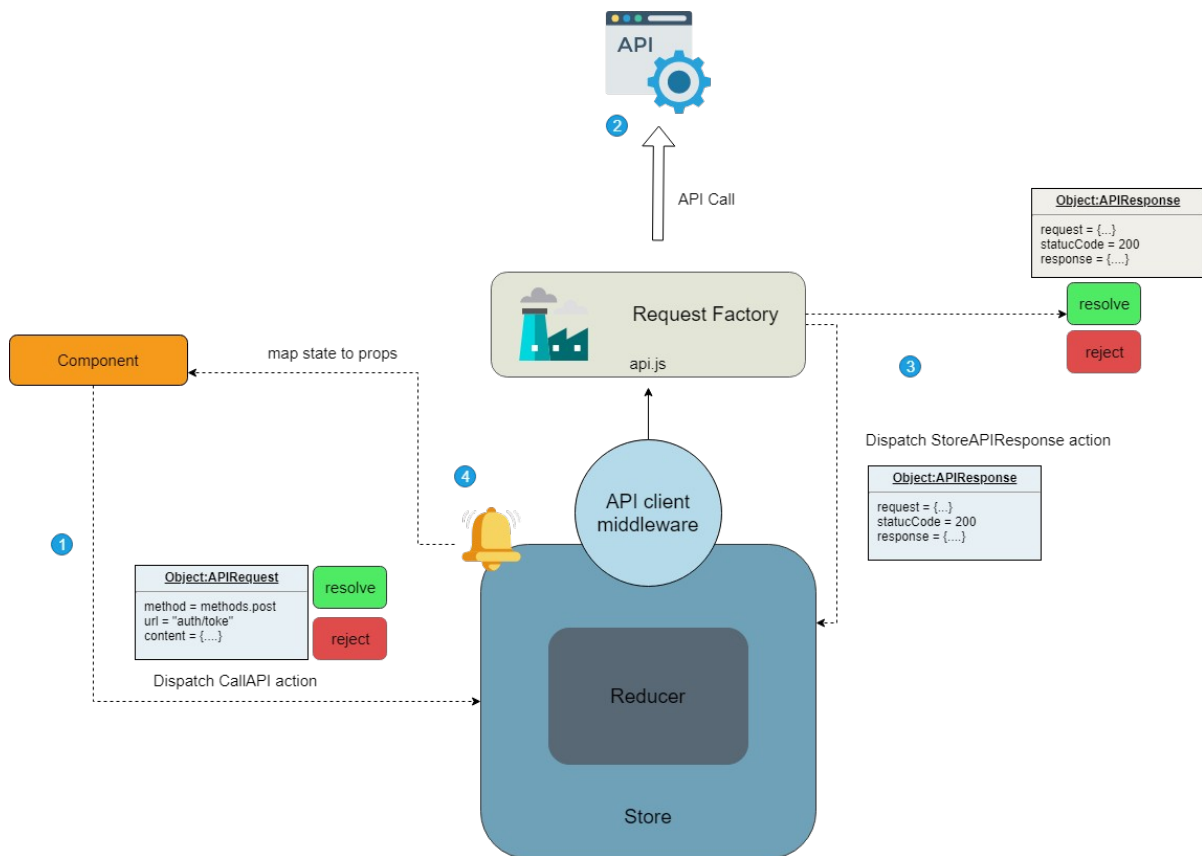
Тъй като в Реакт разделяме ПИ на отделни компоненти и всеки компонент може да бъде разделен на други. След като вземем данните от сървъра ги зареждаме в Redux и ги пазим за когато са необходими на съответния елемент. От там ги взимаме в съответния компонент и ги показваме в интерфейса. Така е много лесно да се направи връзката между отделните компоненти. Съответно и ако не се променят данните зареждането на отделните елементи става доста по бързо отколкото да се обръщаш към сървъра всеки път.

## Глава 2 – Теоретично решение на поставената задача;

За решаването да дадената задача използваме REST архитектура и СУБД MySQL в която се държат всички необходими данни и чрез извикване на endpoints се достъпват необходимите данни и предават на фронтенда.

### 1.Цялостна архитектура

Цялостната архитектура на приложението е изградена от RESTful приложение (сървър) и база данни. Фронтенда комуникира с RESTful приложението чрез HTTP заявки, като междинен слой и място за съхранение на данните се явява Redux. Връзката между RESTful приложението и базата осигурява извличането и записването на данни, съхранявани в базата данни.



Най-използваните HTTP заявките са GET,POST,PUT,DELETE.

