Съдържание

[1. Увод 4](#_Toc196470288)

[2. Цел и задачи на проекта 5](#_Toc196470289)

[2.1 Цел на проекта 5](#_Toc196470290)

[2.2 Задачи на проекта 5](#_Toc196470291)

[2.2.1. Проучване на REST API: 6](#_Toc196470292)

[2.2.2. Анализ на съществуващи училищни информационни системи: 6](#_Toc196470293)

[2.2.3. Избор на технологии: 6](#_Toc196470294)

[2.2.4. Разработка на приложение: 6](#_Toc196470295)

[2.2.5. Тестиране и валидиране на приложението: 6](#_Toc196470296)

[2.2.6. Демонстрация на проекта: 7](#_Toc196470297)

[3. Теоретични основи 7](#_Toc196470298)

[3.1 Въведение в REST API 7](#_Toc196470299)

[3.2 Основни принципи на REST 7](#_Toc196470300)

[3.3 Формати за обмен на данни 8](#_Toc196470301)

[3.3.1. JSON: 8](#_Toc196470302)

[3.3.2. XML: 8](#_Toc196470303)

[3.4 Как JSON и AJAX спомагат за разработката на RESTful приложения 8](#_Toc196470304)

[3.4.1. Какво е AJAX? 9](#_Toc196470305)

[3.4.2. Как работи AJAX? 9](#_Toc196470306)

[3.4.3. Предимства на AJAX 9](#_Toc196470307)

[3.4.4. Взаимодействие между JSON и AJAX в RESTful приложения 9](#_Toc196470308)

[3.5 Работа с HTTP методи 10](#_Toc196470309)

[3.6 Свързани технологии и библиотеки 11](#_Toc196470310)

[3.6.1. Axios: 11](#_Toc196470311)

[3.6.2. Fetch API: 11](#_Toc196470312)

[3.6.3. jQuery: 11](#_Toc196470313)

[3.7 Axios: Библиотека за HTTP заявки в JavaScript 11](#_Toc196470314)

[3.7.1. Какво е Axios? 12](#_Toc196470315)

[3.7.2. Основни характеристики на Axios 12](#_Toc196470316)

[3.7.3. Инсталиране на Axios 12](#_Toc196470317)

[3.7.4. Изпращане на параметри и заглавия 12](#_Toc196470318)

[3.7.5. Предимства и недостатъци на Axios 12](#_Toc196470319)

[3.8 Fetch API: Подробно ръководство 14](#_Toc196470320)

[3.8.1. Основна употреба 14](#_Toc196470321)

[3.8.2. Параметри на fetch() 14](#_Toc196470322)

[3.8.3. Предимства на Fetch API 15](#_Toc196470323)

[3.8.4. Недостатъци на Fetch API 15](#_Toc196470324)

[3.9 jQuery 15](#_Toc196470325)

[3.9.1. Предимства на jQuery 16](#_Toc196470326)

[3.9.2. Недостатъци на jQuery 16](#_Toc196470327)

[3.10 Фреймуъркове 17](#_Toc196470328)

[3.10.1. Типове уеб фреймуъркове 17](#_Toc196470329)

[3.10.2. Предимства на фреймуърковете 19](#_Toc196470330)

[3.10.3. Недостатъци на фреймуърковете 20](#_Toc196470331)

[3.10.4. Примери за приложения на фреймуърковете 20](#_Toc196470332)

[3.11 Примери за приложения на REST API 21](#_Toc196470333)

[4. Анализ на съществуващи решения 21](#_Toc196470334)

[4.1 Архитектурен стил 22](#_Toc196470335)

[4.2 Технологии за клиентската част 23](#_Toc196470336)

[4.2.1. React: 23](#_Toc196470337)

[4.2.2. Vue.js: 23](#_Toc196470338)

[4.3 Технологии за сървърната част 23](#_Toc196470339)

[4.3.1. Node.js: 23](#_Toc196470340)

[4.3.2. Express.js: 24](#_Toc196470341)

[4.4 Система за управление на бази данни 24](#_Toc196470342)

[4.4.1. MongoDB: 24](#_Toc196470343)

[4.4.2. Осигуряване на сигурност 24](#_Toc196470344)

[4.5 Тестова среда 25](#_Toc196470345)

[5. Практическа част 25](#_Toc196470346)

[5.1 Въведение 25](#_Toc196470347)

[5.2 Изграждане на архитектурата и избор на технологии 25](#_Toc196470348)

[5.3 Проблеми при интеграция с външен REST API и тяхното решение 26](#_Toc196470349)

[5.3.1. Реализация на backend частта 26](#_Toc196470350)

[5.3.2. Реализация на frontend частта 28](#_Toc196470351)

[5.3.3. Тестване и отстраняване на проблеми 29](#_Toc196470352)

[5.3.4. Резултати и изводи 29](#_Toc196470353)

[6. Заключение 30](#_Toc196470354)

[6.1 Основни изводи 30](#_Toc196470355)

[6.2 Препоръки и бъдещи подобрения 30](#_Toc196470356)

[7. Литература: 32](#_Toc196470357)

# Увод

Съществуването на динамични уеб приложения е не само значимо, но и наложително в съвременния свят, където информацията мигновено и в реално време взаимодейства с потребителите. Въвеждането на технологии, които позволяват на клиентите да взаимодействат с данни по нов, по-гъвкав и интуитивен начин, е ключов момент в развитието на софтуерната индустрия. Настоящият дипломен проект е фокусиран върху използването на REST API от страна на клиента, което представлява важен аспект от тези иновации.

API (Application Programming Interface) е интерфейс, който позволява на различни софтуерни приложения да комуникират помежду си. REST (Representational State Transfer) е архитектурен стил за създаване на мрежови услуги, който използва HTTP протокола за обмен на данни. Основната цел на проекта е да се разработи уеб приложение, което динамично ще извлича и визуализира данни от съществуващ REST API, представяйки информация в удобен и лесен за ползване интерфейс.

Използването на JavaScript и свързаните технологии като AJAX, jQuery и JSON не само опростява кода, но и повишава производителността на клиентските приложения. Настоящият проект ще предостави поглед върху начина, по който REST API функционира и как клиентите могат да взаимодействат с него, за да извлекат необходимата информация.

