**Имена:**  Ерик Здравков 62511, Николай Георгиев 62307

**Имейл:** [erikzdravkov99@gmail.com](mailto:erikzdravkov99@gmail.com), [nikolaimgeorgiev@abv.bg](mailto:nikolaimgeorgiev@abv.bg) **Начална година:**2020 **Програма:** бакалавър (СИ) **Курс: 3**   
**Тема: Виртуална приемна  
Дата: 2021-06-01 Предмет: w16prj\_SI \_final**

**преподавател:** доц. д-р Милен Петров

Предал (подпис): ………………………….

/*фн, имена, спец., група*/

Приел (подпис): ………………………….

/доц. *Милен Петров*/

# ТЕМА: Виртуална приемна

## Условие

“Всеки преподавател може да дефинира няколко виртуални кабинета (опашки) с различни цели - студентите се нареждат, и преподавателя избира кой/кои от тях да 'лвезнат' в кабинета. Под влизане - като влезнат в системта им дава линк към онлайн среща - може да е BBB линк, генерира линк към ZOOM, Meets, meet.jit.si, или друг хард-коднат, както и им дава код за достъп - ако е необходим;

Може да прави по зададени от преподавателя оценки - на всеки от опашката след колко време ще му дойде реда - като вариатн е преподавател ръчно да задава час в който кой кога ще влезне, или автоматично да дава по 15 минути на човек и да дава оценка за времето, което се 'чака';

Всеки потребител може да добавя и коментар, който да вижда или само преподавателя или всички на опашката от вида 'налага ми се да излезна за 30 мин', ще може ли да ме изчакате? (или да се сменя с някой...);

Да има възможност да се канят и други чакащи - временно (без губене на реда) - влизат и после излизат пак на опашката, или постоянно - влизат и после си губят реда; Да може да се покани и цялата опаша в този режим;

Да има статистика например за средно време за чакане по темата, за която чакат (например дали е защита на проект, нанасяне на оценка, консултация за реферат или друго)“

## Въведение

MeetUp е виртуална приемна, създадена от студенти за преподаватели. Meetup улеснява организацията на предстоящи изпити, контролни и изпитвания. Целта му е да подпомогне провеждането им. С MeetUp е лесно, удобно и бързо да се направи всичко необходимо за едно приятно изпитване(зависи дали студентът е учил), което не стресира студентите с излишни проблеми като „не мога да вляза в стаята“ или „стаята заби“. Основната идея на MeetUp, която ще води по-нататъшното му развитие, е ползващите системата да се чувстват все едно са в истинска приемна.

MeetUp поддържа два вида потребители – студент и преподавател. Всеки потребител се регистрира избирайки вида си потребител. За студентите се изисква и допълнителна информация – факултетен номер, специалност и курс. След регистрацията всеки потребител може да влезе в профила си чрез верни имейл и парола. След влизането следва пренасочване към потребителския панел. В него има потребителят може да намери стаите, в които участва, и информация за профила си. Когато свърши своята работа, потребителят може да излезе от профила си. Преподавателите имат възможността да създават стаи. За целта трябва да въведат информация за стаята като име на стаята, време за среща, време между срещите и начало на срещата. Също така е необходимо и да предостави ориентировъчен график на студентите, които се очаква да влязат в стаята.

След като бъде създадена, стаята ще бъде видима за всички студенти от разписанието, но няма да бъде достъпна. За целта преподавателят трябва да влезе в стаята и да натисне бутона „Старт“. След това студентите вече могат да се нареждат в опашката. Редът на студента в опашката се определя от реда му в графика. Ако са влезли двама студенти, които са съответно 3-ти и 5-ти в графика, и влезе студента на 2-ра позиция, то той ще се нареди пръв на опашката. Преподавателят може да избере да приеме следващия в опашката или който друг желае, като натисне реда му в опашката. След като времето изтече се появява съобщение. Това не пречи на преподавателя, а му помага да си организира времето по-добре. Той може да спре срещата в посочен от него момент като натисне бутона почивка. На всички чакащи се появява съобщение за почивка. Когато тя изтече отново се появява съобщение. Преподавателят може да не се съобрази с това. В удобен за него момент натиска следващ и започва срещата.

## Теория

* 1. Семантичен HTML

Проектът следва припоръките за изграждане на семантично уеб съдържание. Това означава, че се използват елементи, които имат определено значение от гладна точка на очакваното им съдържание. Това помага не само за подобряване на достъпността за хора с увреждания, но дава възможност за по-добро автоматично обхождане и таргетиране на информация в страниците.

Използвани са section и article таговете съответно за обособяване на отделни секции от свързани данни и секции със самостоятелни данни. За всеки от тях има и съотвестващ header елемент с heading с подобаващо ниво на “дълбочина“.

За добавяне на семантично значение към отделните компоненти на формите са ползвани fieldset таговете, чрез които се групират полета за входни данни за сродна информация. Освен това за всеки input е добавен и съотвенет label таг, за определяне на очаквана информация. А за изпращане на формата е използва бутон от тип submit, с което тези бутони се разграничават от други бутони на страницата.

* 1. Стилизиране с flex и grid

За улесняване на стилизирането на контейнерите за елементи и придаването на симетрична структура са ползвани display: flex / grid.

Flex е използва главно за центриране на елементие вертикално и/или хоризонтално или за равномерното разпределяне на пространството между елементи в даден контейнер.

Grid е използван за подреждането на елементите в даден контейнер в точно определен брой колони/редове. Това става с дефиниране на шаблони чрез grid-template-columns и grid-template-row. Размерите на редовете/колоните могат да бъдат дефинирани в шаблоните с комбинация от точни числа и размери, които са определена част от размера на контейнера, върху който е приложен display: grid.

* 1. Асинхронни функции

За изпращане на заявки до сървърните PHP скриптове са използвани асинхронни функции, декларирани чрез ключовата дума async. При изпращане на заявка се създава обект Promise, който е “обещание“, че в някакъв момент на негово място ще бъде получен отговорът от заявката.

Този момент не е ясен и е много вероятно отговорът не бъде върнат преди да стойноста му да бъде нужна. Затова е необходимо чрез ключовата дума await да си осигурим, че отговорът ще бъде изчакан преди да се ползва стойността му. Това обаче може да се случи само в асинхронна функция – правило, наложило се с цел поддържане на ефективна работа на процесора и избягване на прекарване на дълго времме в изчакване на отговор от сървър. Вместо това докато чакат отговор асинхронните функции отстъпват процесорно време за работа на други процеси.