## HTTP GET

GET заявки са за да изтегляне и само представяне на информация / информация - и да не я променят по никакъв начин. Тъй като GET заявките не променят състоянието на ресурса, те са така наречените безопасни методи . Освен това, GET API трябва да бъде idempotent , което означава, че отправянето на множество еднакви заявки трябва да произвежда един и същ резултат всеки път. При архитектурата с Redux след дадена GET заявка данните се зареждат в така наречения store на Redux чрез така наречените Reducers който взимат payload-а от GET заявката и зареждат store-а. След всички тези операции фронтенда чете данните директно от store-а в който вече са заредени дадените данни.

## HTTP POST

POST е за създаване на нови подчинени ресурси , например файл е подчинен на директория, която го съдържа или ред е подчинен на таблица на база данни. По отношение на REST, POST методите се използват за създаване на нов запис в базата главно.

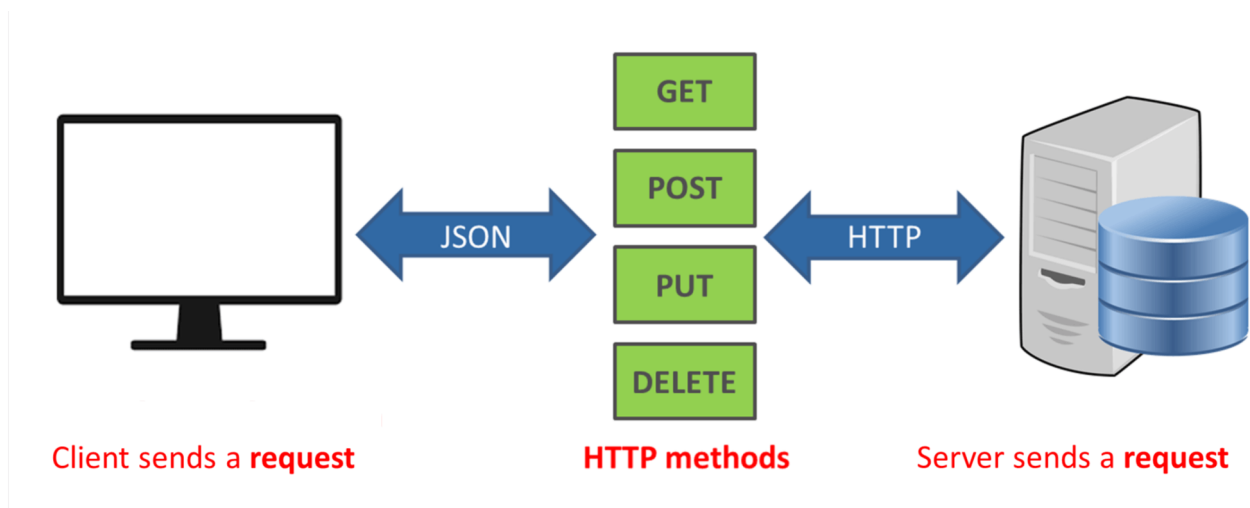


## HTTP PUT

Методът PUT главно се използва за променяне на вече създаден запис в базата. Извикването на една и съща заявка PUT многократно винаги ще доведе до един и същ резултат. С

## DELETE

Методът DELETE изтрива запис в базата. Добра практика е да не си трият никога данните в базата, ако е възможно. Прави се изкуствено изтриване променя се дадено поле в базата обикновено е булево поле което се променя от 1 на 0 и така знаем, че дадената стойност не трябва да се използва. В повечето случаи се постига чрез overwrite на дадения метод който отговаря за изтриването.



### 3. Данни предоставени от информационната система.

Данни, включени в API-то:

Студенти: Списък с всички студенти с приложими филтри към тях.

Пример:

```
{
  "0":{
    "id":1,
    "student_profile":{
      "id":2,
```

```
"student_num":19154562,
"qualification_type":"BD",
"group":{
  "id":1,
  "name":"42a",
  "major":{
    "id":1,
    "name":"KST"
  }
},
"faculty":"FEA",
"semester":1
},
"grades":[
  {
    "id":1,
    "discipline":{
      "id":1,
      "name":"SAA",
      "faculty":"FEA",
      "semester":2
    },
    "grade":2,
    "student":2,
    "created":"2019-11-06"
  },
  {
    "id":2,
    "discipline":{
      "id":4,
```

```

        "name":"DS",
        "faculty":"FEA",
        "semester":1
    },
    "grade":4,
    "student":2,
    "created":"2019-11-06"
},
{
    "id":3,
    "discipline":{
        "id":5,
        "name":"ASLS",
        "faculty":"FEA",
        "semester":1
    },
    "grade":3,
    "student":2,
    "created":"2019-11-06"
}
],
"is_superuser":true,
"username":"student",
"first_name":"Gosho1",
"last_name":"Vankov",
"surname":"Iliev",
"email":"b.tumbev@code-nest.com",
"is_active":true
}
}

```

Учители: Списък с всички учители с приложими филтри към тях.