Допълнително, проектът ще отрази развитието на приложението и ще анализира предизвикателствата, които възникват при работа с API, като управление на асинхронни заявки, обработка на данни и визуализация на информацията за потребителите. Приложението, което ще разработим, ще бъде основано на училищната информационна система, предоставяйки информация за ежедневното и седмичното разписание на учениците.

Теоретичната част на проекта ще разгледа основните концепции на REST API, като ще осигури задълбочено разбиране на принципите и протоколите, които позволяват на данните да бъдат манипулирани в реално време. Практическата част ще демонстрира фактическото приложение на тези концепции, в включително примери за код и функционалности, които ще бъдат реализирани.

Проектът е не само теоретично упражнение, а и важна стъпка за подготовка на студентите в областта на приложното програмиране. Той ще предостави знания и умения, които ще бъдат полезни както за бъдещи академични начинания, така и за професионално развитие в динамично променящата се технологична среда.

Дипломният проект за използване на REST API от страна на клиента е актуален и изключително полезен в контекста на текущите тенденции в уеб разработването. Съсредоточаването върху приложения, които предлагат динамична и интерактивна работа с данни, е не само предизвикателство, но и възможност за иновации и оптимизация на работния процес, който ще демонстрира възможността за обучение и професионално развитие в областта на информационните технологии.

# Цел и задачи на проекта

## Цел на проекта

Основната цел на дипломния проект е да разработи приложение, което да демонстрира практическото приложение на REST API (Representational State Transfer Application Programming Interface) с конкретна насоченост към училищни информационни системи. Проектът цели да осигури динамичен уеб интерфейс, който позволява извличане и визуализация на данни относно учебния процес, като заданията за учители, графици за занимания и т.н. Тази разработка не само ще подпомогне учениците в достъпа до важна информация, но и ще служи като практически пример за работа с API технологии, JavaScript и свързани протоколи.

## Задачи на проекта

За постигането на основната цел на проекта, трябва да бъдат решени следните конкретни задачи:

### ****Проучване на REST API****:

* Изследване и описание на принципите на работа на REST API, включително комуникационни протоколи, структура на заявките и отговорите, както и статус кодовете.
* Анализ на различни библиотеки и инструменти, използвани за взаимодействие с APIs, с акцент на JavaScript и популярните фреймуъркове.

### ****Анализ на съществуващи училищни информационни системи****:

* Извършване на анализ на налични информационни системи и приложения, които използват REST API за предоставяне на данни.
* Сравнение на функционалността на различни системи и определяне на конкретните нужди на своето училище.

### ****Избор на технологии****:

* Определяне на технологичен стек, включващ фронтенд библиотеки и фреймуъркове (например React, Angular или Vue.js), които ще се използват за разработката на уеб приложението.
* Решаване на въпроси относно интеграцията на приложението с текущата ИТ инфраструктура на училището.

### **Разработка на приложение**:

* Създаване на основната структура на уеб приложението, включваща HTML, CSS и JavaScript.
* Разработка на функционалностите за извличане и визуализация на данни от REST API.
* Интегриране на динамични елементи и интерактивни компоненти, за да се подобри потребителското изживяване.

### ****Тестиране и валидиране на приложението****:

* Провеждане на тестове за функционалност и производителност, за да се гарантира, че приложението работи оптимално и е надеждно.
* Събиране на обратна връзка от потребителите и прилагане на подобрения въз основа на техните коментари.

### ****Демонстрация на проекта****:

* Подготовка на презентация, в която да се представи разработеното приложение, основните му функции и приложимост в реалната учебна среда.
* Създаване на потребителска документация, която да улесни потребителите в работата с приложението.

# Теоретични основи

## Въведение в REST API

REST (Representational State Transfer) е архитектурен стил, който предоставя правила и принципи за обмен на данни между клиент и сървър в мрежови приложения. Използването на REST API е станало стандарт в разработката на уеб приложения, особено поради своята простота и лекота на интеграция. Основната идея на REST е да предостави голяма гъвкавост при извличането, манипулирането и представянето на данни, използвайки стандартни HTTP методи.

REST API работи на основата на определен набор от правила, които позволяват на разработчиците да взаимодействат с данни в мрежата. Прилагането на REST архитектура дава възможност за изграждане на приложения, които могат да се мащабират и да се развиват с времето.

## Основни принципи на REST

REST API функционира на базата на следните основни принципи:

* + **Клиент-сървър архитектура:** REST се основава на разделянето на клиента и сървъра. Клиентът е отговорен за потребителския интерфейс и опит, докато сървърът управлява данните и логиката на приложението. Тази отделеност позволява да се променят и мащабират клиентските приложения независимо от сървърните.
  + **Безсъстояние (Stateless)**: Всеки обмен между клиента и сървъра е независим от предходния. Сървърът не пази информация за статуса на клиента, което на свой ред ускорява обработката на заявки и намалява сложността на системата.
  + **Кеширане**: За да се повиши производителността, отговорите от сървъра могат да бъдат кеширани. Това означава, че клиентите могат да съхраняват отговорите за определено време и да ги използват повторно, намалявайки така броя на заявките към сървъра.
  + **Псевдосистеми**: Ресурсите в REST API обикновено са идентифицирани чрез URL адреси, а действията върху тях се извършват чрез HTTP методите (GET, POST, PUT, DELETE).

## Формати за обмен на данни

Най-често използваните формати за обмен на данни в REST API са JSON (JavaScript Object Notation) и XML (eXtensible Markup Language). JSON е предпочитан там, където бързината и простотата са важни, докато XML предоставя по-строга структура, но е по-тежък. В случай на уеб приложения, JSON предпазва от усложнения и ускорява обмена на данни между клиент и сървър.

### JSON:

* Лек и четим формат за данни.
* Лесно се интегрира със JavaScript.
* Позволява предаване на сложни структури от данни.

### XML:

* Строга структура и формат.
* По-подходящ за обмен на данни, изискващи сложна релационна структура.
* Може да бъде излишен за много уеб приложения.

## Как JSON и AJAX спомагат за разработката на RESTful приложения

Развитието на уеб приложения изисква ефективно и гъвкаво взаимодействие между клиент и сървър. В този контекст, JSON (JavaScript Object Notation) и AJAX (Asynchronous JavaScript and XML) играят ключова роля, особено когато става въпрос за създаването на RESTful API. Чрез използването на JSON и AJAX, разработчиците могат да създават динамични, интерактивни приложения, които предлагат отличен потребителски опит, без да изискват презареждане на страницата. В следващите секции ще разгледаме как JSON и AJAX подпомагат разработката на RESTful приложения, тяхната структура и приложение.