* 1. Изпълняване на код през даден интревал чрез setInterval()

В основата на проекта стоят динамини операции, които включват постоянна промяна на данните. Това налга съдържанието на страницата да бъде периодично презареждано, за да бъде актулно с изменящите се данни. За целта е използван setInterval(), на който се подабав callback функция, която ще се изпълнява през зададен интервал - в случая 5 секунди.

## Използвани технологии

Приложението има три-слойна архитектура:

1. презентационен слой - css/js/html,

2. БД: MySql

3. Бизнес логика – Php

Приложението използва REST архитектурата за връзка между презентационния слой и бизнес логиката. Клиентът комуникира със сървъра чрез HTTP заявки в JSON формат. Ресурсите на сървъра, които комуникират с презентационния слой, са организирани като отделни файлове за обработване на различен обект. Някои от тези ресурси могат да получават повече от един тип HTTP заявка, в зависимост от целта на клиента. Например GET за вземане на ресурс, POST за промяна на ресурс(регистрация, вход) и DELETE за изтриване на ресурс(изход). Всеки от тези специални файлове, наречени ендпоинт-и, извиква методи от класовете с общото название handlers, които предоставят основната бизнес логика на системата и комуникират с базата данни.

Може да обозначим ендпоинтите като level 1 и хендлърите като level 2. Има също и класове за моделиране на обектите от базата данни, както и класове, които са изключения. Всеки файл на сървъра първоначално прави инициализация чрез класа AppBootStrap. В него е логиката за стартиране на сесия. Той дефинира autoload\_register, който добавя класовете, които се използват от даден скрипт, но се намират в други файлове. В него също има и exception handler, който обработва хвърлените изключения според вида им и връща подходящи HTTP response codes, след което терминира програмата.

## Инсталация и настройки

Как да стартираме проекта?

1. Включете xampp или друг Apache и MySql сървър.

2. Създайте база данни

3. Изпълнете sql/meetup.sql и след това sql/example\_data.sql в MySql базата данни.

4. Заменете конфигурацията за базата данни в config/config.ini с конфигурацията на вашия MySql сървър.

5. Отворете приложението в браузър.

## Кратко ръководство на потребителя

* 1. **Горно меню**

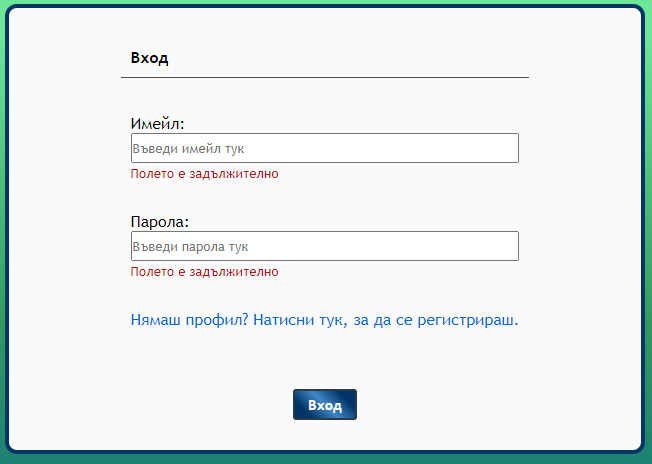


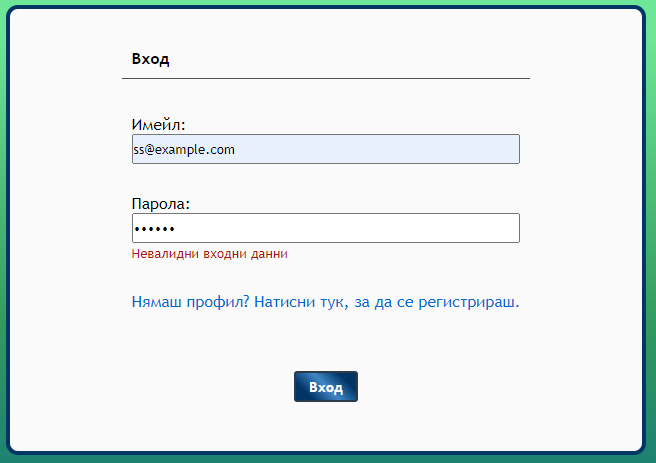
Фигура 1 – горно меню

Горното меню е лентата в грната част на всяка страница. Тя се състои от следните елементи:

* лого – при натискане върху се достъпва началната страница index.html
* линк за регистрация и линк за вход
* икона за достъпване на потребителския панел – ако потребителят не е влязъл в акаунта си, го отвежда към страницата за вход
  1. **Вход**

Страницата за вход може да бъде достъпена от грното меню чрез натискане въху “Вход“. Тя съдържа форма с две полета – за имейл и за парола.

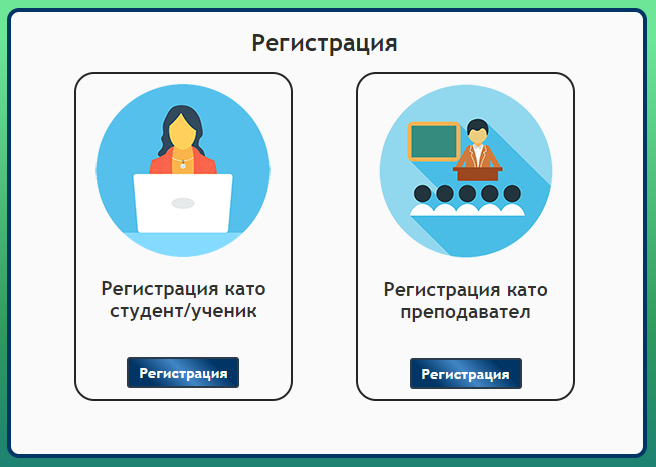
При опит за вход с празни полета потребителят е уведомен чрез съобщенията за грешка представени на фигура 3, а при опит за вход с несъответсващи парола и имейл - чрез съобщенията за грешка представени на фигура 2.