Пример :

```
{
  "0":{
    "id":4,
    "teacher_profile":{
      "id":1,
      "faculty":"FEA",
      "groups":[
        {
          "id":1,
          "name":"42a",
          "major":1
        },
        {
          "id":3,
          "name":"41b",
          "major":1
        },
        {
          "id":4,
          "name":"42b",
          "major":1
        }
      ]
    },
    "is_superuser":false,
    "username":"teacher",
    "first_name":"teacher",
    "last_name":"teacher",
```

```

    "surname":"teacher",
    "email":"teacher@teacher.com",
    "is_active":true
  }
}

```

Оценки: Оценките по предмети и оценките на даден студент с дата кога е била въведена и за кой предмет.

Пример:

```

{
  "grades":[
    {
      "id":1,
      "discipline":{
        "id":1,
        "name":"SAA",
        "faculty":"FEA",
        "semester":2
      },
      "grade":2,
      "student":2,
      "created":"2019-11-06"
    },
    {
      "id":2,
      "discipline":{
        "id":4,
        "name":"DS",
        "faculty":"FEA",
        "semester":1
      },

```

```

    "grade":4,
    "student":2,
    "created":"2019-11-06"
  },
  {
    "id":3,
    "discipline":{
      "id":5,
      "name":"ASLS",
      "faculty":"FEA",
      "semester":1
    },
    "grade":3,
    "student":2,
    "created":"2019-11-06"
  }
]
}

```

Студентските групи: Списък на групите на студентите

```

{
  "0":{
    "id":2,
    "name":"41a",
    "major":{
      "id":1,
      "name":"KST"
    }
  }
}
}

```

График със всички занятия: Предоставени под формата на календар.

Пример:

```
{
  "0":{
    "id":1,
    "resourceId":1,
    "title":"SAA 1222 teacher",
    "type_of":"P",
    "start":"2019-02-11 05:30:00",
    "end":"2019-02-11 07:15:00",
    "bgColor":"#c02628",
    "resizable":false,
    "movable":false,

    "rrule":"FREQ=WEEKLY;DTSTART=20190211T053000Z;UNTIL=20200411T061500Z;BYDAY=MO",
    "teacher":"teacher",
    "room":"1222"
  }
}
```

Данните от проведени анкети: Предоставени под формата на графика(пай)

Пример:

```
{
  "vypros 11":[
    {
      "answer__title":"otg na vypros 1 - 2",
      "question__title":"vypros 11",
      "survey__title":"Servey",
      "answ_count":1
    }
  ]
}
```

```
    }  
  ]  
}
```

Списък със специалности:

Пример:

```
[  
  {  
    "id":1,  
    "name":"KST"  
  },  
  {  
    "id":2,  
    "name":"E"  
  },  
  {  
    "id":3,  
    "name":"ET"  
  },  
  {  
    "id":4,  
    "name":"ASD"  
  }  
]
```