### Какво е AJAX?

AJAX е техника, която позволява на уеб приложенията да обменят данни с сървъра асинхронно, без необходимост от презареждане на страницата. Тази функционалност е от ключово значение за динамичните уеб приложения, които изискват чести актуализации на данни.

### Как работи AJAX?

AJAX работи чрез използването на JavaScript, за да изпраща HTTP заявки до сървъра. Тези заявки могат да бъдат от различен тип, включително GET и POST, и след получаване на отговор, AJAX позволява на разработчика да обработи данните и да актуализира част от уеб страницата.

### Предимства на AJAX

* Подобрено взаимодействие: AJAX позволява на потребителите да взаимодействат с уеб приложения, без неприятността от презареждане на страницата.
* По-добра производителност: Чрез асинхронния обмен на данни, приложенията могат да реагират по-бързо на потребителски действия и да дават мигновен отговор.
* Справяне с различни формати: AJAX може да обработва не само JSON, но и XML, HTML и плаващи текстови формати.

### Взаимодействие между JSON и AJAX в RESTful приложения

Как взаимодействат?

Комбинацията от JSON и AJAX е основна за разработката на RESTful API, тъй като клиентите могат динамично да извличат и манипулират данни. Системата работи по следния начин:

* Клиентът отправя AJAX заявка до RESTful API, изисквайки данни.
* Сървърът отговаря с данни форматирани в JSON.
* Клиентът обработва отговора и актуализира потребителския интерфейс, основавайки се на получените данни.

Пример за приложение

В училищна информационна система, директорът може да поиска списък с учениците. При отправяне на AJAX заявка до API, системата връща JSON данни с информация за всеки ученик. Приложението след това може да визуализира списъка с ученици на уеб страницата, като обновява само необходимата част без презареждане.

Необходимостта от контрол на грешки

При работа с AJAX е важно да се предвидят различни ситуации, като мрежови проблеми или проблеми със сървъра. Проверките за грешки трябва да бъдат включени, за да се уверите, че крайните потребители получават информация за проблема по приятелски начин.

Значение и употреба

Използването на JSON и AJAX е от съществени значения за разработката на RESTful приложения, предлагайки гъвкаво и интерактивно взаимодействие между клиент и сървър. JSON служи като идеален формат за предаване на данни, а AJAX позволява асинхронно извличане и обработка на информация, придавайки динамичност на уеб приложенията. Широкото приемане на тези технологии е причина за популярността на RESTful API в модерната уеб разработка и е предизвикало значителна иновация в начина, по който взаимодействаме с интернет ресурси.

## Работа с HTTP методи

REST API използва следните основни HTTP методи, за да извършва действия върху ресурси:

* **GET**: Извлича ресурси от сървъра. Подходящ за операции, които не променят състоянието на ресурсите.
* **POST**: Създава нов ресурс на сървъра. Използва се, когато клиентът иска да добави нови данни.
* **PUT**: Актуализира съществуващ ресурс на сървъра. Често се използва за актуализиране на информация за вече съществуващи ресурси.
* **DELETE**: Изтрива ресурс от сървъра. Използва се за премахване на ненужни данни.

Тези методи осигуряват универсален начин за взаимодействие с ресурсите и позволяват на разработчиците да изградят ваши приложения с CRUD (Create, Read, Update, Delete) функционалности.

## Свързани технологии и библиотеки

В работата с REST API, JavaScript е основният език за писане на клиентски код, а различни библиотеки и фреймуъркове правят работата с API значително по-лесна. Най-популярните библиотеки за работа с API включват:

### Axios:

* + Библиотека, която улеснява выполнения HTTP заявки в JavaScript.
  + Поддържа обещания и предоставя прост интерфейс за управление на асинхронни операции.

### Fetch API:

* + Вградено решение в модерните браузъри за работа с HTTP заявки.
  + Предлага обещания и е по-съвременно решение от стария XMLHttpRequest.

### jQuery:

* + Макар по-стара, jQuery остава популярна библиотека за работа с AJAX, позволявайки лесни манипулации на DOM и работа с REST API.

## Axios: Библиотека за HTTP заявки в JavaScript

Axios е популярна библиотека за извършване на HTTP заявки в JavaScript, която предлага лесен и консистентен начин за взаимодействие с RESTful API. Библиотеката е построена върху Promise API, което позволява асинхронно обработване на заявки и отговори. Axios е широко използван в разработката на уеб приложения, особено за приложения, изградени с фреймуъркове като React, Vue.js и Angular. В следващите секции ще разгледаме какво е Axios, неговите основни функции, предимства, недостатъци и примери за приложение.

### Какво е Axios?

Axios е клиент за HTTP заявки, създаден с цел опростяване на обмена на данни между клиент и сървър. Тя е независима от фреймуъркове библиотека, която може да бъде интегрирана лесно във всяко JavaScript приложение. Axios поддържа Promise, което означава, че разработчиците могат да използват .then() и .catch() за обработка на успешни и неуспешни заявки, което улеснява четенето и поддържането на кода.

### Основни характеристики на Axios

* Promisified API: Axios връща промиси, позволявайки на разработчиците да работят с асинхронния код по по-лесен начин.
* Поддръжка на JSON: Axios автоматично преобразува JSON данни, което опростява обработката на входящи и изходящи данни.
* Създаване на инстанции: Поддържа създаване на инстанции с индивидуални настройки, което е полезно при взаимодействие с различни API.
* Отменяне на заявки: Axios позволява отменяне на активни заявки, което е полезно при работа с динамични приложения.
* Поддръжка на интерсептори: Разработчиците могат да добавят интерсептори, за да обработват заявки или отговори преди да бъдат изпратени или получени.

### Инсталиране на Axios

Axios може да бъде инсталиран лесно с помощта на npm (Node Package Manager). След успешна инсталация, Axios може да бъде импортиран в JavaScript файловете.