Фигура 3 – съобщения за грешка при празни полета във формата за вход

Фигура 2 – съобщения за грешка при невалидни входни данни във формата за вход

* 1. **Регистрация**

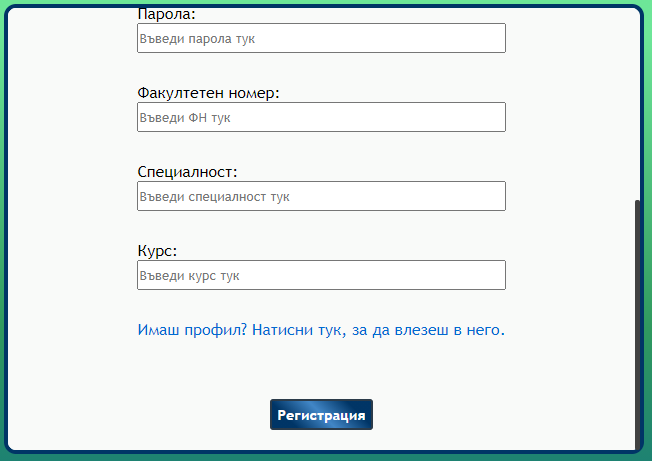
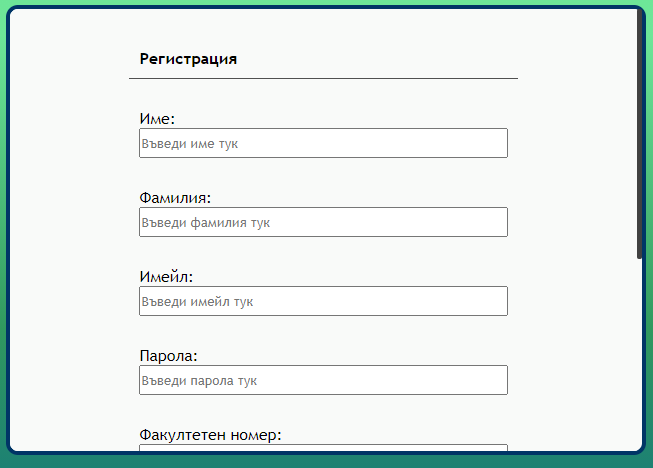
Страницата за регистрация може да бъде достъпена от горното меню чрез натискане върху “Регистрация“. Тя съдържа два панела с бутони за регистрация като студент или като преподавател.

Фигура 4 – страница за избор на вид регистрация

* + 1. Регистрация като студент

При избор на регистрация като студен, потребителят е отведен на страница с форма за регистрация, показана на фигура 5 и 6. Тя съдължа полета за име, фамилия, имейл, парола, факултетен номер, специалност, курс.

Паролата трябва да е поне 6 символа, включващи поне една цефра и поне една буква.

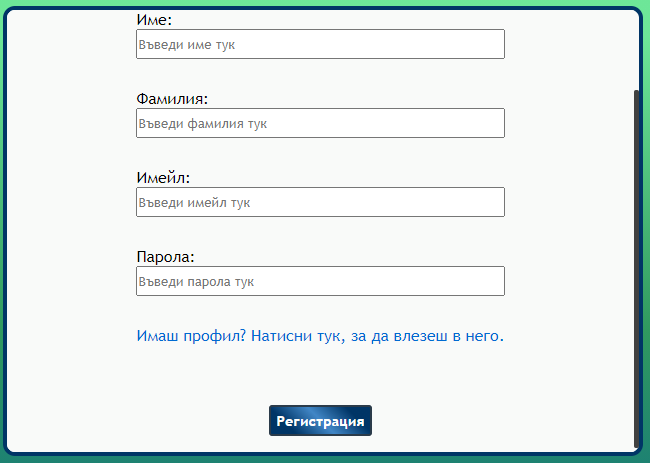


Фигура 5 и 6 – форма за регистрация на потребител

При празни полета или невалидни данни потребителят е уведомен с подобни събщения за грешка, като тези в страницата за вход.

* + 1. Регистрация като преподавател

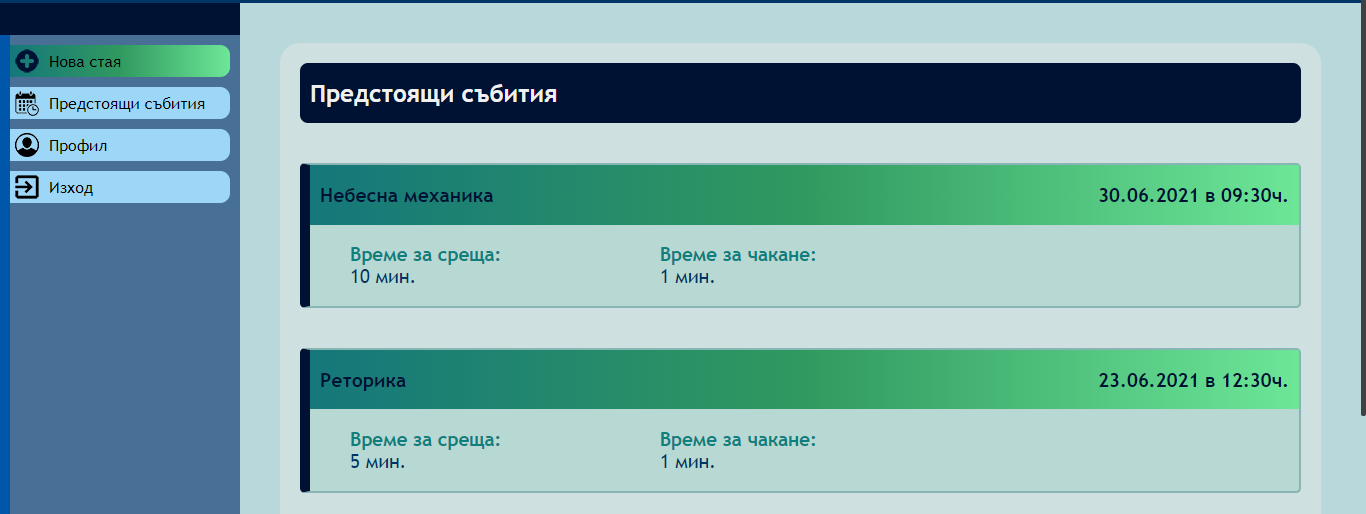
При избор на регистрация като преподаветл, потребителят е отведен на страница с форма за регистрация, показана на фигура 7. Тя съдължа полета за име, фамилия, имейл и парола.



Фигура 7 – форма за регистрация на преподавател

* 1. **Потребителски панел**

Потребителският панел е достъпен след вход от горното меня чрез натискане на иконата най-вдясно. Различните страници от панела могат да бъдат достъпени от страничното навигационно меню, представено на фигура 8.