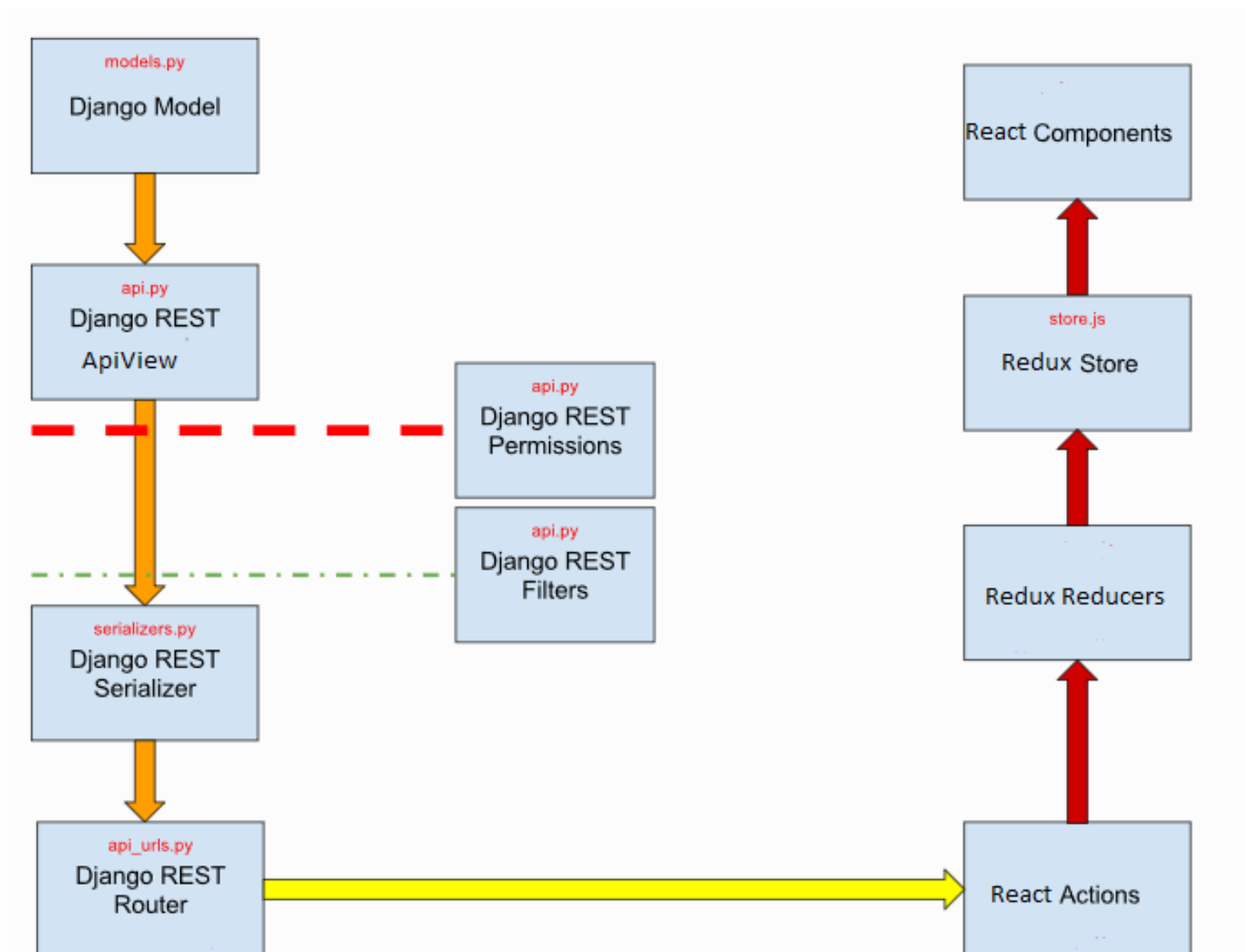


## Глава 3 – Описание на софтуерната част;

Системата е изградена от BE(API) REST архитектура, FE изграден от отделни компоненти.

BE е изграден от ApiView-та, Модели на данните и Serializers.

Всяка FE страница има съответно изглед (HTML часта във всеки компонент), логически код (js код, който се грижи за всички events на всеки компонент) .Навигацията между отделните компоненти се реализирана с помощта на рутер, всеки FE endpoint е асоцииран към отделен компонент.



1. Django Models – описание на всяка таблица в базата под формата на клас с пропертията тоест всяко поле с неговия тип. По този модел Django си създава миграционен файл с който и да се създаде съответната база данни.

```
class UniUser(AbstractUser):
    roles = models.ManyToManyField(Role)
    surname = models.CharField(max_length=30)
    email = models.EmailField(unique=True)
```

2. ApiView – помага да опишем логиката, която прави всеки endpoint свързан към него. В това view дефинираме основните типове HTTP методи – GET, POST, PUT, DELETE

```
class ResetPasswordConfirm(APIView):

    def get(self, request, token):

        token_obj = AuthTokenPassReset.objects.filter(token=token)
        if not token_obj:
            return JsonResponse({'error': 'token is invalid'},
status=400)
        else:
            return HttpResponse(status=200)

    def post(self, request, token):

        token_obj = AuthTokenPassReset.objects.filter(token=token)
        if token_obj:
            user_obj =
UniUser.objects.get(email=token_obj[0].email)
            try:
                password = request.data['password']
                user_obj.set_password(password)
                user_obj.save()
                token_obj[0].delete()
                return JsonResponse({'msg': 'password is changed
successfully'})
            except:
                return JsonResponse({'error': 'Please provide
password'}, status=400)
        else:
            return JsonResponse({'error': 'token is invalid'},
status=400)
```