### Изпращане на параметри и заглавия

Чрез Axios е възможно да се изпращат URL параметри и заглавия с заявките, което е полезно за настройки и удостоверяване

### Предимства и недостатъци на Axios

Предимства на Axios

* Лесен за употреба: Axios предлага интуитивен и прост API за извършване на HTTP заявки, което го прави удобен за разработчиците, особено за начинаещи.
* Поддръжка на промиси: Axios е основан на Promise, което позволява асинхронно обработване на HTTP заявки. Разработчиците могат лесно да използват функции .then() и .catch() за управление на успешни и неуспешни отговори.
* Автоматично преобразуване на JSON: Axios автоматично обработва JSON данни, така че разработчиците не трябва да извършват ръчно сереализация и десереализация на данни. Това опростява взаимодействието с RESTful API.
* Лесно конфигуриране: Axios позволява настройване на глобални параметри, като заглавия и таймаути, което улеснява бързата конфигурация на API взаимодействия.
* Интерсептори: Axios предлага функционалността на интерсептори, които позволяват обработката на заявките и отговорите от API. Това е полезно за внедряване на аутентикация, бизнес логика или обработка на грешки.
* Отменяне на заявки: Axios предлага функция за отменяне на активни HTTP заявки, което е особено полезно при динамични приложения, където потребителите могат да изпращат множество заявки последователно.
* Поддръжка на браузери: Axios работи с активно възстановяване и поддържа стари браузъри, което подобрява съвместимостта.
* Подобрена обработка на грешки: Axios прави разграничение между различни типове грешки, като предоставя по-информативно управление на грешки, което улеснява отстраняване на проблеми.

Недостатъци на Axios

* Размер на библиотеката: Axios е библиотека и добавя допълнителен размер в проекта. За проекти, които трябва да бъдат възможно най-леки, това може да бъде недостатък в сравнение със стандартните методи за работа с Fetch API.
* Зависимост: Когато използвате Axios, добавяте външна зависимост в проекта си, която трябва да поддържате. Ако библиотеката подготвя нови версии, проектът може да изисква актуализации.
* Малко по-неизчисляем вход на програмни скриптове: Въпреки че Axios се справя добре при работа с промиси, новите разработчици може да срещнат объркване между синхронни и асинхронни методи за работа, ако не са запознати с концепцията на промиси или асинхронност.
* Необходимо конфигуриране за изключения: Често, при работа с Axios, е необходимо да се конфигурират обработчици на грешки, тъй като Axios не автоматично обработва определени грешки, свързани с мрежата, оставайки на разработчика да определи как да реагира.
* Отказ от поддръжка на XML формати: Докато Axios е насочен най-вече към работа с JSON, ако имате нужда от работа с XML, може да се наложи допълнително конфигуриране и обработка.

## Fetch API: Подробно ръководство

Fetch API е модерен интерфейс за работа с HTTP заявки от страна на клиента, предоставяйки по-лесен и гъвкав начин за извършване на мрежови заявки в сравнение с предшественика си - XMLHttpRequest (XHR). Чрез Fetch API разработчиците могат да правят асинхронни изисквания, които се основават на обещания (Promises), което позволява по-добро управление на асинхронния код и обработка на отговори.

### Основна употреба

Fetch API предлага метод fetch(), който използва синтаксис, наподобяващ методите на JavaScript. По принцип, fetch() приема URL адрес, като за опционален втори параметър може да се получи обект с настройки, определящи HTTP метода (GET, POST и т.н.).

### Параметри на fetch()

Методът fetch() може да приема втори параметър, който е конфигурационен обект, включващ:

* method: HTTP метода (GET, POST, PUT, DELETE и т.н.)
* headers: HTTP заглавия
* body: информацията, която искаме да изпратим (при POST, PUT и др.)
* mode: режим на заявката (cors, no-cors, same-origin)
* cache: правила за кеширане на отговорите

### Предимства на Fetch API

* По-модерен синтаксис: Използването на Promises улеснява разбирането и работата с асинхронен код.
* По-добро управление на грешки: Възможността за обработка на отговори правилно намалява броя на грешките.
* Поддръжка на различни формати: Лесно извличане на различни формати на данни.
* Кросплатформена съвместимост: Работи на множество платформи, включително браузъри и мобилни приложения.

### Недостатъци на Fetch API

* Без автоматично управление на сесии: Въпреки че се справя с CORS, разработчиците трябва да управляват сесиите и кеша.
* Не поддържа старите браузъри: Потребителите на Internet Explorer, например, ще срещнат проблеми с Fetch API. Възможно е да е необходимо полифилно решение или алтернатива (напр. Axios).

## jQuery

* jQuery е бърза, малка и функционално поддържана JavaScript библиотека, разработена с цел да опрости обработката на HTML документи, управление на събития, анимация и взаимодействие с API. Създадена от Джон Резиг през 2006 година, jQuery бързо се утвърди като стандарт в света на уеб разработката. В момента е една от най-предпочитаните библиотеки за работа с HTML и DOM (Document Object Model).
* Библиотеката е проектирана така, че да улесни работата с JavaScript, предлагайки синтаксис, който позволява на разработчиците бързо и лесно да манипулират мрежови елементи. Интуитивният му подход намалява нуждата от сложни кодове, като в същото време предоставя мощни функционалности и функционалности, от една страна, и стабилна производителност, от друга.

### Предимства на jQuery

Простота и Леснота на Използване

Едно от основните предимства на jQuery е неговата семантична простота. С минимален код, разработчиците могат да постигнат сложни взаимодействия.

Крос-браузърна Подкрепа

jQuery осигурява унифициран интерфейс за работа с различни браузъри, което е особено важно предвид разликите между тях. Разработчиците не трябва да се тревожат за различия в поведението на JavaScript между браузъри, а могат да се съсредоточат върху функционалността на своите приложения.

Удобно Управление на Събития

jQuery предоставя опростено API за управление на събития, което прави добавянето на обработчици на събития лесно и интуитивно. Системата за управление на събития позволява да се добавят множество обработчици и дори да се отменят събития при необходимост.

AJAX интеграция

jQuery предоставя лесен и интуитивен начин за работа с AJAX, позволявайки на разработчиците да извършват асинхронни заявки с малко усилия.

Широка общност и поддръжка

Съществува множество ресурси, уроци и примери от общността на разработчиците, което прави намирането на помощ и обучения много достъпно.