* + 1. Предстоящи събития

Страницата съдържа списък с всички предстоящи събития.

Фигура 8 – страницата на предстоящите събития и странично навигационно меню

**Студенти:**

Списъкът представя събитията, в чийто график е включен студентът. Ако събитието е активно в момента, чрез натискане върху него студентът е препратен към страницата на чакалнята и е автоматично подреден спрямо мястото си в графика.

**Преподаватели:**

Списъкът представя събитията, създадени от преподавателя. Чрез натискане върху дадено събитие преподавателят е препратен към страницата на чакалнята, от където може да го контролира.

* + 1. Профил

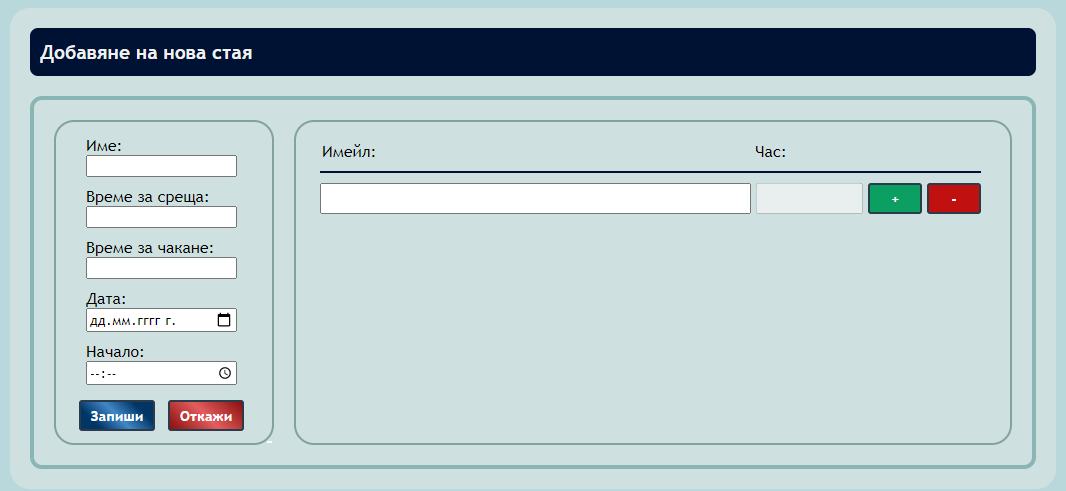
Представя потребителските данни за текущия потребител.

* + 1. Изход

При натискане върху “Изход“ се излиза от текущия профил и потребителят е препратен към началната страница.

* 1. Създаване на стая

Ако текущия потребител е преподавател, може да създаде нова стая от бутона “Нова стая“ от навигационното странично меню.



Фигура 9 – страница за създаване на стая

За създаване на стая е необходимо да се зададе име на стаята, време за среща,време за чакане между срещите, дата и време на начало на събитието. За създаване на графика се използва въвеждане на имейл за всеки студент. Автоматично се изчислява очаквано време за среща за всеки студент. Чрез двата десни бутона може да се добавят и премахват студенти на съответната позиция.

При запис на стаята се прави валидация за празни полета.

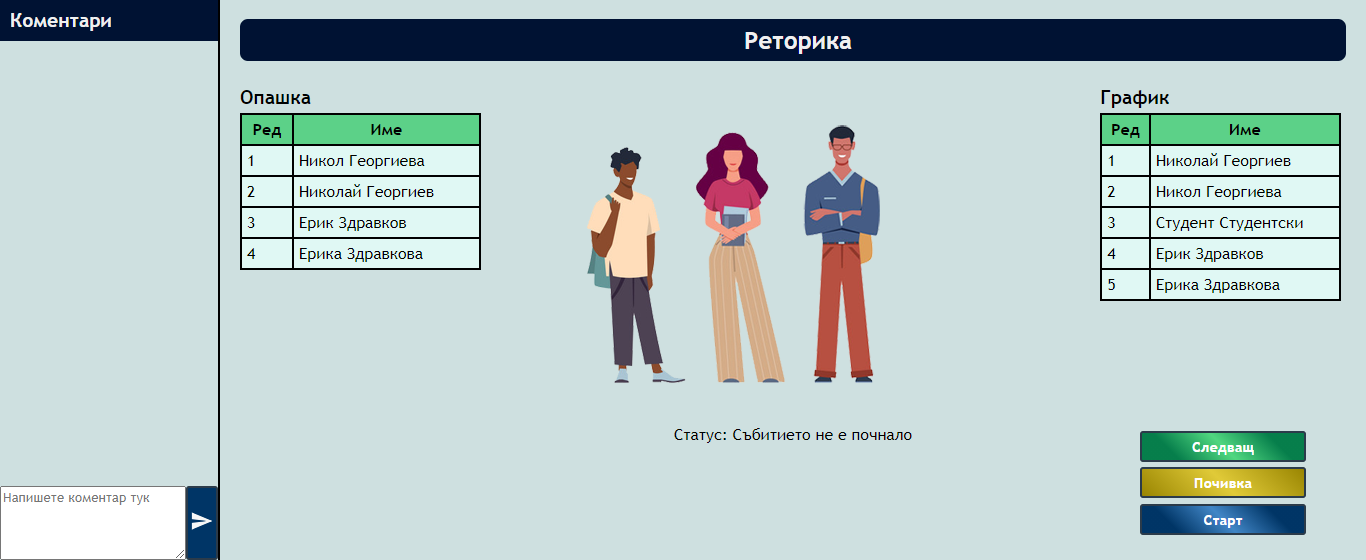
* 1. **Опашка**

Опашката е достъпна чрез натискане върху съответната стая от списъка с предстоящи събития. В дясно е панелът за коментари, където преподаватели и студенти могат да кореспондират.