3. Permissions – е логика която проверява ауторизираният потребител какво роля има и дали да го допусне до съответния Endpoint.

```
class GroupList(generics.ListCreateAPIView):  
    permission_classes = [permissions.IsAuthenticated,]
```

4. Serializers – описва как данните ще бъдат предоставени от Endpoint-а. Всеки serializer има методи Create, Update в който се пише логика например, когато със създаването на един обект трябва да се създаде друг  
преждевременно, тоест се пише custom логика за създаването или  
променянето на даден обект. serSerializerST(serializers.ModelSerializer):

```
student_profile = StudentProfileSerializer()
```

```
class Meta:  
    model = UniUser  
    fields = ('id', 'student_profile', 'password', 'is_superuser', 'username',  
'first_name', 'last_name', 'surname',  
'email', 'is_active')
```

```
def __init__(self, *args, **kwargs):  
    super().__init__(*args, **kwargs)  
    if self.context['request'].method == "PUT":  
        self.fields.pop('password')
```

```
def create(self, validated_data):  
    student_data = validated_data.pop('student_profile')  
    password = validated_data.pop('password')  
    user_obj = UniUser(**validated_data)  
    user_obj.set_password(password)  
    user_obj.save()
```

```
role_obj, _ = Role.objects.get_or_create(id=Role.STUDENT)  
user_obj.roles.add(role_obj.id)  
st_obj = StudentProfile(user=user_obj,  
student_num=gen_student_num(student_data['faculty']), **student_data)  
st_obj.save()
```

```
return user_obj
```

```
def update(self, instance, validated_data):  
    user_obj = instance  
    st_obj = instance.student_profile
```

```
student_data = validated_data.pop('student_profile')  
if 'password' in validated_data:  
    password = validated_data.pop('password')  
    user_obj.set_password(password)
```

```

        user_obj.save()

        user_data = validated_data

        StudentProfile.objects.filter(id=st_obj.id).update(**student_data)
        UniUser.objects.filter(id=user_obj.id).update(**user_data)
        return user_obj

```

## 5. Reducers

Приемат данните от даден action и ги записват в междинния store на Redux, от който FE чете данните.

```

export const updateObject = (oldObject, updatedProperties) => {
  return {
    ...oldObject,
    ...updatedProperties
  }
}

const getStudents = (state, action) => {
  return updateObject(state, {
    students: action.payload,
  });
}


```

Помощната функция updateObject взима стария обект и добавя новите данни към предшното състояние на обекта. Action.payload са данните от самия action.


## Глава 4 – Функционално тестване;


Система се състои от начална страница с основния календар и резултати от проведена анкета. Страница за създаване на анкети и списък с вече съществуващите. Контролен панел, в който има списък с всички упражнения и лекции, които се показват в календара, Групи със студенти, всички дисциплини и специалности. Може да се добавят нови и да се редактират вече съществуващи. Страница в която има списък с всички учители и студенти могат да се добавят нови да се редактират вече съществуващи. Страница с личния профил на всеки студент.

1. Начална страница: Основен календар със всички упражнения за деня (като може да се разбие на ден, седмица, месец и година и да променяш за коя специалност да ти покаже упражненията) и резултати от дадена анкета.



- Surveys
- Control Panel
- Users


**STUDENT INFORMATION SYSTEM**

Profile 

KST

< Feb 2-8, 2020 >

Day Week Month Quarter Year

Resource Name	Sun 2/2	Mon 2/3	Tue 2/4	Wed 2/5	Thu 2/6	Fri 2/7	Sat 2/8
41a		<div>OS 1222 teach...</div> <div>SS 1222 teacher</div> <div>DS 1222 teach...</div> <div>ASLS 1222 tea...</div>	<div>TB 1222 teacher</div> <div>Fizichsko 122...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div>			
41b		<div>ASLS 1222 tea...</div> <div>SAA 1222 teac...</div> <div>DS 1222 teach...</div> <div>SS 1222 teacher</div> <div>OS 1222 teach...</div>	<div>TB 1222 teacher</div> <div>Fizichsko 122...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div>			
42a		<div>SAA 1222 teac...</div> <div>OS 1222 teach...</div> <div>ASLS 1222 tea...</div> <div>SAA 1222 teac...</div> <div>SS 1222 teacher</div> <div>DS 1222 teach...</div>	<div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div>			
42b		<div>OS 1222 teach...</div> <div>SS 1222 teacher</div> <div>DS 1222 teach...</div>	<div>ASLS 1222 tea...</div> <div>SAA 1222 teac...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div> <div>Fizichsko 122...</div>			
43a		<div>DS 1222 teach...</div> <div>SS 1222 teacher</div> <div>DS 1222 teach...</div>	<div>OS 1222 teach...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div> <div>Fizichsko 122...</div>			
43b		<div>DS 1222 teach...</div> <div>DS 1222 teach...</div> <div>SS 1222 teacher</div> <div>SAA 1222 teac...</div>	<div>OS 1222 teach...</div> <div>ASLS 1222 tea...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div> <div>Fizichsko 122...</div>			