### Недостатъци на jQuery

Размер на библиотеката

Въпреки че jQuery е малка в сравнение с много други библиотеки, добавя допълнителен размер на файла, което може да повлияе на времето за зареждане на страницата, особено при по-малки проекти.

Изменение в популярността

С напредването на технологиите, нови фреймуъркове и библиотеки, като React и Vue.js, започнаха да заменят jQuery за много проекти. Успехът на jQuery може да намалее с времето.

Поддръжка на стари браузъри

jQuery не поддържа Internet Explorer 8 и по-долу, което представлява проблем за приложения, които все още искат да бъдат достъпни за потребители на стари браузъри.

Възможни конфликти с други библиотеки

При използване на jQuery в комбинация с други JavaScript библиотеки, могат да възникнат конфликти, особено ако библиотеки използват подобни имена за методи.

Обработване на промените в JavaScript

С развитието на ES6 и новите функции на JavaScript, jQuery става все по-малко необходимо, тъй като много от функциите, които предоставя, вече са нативно подкрепяни от JavaScript.

## Фреймуъркове

Фреймуърковете са основен компонент в разработването на съвременни уеб приложения. Те предлагат структурирана среда за изграждане на динамични уеб приложения, като предоставят набор от инструменти, библиотеки и стандартизиран подход към разработването. Фреймуърковете могат да бъдат определени като предварително написан код, който служи за основа и ускорява процеса на разработване. В тази част ще разгледаме основните типове фреймуъркове, техните предимства и недостатъци, както и примери за популярни решения.

### Типове уеб фреймуъркове

Фронтенд фреймуъркове

* Тези фреймуъркове обикновено се използват за разработване на клиентската част на уеб приложения. Те се фокусират върху взаимодействащия интерфейс (UI), управлението на събития и манипулирането на DOM.
* Примери:
* **React**: Разработен от Facebook, React е библиотека за изграждане на потребителски интерфейси, която разделя приложението на компоненти за по-добра организираност и повторна употреба на кода.
* **Angular**: Поддържан от Google, Angular е пълен фреймуърк за изграждане на динамични уеб приложения, предлагащ модулна структура, двупосочно свързване на данни и интегрирана поддръжка за приложения на базата на маршрути.
* **Vue.js**: Vue е прогресивен фреймуърк, който е лесен за интегриране с други проекти и библиотеки. Той предлага реактивност и подход за компонентно базирано развитие.
* **Бекенд фреймуъркове**
* Тези фреймуъркове се използват за разработка на сървърната част на приложението. Те се фокусират върху управление на база данни, обработка на бизнес логика и API интеграции.
* Примери:
* **Express.js**: Минималистичен фреймуърк за Node.js, който опростява разработването на сървърни приложения и API.
* **Django**: Python-базиран фреймуърк, който предлага високостепенна интеграция с бази данни, система за аутентикация и удобства за работа с формуляри.
* **Ruby on Rails**: Фреймуърк, който следва конвенцията преди конфигурацията, предоставяйки мощни инструменти за бързо изграждане на приложения.

Фулстек фреймуъркове

* Комбинират фронтенд и бекенд технологии, което позволява разработване на цялостни приложения от една платформа.
* **Примери**:
* **Meteor.js**: Фреймуърк за разработване на приложения в реaлно време, който предоставя инструменти за фронтенд и бекенд разработка.
* **MEAN stack**: Комбинация от MongoDB, Express.js, Angular и Node.js, предлагаща пълна среда за разработка на JavaScript-базирани приложения.

### Предимства на фреймуърковете

Структурираност и организация на кода

* Фреймуърковете предлагат норми и структури, които помагат на разработчиците да поддържат кода организиран и удобен за работа. Това е особено полезно при работа в екип, тъй като различни членове на екипа могат да се придържат към същите стандарти.

Ускорено разработване

* С предоставените инструменти и библиотеки, фреймуърковете ускоряват времето за разработка, тъй като разработчиците могат да избягват писането на повторяеми задачи и функции от нулата.

Увеличена производителност

* Много фреймуъркове предлагат оптимизации и интеграции, които подобряват производителността на приложенията. Например, рекомпилация на кода и опаковане на ресурсите.

Поддръжка на общността и ресурси

* Популярните фреймуъркове имат активни общности и множество ресурси за обучение, което помага на новите разработчици да се запознаят бързо с инструментите.

Вградена сигурност

* Много фреймуъркове предлагат вградени механизми за сигурност, които помагат за защита на приложенията от несанкциониран достъп и уязвимости.

### Недостатъци на фреймуърковете

Крива на обучение

* Някои фреймуъркове изискват време за обучение и свикване с техния синтаксис и концепции, особено за начинаещи разработчици.

Ограничения

* Прилагането на определен фреймуърк може да наложи ограничения върху начина, по който могат да се структурират или развиват приложенията. Например, забраненият достъп до определени функции или наличието на предварително зададени шаблони.

Производителност при излишък

* При работа с определени фреймуъркове, производителността може да се влоши, ако приложението не е оптимизирано, особено в случаи на излишна абстракция или ненужни сложни структури.

Зависимост от фреймуърка

* Когато проектът е силно зависим от конкретен фреймуърк, мигрирането или преминаването към нови технологии може да бъде трудно и времеемко.

Версионна несъответствие

* Фреймуърковете постоянно се развиват, което може да доведе до конфликти с кодовете при обновления или версии, които имат несъвместимости с предишните.

### Примери за приложения на фреймуърковете

Фреймуърковете се използват в широк спектър от приложения, включително:

* **Уеб сайтове и приложения**: Създаване на сложни уеб приложения с интерактивни интерфейси и динамично съдържание.
* **Мобилни приложения**: Много фреймуъркове предлагат решения за изграждане на хибридни приложения, които работят на различни платформи.
* **API разработка**: Уеб фреймуъркове често се използват за изграждане на RESTful или GraphQL API.

## Примери за приложения на REST API

REST API намира приложение в множество области — от взаимодействия в социални мрежи до управление на бази данни. Примери включват:

* + **Социални платформи**: API на Facebook, Twitter и Instagram предоставят на разработчиците достъп до потребителски данни и публикации.
  + **Електронна търговия**: API на eBay и Amazon позволяват на приложенията да извършват търсене на продукти и управление на поръчки.
  + **Училищни информационни системи**: Използването на REST API в училищното образование позволява на учениците и учителите да взаимодействат с бази данни за оценяване, разписание и други ресурси.