Полето за статус на опашката може да има една от следните стойности:

* Събитието не е почнало – преди да бъде стартирано от преподавател чрез съответния бутон
* Изчакване – време за изчакване между срещите
* Провежда се среща
* Времето за среща изтече / Времето за чакане изтече – сигнализира, че предварително определеното време за съответната дейност е изминало

**Преподаватели:**



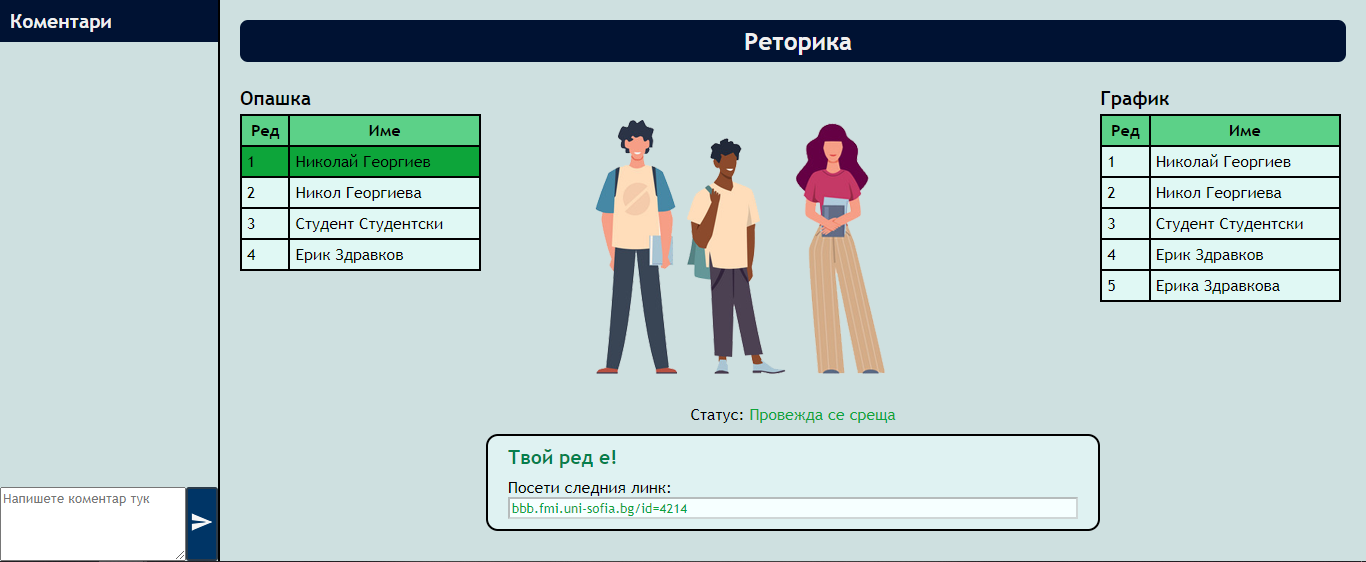
Фигура 10 – страница на чакалнята(опашката) за преподавател

Преподавателите имат допълнителна секция с бутони за контрол на събитието. Чрез тях могат да:

* стартират/спират опашката
* пускат директно следващия студент за среща
* стартират време за почевка/изчакване след край на една среща и преди начало на следващата
* чрез натискане върху реда на даден студент в таблица на опашката могат да извикат директно този студент за среща, независимо от техния ред

**Студенти:**

Когато дойде реда на текущия студент на екрана се появява поле, съдржащо линка за среща.



Фигура 11 – страница на чакалнята(опашката) за студент при получаване на линк

## Примерни данни

Примените данни могат да бъдат намерени във файла example\_data.sql. Те могат да послужат за тестване на проекта.

Паролите на всички потребители са “pass123456“, изключение е Мастър Йода, чиято парола е “123456pass“.

## Описание на програмния код

* 1. **index.html**

Представя структурата на началната страница. За структуриране на информацията са използвани семантичните елементи section и article със съответстващи им header-и, в които се намира подходящ heading за нивото на „важност“ на информацията в дадената секция. Чрез <h1> e представено заглавието на страницата, а чрез <h2> - заглавията на двете подсекции на страницата.

* 1. **index.css**

Представя стила на началната страница. За постигане на ефекта в секциите “Преподаватели“ и “Ученици/студенти“ е използвано свойството float: left/ float: right, приложено към картинките. По този начин те се позиционират съответно отляво или отдясно на текста.

* 1. **main.css**

Съдържа общите стилове за повечето страници в проекта. Като пример за това са стиловете за главния header, стилове за бутоните и за scrollbar-а.

* 1. **login.html**

Страница с форма за въвеждане на входни данни на регистриран потребител. Използвани са семантичните елементи fieldset за обозначаване, че всички елементи принадлежат към една и съща група – тази на входните дании, и legend за обзначаване на наименованието на групата. На всеки input съответства label елемент, описващ очакваната информация.

За изпращане на формата се ползва бутон от тип submit.

* 1. **login.css**

Представя стила на страниците за вход, за избор на тип регистрация и двете страници с форми за съответния тип регистрация.

Причината за използване на един файл е, че тези страници използват почти едни и същи стилове за елементите си.

* 1. **login.js**

Съдържа логиката, използвана за споменатите в миналата подточка страници.

И тук се ползва един файл, защото логиката за отделните страниците има много общи части. Тя включва следните функции:

* + 1. init()

Извиква се при отваряне на страницата и изпълнява всичко необходимо за зареждане на динамичното съдържание.

В случая това е добавянето на EventListener-и към дадени елементи, чрез което се следи за настъпването на дадено събитие зададено като първи аргумент на EventListener-а и при неговот настъпване се изпълнява функцията, подадена като втори аргумент.

Следим за следните събития:

* + - натискане въру бутон за регистрация на ученик или бутон за регистрация на преподавател в страницата register.html, при което се извиква функция redirect() с аргумент съответната страница за регистрация на даден тип потребител
    - изпращане (submit) на форма за вход, при което се проверява за валидни входни данни чрез validateInput() и validateEmptyFields(). Ако са валидни се изпълнява postForm(), а ако не са – се предотвратява изпращането на формата чрез event.preventDefault(), където event е събитието, прихванато от EventListener-а. Submit събитието представлява натискане на submit бутона в съответната форма.
    - изпращане (submit) на форма за регистрация на ученик или преподавател – по аналогичен начин на горната точка
    1. async postForm (formType, userTypeId)

Асинхронна функция, която извършва операциите, свързани с изпращане на заявка с данни от форамта и с обработката на отговора от тази заявка.

Аргумента formType може да е “login” или“register” и представя вида на формата, коята ще бъде изратена. Аргумента userTypeId може да е “1”(ученик) или “2”(преподавател) и представя вида потребител.

Изпраща POST заявка към съответния адрес с обработените от getFormDataJSON() данни от формата. Ако отговорът на заявката не връща, че е успешна(т.е. няма обектът отговор няма поле success със стойност true), след контейнера на писледното поле от формата се добява съобщеине за невалидни входни данни.