Survey

vypros 11

vypros 2

100%

100%

KST

< Feb 2-8, 2020 >

Day Week

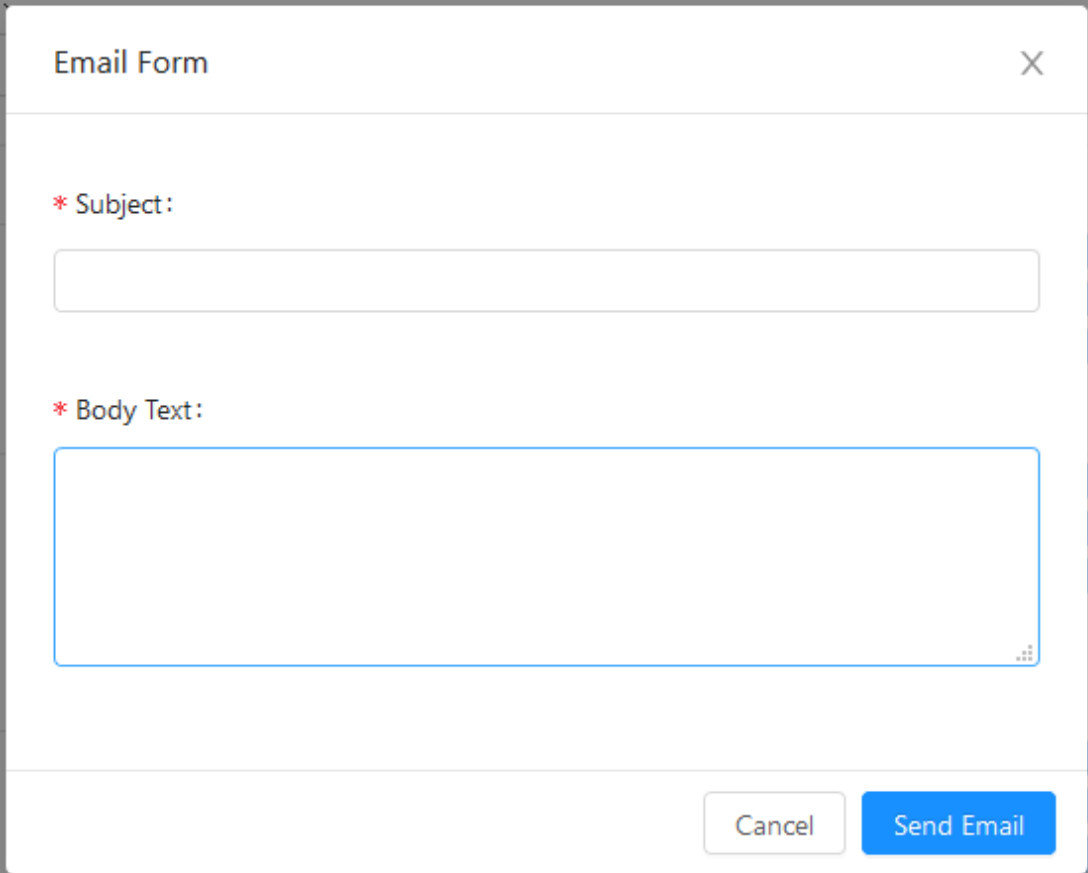
KST

E

ET

ASD

	Sun 2/2	Mon 2/3	Tue 2/4	Wed 2/5	Thu 2/6
		<div>OS 1222 teach...</div> <div>SS 1222 teacher</div>	<div>TB 1222 teacher</div> <div>Fizichsko 122...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>2 teach...</div>	
41b		<div>OS 1222 teacher</div> <div>11:30 Feb 3 - 13:15 Feb 3</div> <div>send email to students Ops 2</div> <div>SAA 1222 teac...</div> <div>DS 1222 teach...</div> <div>SS 1222 teacher</div> <div>OS 1222 teach...</div>	<div>Fizichsko 122...</div> <div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>OS 1222 teach...</div> <div>SAA 1222 teac...</div>	
42-		<div>SAA 1222 teac...</div> <div>OS 1222 teach...</div> <div>ASLS 1222 tea...</div>	<div>SS 1222 teacher</div> <div>ASLS 1222 tea...</div>	<div>DS 1222 teach...</div> <div>OS 1222 teach...</div> <div>SAA 1222 teac...</div>	



The image shows a screenshot of a software interface with a modal dialog box titled "Email Form". The dialog has a close button (X) in the top right corner. It contains two required fields, both marked with a red asterisk (\*):

- \* Subject:** A single-line text input field.
- \* Body Text:** A multi-line text area.