# Анализ на съществуващи решения

В съвременната разработка на уеб приложения, използващи REST API, съществуват множество решения и платформи. Анализът на съществуващите решения е ключова стъпка в процеса на проектиране и разработка на нови приложения, тъй като той ни позволява да идентифицираме успешни практики, да избегнем повтарянето на вече решени проблеми и да се възползваме от иновации и технологии, които вече са доказали своята ефективност. Ето няколко популярни решения, които използват REST API, техните предимства и недостатъци, и как те могат да информират разработката на ново приложение за училищна информационна система.

* + **Google Classroom** е платформа, предназначена за управление на учебния процес, която предлага обширен набор от функционалности за учители и ученици. Платформата интегрира различни аспекти на обучението, от разпределение на задания до оценяване и комуникация.
  + **Microsoft Teams** предлага интегрирано решение за управление на образованието и етапи на обучение. Платформата предлага множество функционалности за видео конференции, управление на задания и колаборация между ученици и учители.
  + **Moodle** е платформа с отворен код за управление на обучението, известна с факта, че предоставя възможности за персонализиране в голяма степен. Moodle е основно използвана в учебни заведения по целия свят и предлага мощен набор от инструменти за обучение.
  + **Schoology** предлага платформа за управление на обучението с множество интегрирани инструменти за учители, ученици и родители. Тя се характеризира със социални функции, подобно на социални мрежи.
  + **Canvas** е иновативна платформа за управление на обучението, която съчетава нуждите на преподаватели и студенти. Платформата предлага функционалности за проектиране на курсове, управление на задания и активно взаимодействие.

## Архитектурен стил

Избраната архитектура за разработването на приложението ще бъде **модел-контролер-изглед** (Model-View-Controller, MVC). MVC е архитектурен шаблон, който разделя приложението на три компонента: модел, контролер и изглед:

* + Модел: Отговаря за управлението на данни, логическите правила и съхранението на информация.
  + Контролер: Свързва модела и изгледа, обработва входните данни и определя какво да се показва на потребителя.
  + Изглед: Представлява данните по удобен за потребителите начин, обикновено чрез графичен интерфейс.

Тази архитектура осигурява ясна разделеност на отговорностите и улеснява поддръжката и разширяемостта на приложението.

## Технологии за клиентската част

* + Клиентската част на приложението ще бъде изградена с помощта на модерна JavaScript библиотека, като **React** или **Vue.js**. Двете решения предлагат значителни предимства:

### React:

* + Популярна библиотека, разработена от Facebook, която позволява изграждане на динамични и интерактивни потребителски интерфейси.
  + Използва компоненти, които могат да бъдат многократно използвани, подобрявайки ефикасността на разработката.
  + Размерът на общността и наличието на ресурси, библиотеките и готовите примери правят React отлично решение за бързо разработване на уеб приложения.

### Vue.js:

* + По-лека алтернатива на React, известна със своята простота и опростена синтаксис.
  + Има добра документация и е лесна за интеграция в съществуващи проекти.
  + Предлага двупосочна свързаност на данните, което улеснява управлението на взаимодействията с потребителския интерфейс.

Изборът между двете библиотеки ще зависи от конкретните нужди на проекта и опита на екипа, но и двете решения предлагат мощни инструменти за работа с REST API.

## Технологии за сървърната част

* + За сървърната част на приложението ще изберем **Node.js** заедно с **Express.js**. Node.js е JavaScript среда за изпълнение, която позволява разработването на сървърни приложения, а Express.js е минималистичен уеб фреймуърк за Node.js, удобен за изграждане на RESTful API.

### Node.js:

* + Позволява разработчиците да пишат целия код на приложението (както на клиента, така и на сървъра) с един единствен език - JavaScript, което опростява разработката и поддръжката.
  + Осигурява перформанс и мащабируемост чрез асинхронна употреба на I/O операции.

### Express.js:

* + Облекчава създаването на маршрутни файлове и управляване на HTTP заявки и отговори.
  + Предлага множество middleware библиотеки, които могат да бъдат добавени за разширяване на функционалността на сървъра.
  + Подкрепя интеграцията с различни шаблони за рендериране и работа с бази данни.

## Система за управление на бази данни

* + За съхранение на данните, свързани с учебния процес, ще използваме **MongoDB**, което е NoSQL база данни, оптимизирана за работа с документи и големи обеми от неструктурирани данни.

### MongoDB:

* + Позволява гъвкаво моделиране на данните, което е особено полезно, ако структурата на данните ще се променя през времето.
  + Осигурява бърз достъп до данните и добро представяне при работа с мащабни приложения.
  + Подкрепя JSON формат за съхранение на данни, което улеснява интеграцията с JavaScript приложения.

Може да се обмисли използването на Mongoose, библиотека за Node.js, която предлага моделизация на данни в MongoDB и опростява работата с бази данни.

### Осигуряване на сигурност

Сигурността е критичен аспект при разработването на уеб приложения, особено когато се извършват операции с чувствителни данни. Основните мерки, които трябва да бъдат предприети, включват:

* + **Аутентификация и авторизация**: Използване на библиотека като **jsonwebtoken** за създаване на JWT токени, осигуряващи сигурен достъп до ресурсите на API.
  + **Валидация на входните данни**: С применението на библиотеката **Joi** или **express-validator** за валидация на входящите данни от потребителите.
  + **Шифроване на пароли**: Употреба на библиотеки като **bcrypt** за хеширане и безопасно съхранение на потребителските пароли.

## Тестова среда

Качеството на приложението ще бъде осигурено и посредством тестова среда. За целите на тестовете ще използваме инструменти за юнит тестове и интеграционни тестове, включително:

* + **Jest**: Модулен тестов фреймуърк за JavaScript, който позволява бързо и лесно писане на тестове.
  + **Mocha** и **Chai**: Осигуряват аналогични функционалности, позволяващи тестване на API, използващо различни методи и структури.