* + 1. validateEmptyFields()

Взима всички input елементи и ги обхожда, като проверява дали полетата са празни и добява съобщеине за грешка след всяко такова празно поле чрез функцията addError(). Ако вече има добавено съобщение за грешка и полето вече не е празно премахва елемента за съобщението.

* + 1. addError(input, errorMessage)

Създава и добавя small таг, чрез който се представя грешката errorMessage за невалидно входно поле input.

* + 1. validateInput()

Проверавя дали входните данни от формата са валидни и добява съответните съобщения за грешка за невалидните полета.

Имейлът и паролата се валидират съответно чрез isValidEmail() и isValidPassword() и по аналогичен на начина за грешки при празни полета се добавят грешки за невалиден имейл.

При всяка открита грешка на флага isValid се задава стойност false. Накрая функцията връща неговата стойност.

* + 1. isValidEmail(email)

Приема символен низ с подадения във формата имейл и тества дали отговаря на валидиращия регулярен израз.

* + 1. isValidPassword(password)

Приема символен низ с подадената във формата парола и тества дали отговаря на валидиращия регулярен израз.

* + 1. getFormDataJSON (userTypeId)

Взима стойностите на input елемнтите от формата и ги добява към обект, който накрая се трансформира в JSON стринг и се връща като резултат от функцията.

* + 1. redirect (page)

Извършва front-end препращане към подадената като параметър страница.

* 1. **register.html**

Страница за избор на тип потребител за регистрация – ученик/студент или преподавател.

* 1. **register\_student.html**

Страница, съдържаща форма с полета за попълване на необходимите данни за регистрация на студент/ученик. Това включва input елементи за име, фамилия, имейл, парола, факултетен номер, специалност и курс.

* 1. **register\_teacher.html**

Страница, , съдържаща форма с полета за попълване на необходимите данни за регистрация на преподавател. Това включва input елементи за име, фамилия, имейл и парола.

* 1. **register\_success.html**

Страница, към която потребителят е препратен след успешна регистрация. Съдържа само съобщение за успешна регистрация и връзка към началната страница.

* 1. **panel.html**

Динамична старница, която според вида на текущия влязъл потребител съдържа събититята, за които ученик/студент е включен или събитията, които преподавател е създал. Тези събития се добавят от panel.js в празната скеция с id=”events”.

* 1. **panel.css**

Съдържа стиловете, използвани в panel.html, profile.html, add\_room.html, т.е. общите стилове за страниците от потребителския панел.

* 1. **panel\_init.js**

Съдържа общата логиката за инициализиране, необходима за динамичнитите страници. Целта на този файл е да обедини операциите, които се изпълняват при зареждане на повечето страници.

Ако те бяха разделени в останалите скриптове, би имало следния проблем - някои страници добавят повече от един скриптов файл с функции, които трябва да се изпълнят при зареждане на страницата, което довежда до изпълнението само на една от тях.

* + 1. window.onload = async function()

Анонимна асинхронна функция, която се изпълнява при зареждане на страницата, в която е добавен скрипта. Инициализира динамичното съдържание в сътоветните страници. Спрямо наличието на дадени елементи на страницата, извиква фукциите initNav(), loadEvents(),loadUserInfo(),initQueuePage().

* + 1. initNav(userType)

Инициализира динамичното съдържание за навигационното странично меню. Следи за натискане на отделните елементи в менюто и препаща към съответната страница за елеменитет “Събития“ и “Профил“ или извиква logout() при натискане на “Изход“.

Ако потребителят е преподавател инициализира и бутон за създаване на нова стая чрез addCreateEventButton().

* + 1. async initQueuePage(userType)

Инициализира динамичното съдържание за страницата на чакалнята(опашката). Взима от адреса на страницата идентификационния номер на страницата, за която се отнася опашката.

Тук се извикват функциите от queue.js за зареждане на динамичното съдържание на страницата queue.html. То включва главно бутоните за контрол на опашщата от преподавателя и бутона за добавяне на коментари. Също така зарежда таблицата на хората в опашката и на хората в графика, както и за състоянието на опашката.

Чрез window.setInterval() през 5 секунди се презареждат таблицата за опашката, състоянието на опашката, коментарите и се проверява дали е дошъл ред на текущия потребител, ако той е студент, при което получава линк.

* + 1. async getUserType()

Изпраща заявка към session.php за получаване на данни за текущия потребител, спрямо което се определя неговия тип – 1 за студент и 2 за преподавател.

* + 1. addCreateEventButton ()

Добавя бутон за създаване на нова стая към страничното навигационно меню, ако текущия потребител е преподавател.

* + 1. async logout()

Изпраща заявка към session.php, чрез която премахва потребителските данни от сесията и излиза от профила на текущия потребител.

* 1. **panel.js**

Съдържа логиката, използвана в panel.html и profile.html, както и за навигацията в старничното меню.

Съдържа следните функции:

* + 1. async loadEvents(userType)

Асинхронна функция, която добавя информацията за събитията в panel.html. Информацията за събитията се взима чрез GET заявка към room.php. Тя се подава като аргумент на getEventHTML() и полученият като резултат символен низ с html елементи се добвая към контейнера за събитията в panel.html.

За всеки елемент, съдържащ информация за събитие добавяме EvenListener, който следи за събитието натискане върху елемент, при което препраща към опашката на стаята (queue.html), като добавя параметър roomId към URL адреса. Стойността на параметъра се взима от името на идентификатора на елемента с информацията за събититето.

Ако няма събития за дадения потребител добавя подходящо съобщение.

* + 1. getEventHTML(userType, event)

Създава и връща символен низ, в който данните за събитие(параметъра event) са добавени като текстово съдържание в html елементи. Символният низ може да е един от два вида спрямо типа на потребителя.

За студенти (userType = 1) връща стаите, в които са включени и информация за име на стая, място на студента в графика, ориентировачен час и преподавател. HTML съдържанието получава чрез функцията getStudentEvent().

За преподаватели(userType = 2) връща саздадените от тях стаи с информация за име на стая, време за среща, време за чакане и дата и час на начало. HTML съдържанието получава чрез функцията getTeacherEvent ().

* + 1. async loadUserInfo(userType)

Асинхронна функция, която довабя потребителските данни в profile.html. Данните се взимат чрез GET заявка към session.php и ги добява като текстово съдържание в html елемнти от getUserHTML().