At the bottom right of the dialog are two buttons: "Cancel" and "Send Email". The "Send Email" button is highlighted in blue. In the background, a dark red button labeled "SAA 1222 teac..." is partially visible.

Както и възможност на учител да изпрати Email на всички студенти, които ще присъстват на дадено упражнение.

2. Контролен Панел: Отделни табове за CRUD на дисциплинте в Календара, групите, отделните дисциплини и специалност.

Group	Discipline	Type	Start	End	Whole practice end	Edit
42a	SAA	P	2019-02-11 05:30	2019-02-11 07:15	2020-04-11 06:15	Edit
41a	OS	P	2019-02-11 11:30	2019-02-11 13:15	2020-04-11 13:15	Edit
41a	SS	P	2019-02-11 13:45	2019-02-11 15:30	2020-04-11 15:30	Edit

### 3. Страница за CRUD на анкетите

#### SURVEYS

#	Title	Major	Show on home page	Activate	Send Emails	Edit
3	Servey	KST	Show	Activate	Send	Edit
4	фотьойл бира	KST	Show	Activate	Send	Edit

\* Title:

\* Major:

question 1

\* Title:

\* answer - 1:

+ Add Answer

question 2

\* Title:

\* answer - 1:

+ Add Answer

+ Add question

Submit



#### 4. Страница с CRUD на учители и студенти и добавяне на оценки.

Add Student

First Name	Last Name	Email	Student Num	Add Grade	Edit
Gosho1	Vankov	b.tumbev@code-nest.com	19154562	Add Grade	Edit
mitaka	mishakov	student2@student.com	19158649	Add Grade	Edit
zorev	sor	student3@student.com	19224169	Add Grade	Edit

< 1 >

STUDENT INFORMATION SYSTEM

Grade

\* Discipline:

OS

✓

\* Grade:

2

Cancel

Add grade

## 5. Профил на логнатия потребител.

### Student Info

First Name	Surname	Last Name	Email
Gosho1	Iliev	Vankov	b.tumbev@code-nest.com
Faculty	Group	Major	Qualification Type
FEA	42a	KST	BD
Semester	Student Number	Status	
1	19154562	● Active	
Grades			
· SAA : 2 - 2019-11-06 · DS : 4 - 2019-11-06 · ASLS : 3 - 2019-11-06			

## Глава 5 – Приложимост на дипломната работа;

Системата е разработена да обслужва информативно преподаватели и студенти с цел по-лесен достъп до необходимата информация. По-лесно и достъпно записване и управление на оценки.

Също така добра обратна връзка на преподаватели със студенти по даден въпрос чрез анкети(въпросник)

Интерфейса е изчистен и максимално опростен за да може да се работи лесно и бързо с него.

## Глава 6 – Икономическа оценка на резултатите и техническа ефективност

### 1.Заклучение

Системата реализира най-важната и основна функционалност необходима за даден университет, което е информация за предстоящи изпити, календар с всички упражнения/лекции и отбелязани оценки от изпити.

Част от бъдещо развитие на системата включва:

- Подобряване на потребителския интерфейс
- Повече информация.
- Създаване на реални изпити в самата система.

### 2.Извод

В основа на разработената от мен система стои Django Framework, програмния език Python и React Framework за Frontend частта.

Избрах да използвам тези технологии, защото дават възможност за по-обширна функционалност. Развиват се бързо и имат голяма общност и лесно може да намериш решения за дадени проблеми. React също така е от модерните FE Frameworks с които се създават one-page приложения, които са бъдещето. Също така улеснява работа при създаването на самия FE.

Избрах да използвам REST архитектура за да бъде по-добре разделен проекта на FE и BE част, защото улеснява процеса и начина на работа и дава по-голяма възможност за развитие на дадена система.