# Практическа част

## Въведение

В рамките на практическата част на дипломния проект се заех с разработването на уеб приложение, което има за цел да извлича и визуализира данни за дневното и седмичното разписание на ПГЕЕ – гр. Банско. Основната задача беше да се използва съществуващ REST API, предоставен от училището, и да се създаде модерен, динамичен и удобен за ползване уеб интерфейс, който да представя тази информация на потребителите. За реализацията на проекта избрах да използвам съвременни уеб технологии, като за backend частта се спрях на Django, а за frontend – на Vue.js и Axios. В следващите редове ще опиша подробно процеса на изграждане на приложението, трудностите, с които се сблъсках, и решенията, които приложих.

## Изграждане на архитектурата и избор на технологии

Още в началото на работата си по проекта анализирах възможните подходи за реализиране на връзката между клиента и външния REST API. След като се запознах с изискванията на заданието и спецификата на предоставения API, взех решение да разделя приложението на две основни части – backend и frontend. За backend частта избрах Django, тъй като това е утвърдена и добре документирана платформа, която позволява бързо изграждане на уеб приложения и лесна интеграция с външни услуги. За frontend частта се спрях на Vue.js, тъй като тази библиотека предоставя възможност за създаване на динамични и реактивни потребителски интерфейси, а Axios избрах като лесен и удобен инструмент за изпращане на HTTP заявки.

## Проблеми при интеграция с външен REST API и тяхното решение

Още при първите опити да осъществя директна връзка между frontend-а и външния REST API на сайта на ПГЕЕ – гр. Банско се сблъсках с един от най-често срещаните проблеми при работата с външни услуги – CORS политиката на браузъра. При опит да изпратя заявка директно от JavaScript към външния сървър, браузърът блокира заявката и върна съобщение за грешка, че липсва необходимият CORS header. Това наложи да потърся алтернативно решение, което да позволи на приложението ми да достъпва данните от външния API, без да нарушава политиките за сигурност на браузъра.

След проучване на възможните подходи, реших да реализирам така наречения "прокси" слой в backend-а. Това означава, че вместо frontend-ът да се опитва директно да комуникира с външния API, той изпраща заявките си към локалния сървър (Django), който от своя страна препраща тези заявки към външния REST API, получава отговора и го връща обратно към клиента. По този начин всички заявки към външния сървър се изпълняват от backend-а, където CORS ограниченията не важат, а frontend-ът комуникира само с локалния сървър, което е напълно разрешено от браузъра.

### Реализация на backend частта

След като анализирах структурата и възможностите на външното REST API на ПГЕЕ – гр. Банско, взех решението да реализирам отделни изгледи във Django backend-а за трите основни типа данни, които са достъпни чрез този API. По този начин постигнах по-ясна структура на кода и улесних бъдещата му поддръжка и разширяване.

Първата стъпка беше да създам самите изгледи във файла **views.py**. Вместо да използвам един общ прокси изглед, както беше първоначалният подход, се спрях на реализирането на три отделни функции – по една за всяка от основните операции на API-то: получаване на настройки (параметри), получаване на седмично и получаване на дневно разписание. Всяка от тези функции обработва заявката, изпраща необходимата HTTP заявка към сървъра на ПГЕЕ и връща получения отговор във вид, подходящ за фронтенда.

Например, за получаването на **параметрите за настройка** реализирах следния изглед:

def params\_view(request):

external\_url = "https://pgeebansko.org/info/api/Params/"

response = requests.get(external\_url)

return JsonResponse(response.json(), safe=False)

Този изглед използва библиотеката requests, за да изпрати GET заявка към съответния endpoint на външния сървър, след което връща получения JSON към клиента. Така фронтендът получава всички необходими параметри за настройка на приложението в стандартен JSON формат.

За показване на **седмичното разписание** разработих отделен изглед, който приема като параметри избраните клас и номер на седмицата. На базата на тези данни формира съответния URL за външния API, изпраща заявката и връща резултата:

def curriculum\_view(request, week, group):

external\_url = f"https://pgeebansko.org/info/api/Curriculum/{week}/{group}/"

response = requests.get(external\_url)

return JsonResponse(response.json(), safe=False)

Това позволява на клиента лесно да визуализира данните за всяка седмица и за всеки клас с всички детайли – предмети, кабинет, учители и други.

За **дневното разписание** реализацията е сходна – използвам отделен изглед, който изпраща GET заявка към правилния endpoint според избрания ден. Кодът изглежда така:

def timeline\_view(request):

external\_url = "https://pgeebansko.org/info/api/TimeLine/"

response = requests.get(external\_url)

return JsonResponse(response.json(), safe=False)

Този подход осигурява лесен достъп от фронтенда до пълната информация за часовете – име на часовия блок, начален и краен час, както и други специфики.

Като резултат от тази архитектура всеки изглед е специализиран за обработката на конкретен вид заявка, което прави кода подреден и четим. В очаквания JSON отговор от оригиналния API няма нужда от повторно ръчно форматиране и директно го изпращам до клиента. В процеса на интеграция на тези изгледи се убедих, че по отделни функции се постига максимална яснота, като се избягват излишни проверки и допълнителни условности, които биха се наложили при използването на един общ прокси.

След финализиране на изгледите добавих нужната маршрутизация във файла **urls.py**. За всеки от разработените изгледи дефинирах отделен route, така че да мога лесно да насочвам фронтенда към коректния местен адрес, който препраща заявката към съответния endpoint на официалния сървър. Това не само повиши сигурността, като елиминира нуждата frontend-ът да знае пълния адрес на външното API, но и позволи при евентуални бъдещи промени да се правят корекции на едно място – във backend-a.

Този подход направи интеграцията между frontend-а и backend-а структурирана, сигурна и лесна за поддръжка, като всеки компонент поема точно своята роля – изгледите във Django отговарят за комуникацията с външния API и предоставят на потребителя подходящите данни в стандартизиран формат.

### Реализация на frontend частта

След като backend-ът беше готов и можеше успешно да препраща заявки към външния REST API, се заех с изграждането на потребителския интерфейс. За целта създадох HTML шаблон (info\_main.txt), в който дефинирах основната структура на приложението. В този шаблон включих всички необходими елементи за визуализация на разписанието, както и специални контейнери за анимирания часовник, който беше част от изискванията за динамичен интерфейс.