* + 1. getUserHTML(userType, userData)

Приема като параметри тип на потребител(userType) и обект(асоциативен масив), представящ данните, върнати от session.php.

Според вида на потребителя връща един от два варианта на html съдържанието, необходимо за представяне на съответната информация в profile.html. То съдържа име и имейл за преподавател и име, имейл, факултетен номер, курс и специалност за студент.

* 1. **profile.html**

Динамична страница, съдържаща данни за потребителския профил.Тези данни се добавят от panel.js в празната секция с id=”profile”.

* 1. **profile.css**

Съдържа стиловет за елементите в страницата на потребителския профил profile.html.

* 1. **add\_room.html**

Динамична страница, видима само за преподавателите, съдържаща форма за добавяне на нова стая.

Формата съдържа две главни групи от полета, което е предсатевно чрез групирането им в два fieldset елемента.

Първата група (fieldset id=“information”) съдържа input елементи за име на стая, време за среща, време за чакане, дата и час на започване на събитието. Втората група (fieldset id="scheduled\_users") съдържа input елемент за въвеждане на имейл на студент и елемент, показващ ориентировачно време за представяне на всеки студент спрямо информацията зададена в съответните полета от другата група. Нови студенти в графика се добавят или премахват със съответния бутон на желания ред.

* 1. **add\_room.css**

Съдържа стиловет за елементите в страницата add\_room.html.

* 1. **add\_room.js**

Съдържа логиката, използвана за операциите, свързани с добавянето на нова стая в add\_room.html.

Съдържа следните функции:

* + 1. init()

Инициализира динамичното съдържание на страницата, включващо добяване на полетата за въвеждане на информация за събитието и за студентите.

Добавя EventListener-и, които автоматично презареждат часовете в графика при промяна на данните в някое от полетата за информация за събититето.

* + 1. aync postForm()

Асинхронна функция, изпращаща заявка към room.php, чрез която добавя новото събитие към събитията за дадения преподавател и за студентите, включени в графика.

* + 1. buttonsInit(addButton, removeButton)

Инициализира бутоните за запис, отказ и за добавяне и премахване на студент към графика.

* + 1. getFormData()

Събира входните данните от всички полета на страницата и ги обединява в асоциативен масив, който в накрая се превръща в JSON обект.

* + 1. validateFormInput()

Извиква функцията renderError() за отделните полета за входна информация за събитието, чрез което ги валидира и добява обозначения за невалидни полета, където има такива.

* + 1. validateScheduleInput()

Извиква функцията validateInput () за отделните полета за входна информация за графика, чрез което проверява дали всички полета за имейли на студенти са попълнени.

* + 1. validateInput(element)

Проверява дали полето за входни данни, подадено като параметъра element, е празно.

* + 1. renderError(element)

Функция, която, ако полето за входни данни element е празно, добавя към него стилът за обозначаване на празни полета или го премахва, ако полето вече не е празно.

* + 1. changeTime(timeString)

Изчислява и добавя към съотетния ред времето на започване на всеки от студентите в графика на стаята в add\_room.html. За целта се създава Date обект от параметъра timeString и се извличат стойностите за време за среща и време за чакане, въведени в съответните input елементи.

Всики елементи с време на започване на среща със студент(class=”userTime”) се обхождат като ня всяка итерация на полето се задава стойността на Date обекта, след което към него се добява врмеето за среща и времето за чакане. За целта се ползва addMinutes().

* + 1. addMinutes(date, minutes)

Връща Date обект, чието време се определя като се сумират подадените като аргумент минути (minutes) с времето на подадения като аргумент Date обект(date).

* 1. **queue.html**

Съдържа структурата на страницата за чакалнята(опашката). Това включва таблиците за опашката и графика, поле за анимацията, панел за коментари.

* 1. **queue.css**

Съдържа стиловете за елементите и анимациите в queue.html.

* 1. **queue.js**

Съдържа логикат за динамичното съдържание на страницата queue.html. Тук е главната функционалност на проекта.

* + 1. async renderQueueTable(roomId, userType)

Асинхронна функция, която изпраща заявка към queue.php за получаване на данни за студентите в опашката, след което добява съответната информация в html таблицата на опашката.

* + 1. async renderScheduleTable(roomId)

Асинхронна функция, която изпраща заявка към schedule.php за получаване на данни за студентите в графика, след което добява съответната информация в html таблицата на графика.

* + 1. async startEvent(roomId)

Асинхронна функция, чрез която преподавател изпраща заявка към queue.php за стартиране на текущото събитие.

* + 1. async refreshQueueStatus(roomId, userType)

Асинхронна функция, която получава даните за състоянието на опашката от getQueueStatus() и представя статуса в полето “Статус“.

* + 1. async getQueueStatus(roomId)

Асинхронна функция, която изпраща заявка към get-queue-state.php за получаване на текущия статус на опашката.

* + 1. async getLink(userType, roomId)

Асинхронна функция, която изпраща заявка към students.php, ако текущия потребител е студент. Ако е ред на студента, са добява поле с линка, получен от заявката. При приключване на срещата студнта бива препратен обратно към страницата със събития.

* + 1. async finishCurrentMeeting(roomId)

Асинхронна функция, чрез която преподавател изпраща заявка към pause-queue.php за приключване на текущата среща.

* + 1. async startNextMeeting(roomId)

Асинхронна функция, чрез която преподавател изпраща заявка към process-queue.php за стартира нова среща със следващия в опашката студент.

* + 1. async enterQueue(roomId)

Асинхронна функция, чрез която студент изпраща заявка към process- students.php за нареждане в опашката. Функцията се изпраща при отваряне на страницата чрез извикване в panel\_init.js.

* + 1. async startQueue(roomId)

Асинхронна функция, чрез която преподавател изпраща заявка към queue.php за стартиране на опашката.

* + 1. async endQueue(roomId)

Асинхронна функция, чрез която преподавател изпраща заявка към queue.php за спиране на опашката. За целта се праща параметър active=0 в адреса.

* + 1. async selectNextMeeting(roomId, studentId)

Асинхронна функция, чрез която преподавател изпраща заявка към process-queue.php за избиране на конкретен студент независимо от реда му, с който да бъде проведена следващата среща.

* + 1. initTeacherButtons()

Инициализира бутоните за контрол на събитието, които включват бутон за следваща среща, бутон за почивка и бутон за стартиране/приключване на събитието.