## ИСТОЧНИЦИ:

<https://www.wikipedia.org/>

<https://reactjs.org/>

<https://www.djangoproject.com/>

<https://www.django-rest-framework.org/>

<https://redux.js.org/>

<https://github.com/axios/axios>

## Приложения:

```
export const getStudents = () => {  
  
  return dispatch => {  
    axios.get('/api/students')  
      .then(res => {  
        dispatch({  
          type: GET_STUDENTS,  
          payload: res.data  
        });  
      })  
    .catch(err => {  
      console.log('get student error -> ' + err)  
    })  
  }  
}
```

```

//      dispatch(authFail(err))
    })
  }
}

const getStudents = (state, action) => {
  return updateObject(state, {
    students: action.payload,
  });
}

const reducer = (state=initialState, action) => {
  switch (action.type) {
    case ADD_STUDENT: return addStudent(state, action);
    case GET_STUDENTS: return getStudents(state, action);
    case EDIT_STUDENT: return editStudent(state, action);
    case CHANGE_IS_EDIT_FALSE: return changeIsEditFalse(state, action);
    default:
      return state;
  }
}

questionsRender(){
  const { getFieldDecorator, getFieldValue } = this.props.form;
  let that = this;
  var answers = []

  return Object.keys(this.state.answer_arr).map(function(e, i){
    answers = []
    that.state.answer_arr[e].map(function(a, a_i){
      answers.push(
        <div className="answer" key={a_i}>
          <Form.Item className="answer_form" label={`answer - ${a_i + 1}`}>
            {getFieldDecorator(`answer-${a}-${e}`, {
              rules: [{ required: true,
                message: 'Please input answer!'
              }],
            })(
              <Input/>,
            )
          </Form.Item>
          <Icon
            className="dynamic-delete-button"
            type="delete"
            onClick={() => that.removeA(a, i)}
          />
        </div>
      )
    }
  )
}

```

```

    })
    return(
        <div key={i}>
            <Divider className='question_divider'>question {i+1}</Divider>
            <div className="question">
                <Form.Item className="question-form" label={'Title'}>
                    {getFieldDecorator(`question-${e}`, {
                        rules: [{ required: true,
                            message: 'Please input question!'
                        }],
                    })(
                        <Input/>,
                    )}
                </Form.Item>
                <Icon
                    className="dynamic-delete-button"
                    type="delete"
                    onClick={() => that.removeQ(e)}
                />
            </div>
            {answers}
            <div className="VallAs" onClick={() => that.addA(i)}>
                <Icon type="plus" /> Add Answer
            </div>
        </div>
    )
})

// return questions
}

class UniUserSerializerST(serializers.ModelSerializer):
    student_profile = StudentProfileSerializer()

    class Meta:
        model = UniUser
        fields = ('id', 'student_profile', 'password', 'is_superuser', 'username', 'first_name', 'last_name', 'surname',
            'email', 'is_active')

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if self.context['request'].method == "PUT":
            self.fields.pop('password')

    def create(self, validated_data):
        student_data = validated_data.pop('student_profile')
        password = validated_data.pop('password')
        user_obj = UniUser(**validated_data)
        user_obj.set_password(password)
        user_obj.save()

```

```

    role_obj, _ = Role.objects.get_or_create(id=Role.STUDENT)
    user_obj.roles.add(role_obj.id)
    st_obj = StudentProfile(user=user_obj, student_num=gen_student_num(student_data['faculty']),
**student_data)
    st_obj.save()

    return user_obj

def update(self, instance, validated_data):

    user_obj = instance
    st_obj = instance.student_profile

    student_data = validated_data.pop('student_profile')
    if 'password' in validated_data:
        password = validated_data.pop('password')
        user_obj.set_password(password)
        user_obj.save()

    user_data = validated_data

    StudentProfile.objects.filter(id=st_obj.id).update(**student_data)

    UniUser.objects.filter(id=user_obj.id).update(**user_data)

    return user_obj

class StudentDetail(generics.RetrieveUpdateDestroyAPIView):

    def get_serializer_class(self):
        if self.request.method == 'GET':
            return UniUserSerializerStGET
        else:
            return UniUserSerializerST

    def get_queryset(self):
        user_id = self.kwargs.get('pk', "")

        if user_id:
            resp = UniUser.objects.filter(id=user_id, roles=Role.STUDENT)
        else:
            resp = UniUser.objects.filter(roles=Role.STUDENT)

        return resp

    def delete(self, request, *args, **kwargs):
        # self.destroy(request, *args, **kwargs)
        instance = self.get_object()
        instance.is_active = False

```

```
instance.save()  
return Response({  
    "Msg": "Студента е деактивирана успешно."  
})
```