Във файла info\_main.js реализирах основната логика на Vue компонента. Там дефинирах всички необходими данни и методи, които управляват състоянието на приложението и обработват получените от backend-а данни. Използвах Axios за изпращане на заявки към локалния сървър, като по този начин осигурих асинхронно извличане на информацията за разписанието. Пример за такава заявка е следният фрагмент:

axios.get('/proxy/info/api/Curriculum/' + day + '/' + hour + '/')

.then(response => {

// обработка на данните

})

.catch(error => {

console.error(error);

});

Особено внимание отделих на интеграцията на анимирания часовник. Вместо да използвам отделен скрипт (clock.js), преместих цялата логика за обновяване и анимация на цифрите като методи на Vue компонента. Това гарантира, че часовникът ще работи коректно и ще се обновява в реално време, дори когато DOM се рендира динамично от Vue. В mounted hook-а на компонента стартирах функцията за обновяване на часа, която на всяка секунда обновява визуализацията и анимациите на цифрите.

### Тестване и отстраняване на проблеми

В процеса на разработка не липсваха предизвикателства. Още при първите опити за интеграция на frontend-а с външния API се появиха CORS грешки, които наложиха реализирането на прокси слоя. След като този проблем беше решен, се появиха и други дребни затруднения, свързани с правилното предаване на параметрите към прокси изгледа и обработката на различните типове заявки. Често се налагаше да добавям проверки и защити в кода, за да се избегнат грешки при липсващи или неправилно форматирани данни.

При интеграцията на анимирания часовник също се сблъсках с някои особености, тъй като Vue рендира DOM елементите динамично и не винаги те са налични в момента, в който се изпълнява скриптът. За да реша този проблем, преместих цялата логика за обновяване на часовника във Vue компонента и се уверих, че тя се стартира едва след като всички елементи са рендирани.

### Резултати и изводи

В резултат на всички тези стъпки успях да реализирам уеб приложение, което динамично извлича и визуализира данни от външния REST API на ПГЕЕ – гр. Банско. Приложението работи бързо и надеждно, като предоставя на потребителите актуална информация за дневното и седмичното разписание, представена в модерен и анимиран уеб интерфейс. Чрез използването на прокси слой в backend-а успешно преодолях ограниченията, наложени от CORS политиката на браузъра, а интеграцията на Vue.js и Axios направи възможно създаването на динамичен и лесен за поддръжка frontend.

Работата по този проект ми даде възможност да приложа на практика знанията си по уеб програмиране, да се сблъскам с реални проблеми и да намеря ефективни решения. Научих се да работя с REST API, да интегрирам различни технологии и да създавам приложения, които са едновременно функционални и удобни за потребителите.

# Заключение

Въз основа на проведените изследвания и реализация на дипломния проект по тема "Използване на REST API от страна на клиента", можем да направим значителни изводи относно потенциала и приложението на технологиите, стратегиите и подходите, използвани в процеса на разработка.

## Основни изводи

* **Разбиране на концепцията за REST API**: Проектът предостави дълбочинно изучаване на концепцията и принципите на REST API, които се явяват основополагащи за взаимодействието между клиент и сървър. REST API позволява на различни приложения да комуникират помежду си по стандартизиран и разбираем начин, опростявайки интеграцията и разширяемостта на системите.
* **Използване на JavaScript и свързани технологии**: Разработването на фронтенд приложение с JavaScript, заедно с технологии като JSON и AJAX, показа как можем да извлечем данни по асинхронен начин и да актуализираме потребителския интерфейс без необходимост от презареждане на страницата. Този подход увеличава динамиката и интерактивността на приложенията.
* **Придобиване на практически опит**: Практическата част на проекта демонстрираше как данните се извличат и визуализират, предоставяйки на потребителите полезна информация за училищната система. Чрез разработки на реални сценарии успяхме да демонстрираме как REST API може да се прилага в образованието, например чрез извличане на информация за разписания, оценките на учениците и други важни данни.

## Препоръки и бъдещи подобрения

* **Оптимизация на производителността**: В хода на реализацията на проекта стана ясно, че оптимизацията на производителността е критичен аспект, особено когато работим с големи обеми данни. Бъдещата работа може да включва внедряване на механизми за кеширане и оптимизиране на броя на изискванията към API.
* **Разширяване на функционалността**: Предложено е разширяване на приложението с допълнителни функции, като например интеграция на функции за анализ на данни или автоматизирано генериране на отчети. Тези функции могат да добавят значителна стойност и да улеснят управлението на данни в образователната система.
* **Експериментиране с нови технологии**: Съществуват много нови технологии и подходи, свързани с разработването на уеб приложения, които могат да помогнат за подобряване на окончателния продукт. Прилагане на фреймуърци като React или Vue.js би могло да подобри управлението на състоянието на приложението и да увеличи гъвкавостта му.

Проектът "Използване на REST API от страна на клиента" предоставя важни знания и практически умения, които са от съществено значение за бъдеща работа в областта на приложното програмиране. Не само, че успяхме да демонстрираме как можем ефективно да взаимодействаме с REST API, но и как можем да изградим интуитивни и полезни приложения, които да отговарят на нуждите на потребителите.

В заключение, настоящият дипломен проект подчертава важността на интеграцията на съвременните технологии в образователния сектор и предоставя основа за бъдещи разработки, които ще продължат да подобряват управлението и достъпа до информация в нашите училища. Той утвърждава и стратегията за учене и осъвременяване на учебните плани и методики, което е от съществено значение в контекста на бързо променящия се свят на информационните технологии.

# Литература:

1. Колисниченко, Денис, Адаптивен уеб дизайн с Bootstrap, Асеневци, С 2019
2. D.K Academy, Python – практическо програмиране. Асеневци, С 2022
3. Донълдсън, Тоби. Бързо ръководство:Програмиране с Python.АлексСофт, С 2017
4. Есканази, Аврам. Софтуерни техологии. КЛМН, С 2006
5. https://softuni.bg/blog/what-is-django
6. https://cynoteck.com/bg/blog-post/flask-vs-django/
7. https://bg.savtec.org/articles/coding/the-basics-of-rest-and-restful-api-development.html
8. https://urocibg.eu/
9. https://www.django-rest-framework.org/
10. https://softuni.bg/blog/vue-js-january-2021
11. https://bg.wikipedia.org/wiki/MySQL
12. https://priobshti.se/article/strategii-v-pomosht-na-prepodavaneto/taksonomiya-na-blum-ili-kak-da-napravim-uroka-v
13. https://html.w3schools.bg
14. https://www.inventum.bg/web-design/css/