* + 1. async initActivationButton(roomId)

Инициализира бутона за стартиране/приключване на събитието. За целта проверява дали събитието е в процес на действие или не чрез getQueueStatus().

* + 1. async renderActivationButton(roomId)

Извършва промяната на стила на бутон за стартиране/приключване на събитие при прехода между двете състояния.

* + 1. async getRoomData(roomId, userType)

Асинхронна функция, чрез която се изпраща заявка към room.php за получаване на данни за събитията на текущия потребител. От тах се търсят данните за текущото събитие и биват върнати като изходна информация.

* + 1. showLink(link)

Добавя полето за линк със символния низ, подаден като аргумента link.

* + 1. nextAnimation()

Функция за изпълняване на анимацията с хора минаващи на опашка.

* + 1. async addComment(roomId)

Асинхронна функция, чрез която се изпраща заявка към comments.php за добавяне на коментар към базата данни.

* + 1. async renderComments(roomId)

Асинхронна функция, чрез която се изпраща заявка към comments.php за получаване на всички коментари за дадената стая. За получените данни се създават елементи, които се добавят към панела за коментари в queue.html.

* + 1. getCommentHTML(commentData)

Функция, връщаща html елемента за коментар с данните от commentData.

В директорията src се разполагат .php файловете, които имплементират функционалността и ще комуникират с базата данни.

* 1. **AppBootStrap.php**

Съдържа класа AppBootStrap, който се използва за инициализация във всички .php файлове.

* + 1. init()

Използва spl\_autoload\_register за да добавя всички класове, които се използват от скриптовете, но са дефинирани в други .php файлове. Използва set\_exception\_handler за да обработва всички хвърлени изключения според вида им и връща съответните http response codes и response. След изпълнението на set\_exception\_handler следва терминиране на програмата.

* 1. **DB.php**

Съдържа класа DB, който създава връзката с базата от данни, четейки необходимите данни от конфигурационен файл config.ini, който се намира в директорията config.

* + 1. getConnection()

Връща обект от класа **PDO**, който е осъществил връзка с базата от данни.

* 1. **User.php**

Съдържа класа User, който моделира таблицата **users** от базата данни. Има атрибути за име, имейл и id, заедно със съответните getter-и. Имплементира интерфейса JsonSerializable и предефинира метода jsonSeriazle, който ще се извика при извикване на json\_encode с обект от класа и ще представи обекта като асоциативен масив.

* 1. **Student.php**

Съдържа класа Student, който **наследява** класа User. Той ще моделира потребителите, които са студенти. Добавени са атрибути за факултетен номер, курс и специалност и съответните getter-и. Имплементира интерфейса JsonSerializable.

* 1. **Room.php**

Съдържа класа Room, който моделира таблицата **rooms** от базата данни.

Има атрибути за id, име, време за чакане(почивка) между срещите, време на продължителност на срещите и начало на срешите. Дефинирани са и съответните getter-и. Имплементира интерфейса JsonSerializable.

* 1. **Comment.php**

Съдържа класа Comment, който моделира таблицата **comments** от базата данни. Има атрибути за съдържание и дата на създаване, заедно с getter-и. Имплементира интерфеса JsonSerializable.

* 1. **UserRequestHandler.php**

Сърдържа класа UserRequestHandler, който имплементира функционалности, свързани по-тясно с потребителите.

* + 1. createUser(userInfo)

Създава потребител и го записва в базата данни. Като параметър приема

Асоциативен масив с информация за потребителя. След като добави потребителя в таблицата users, проверява неговия атрибут userTypeId за да удостовери какъв тип потребител трябва да бъда създаден. Проверката се осъществява като се сравни userTypeId със записите в таблицата userTypes. Записите в userTypes са за студент и преподавател. Ако се установи, че userTypeId съвпада с id-то на студентския тип в таблицата, то се изпълняват допълнителни операции. Тогава в таблицата student\_details се добавят уникалните за всеки студент данни, като факултетен номер, година и специалност. Редовете в двете таблици, отнасящи се до един потребител се свързват с външен ключ в таблицата students\_details, който сочи към запис в таблицата users. Ако методът е завършил успешно се връща id на новосъздадения потребител.

* + 1. getUserById(id)

Връща информация за потребител по подадено id. Първоначално проверява дали съществува потребител със съответното id в таблицата users. След това сравнява userTypeId със записите от таблицата userTypes. Ако се окаже, че потребителят е студент, взема допълнителна информация от таблицата students\_details. След това връща обект Student или User, в зависимост дали потребителят е студент или преподавател.

* 1. **SessionRequestHandler.php**

Съдържа класа SessionRequestHandler, който имплементира функционалности, които са тясно свързани със сесията на всеки потребител.

* + 1. Login(loginData)

Осъществява влизането на потребителя в профила му и автентикирането му. Като параметър очаква асоциативен масив с имейл и парола. След това търси в таблицата users дали има потребител с такъв имейл и парола. Ако няма такъв потребител или паролата е грешна се хвърла BadRequestExcepton. Иначе се връща обект от тип User с данните на потребителя.

* + 1. requireLoggedUser()

Проверява дали потребителят е влязъл в профила си. Това става чрез проверка на масива Session. Ако не е влязъл, то се хвърля AuthorizationExcepton.

* + 1. requreLoggedTeacher()

Проверява дали потребителят е влязъл в профила си и дали той е учител.

Това става чрез проверка на масива Session, като се проверява дали ключът typeId(id-то на ролята на потребителя) съответства на id-то на ролята учител. Ако потребителят не е влязъл в профила си или е студент, то се хвърля AuthorizationException.

* 1. **RoomRequestHandler.php**

Съдържа класа RoomRequestHandler, който имплементира фунционалности, които са свързани със стаите.

* + 1. createRoom(roomData)

Създава стаите. Приема като параметър асоциативен масив с информация за стаята. Добавя в таблицата rooms запис за новата стая, съдържащ информацията за стаята. В асоциативният масив roomData се съдържа ключ schedule, в който има масив, в който са записани имейлите на студентите в реда, в който преподавателят ги е наредил за влизане първоначално. Този масив от имейли се обхожда и след това се търсят в таблицата users потребителите, които отговарят на тези имейли, за да се намери информация за тях. След това за всеки студент се добавя в таблицата schedule неговото id(външен ключ), id на стаята(външен ключ) и позицията му в разписанието. Ако всичко е преминало успешно, то се връща id на новата стая.

* + 1. getUserRooms(id)

Връща стаите, в които участва влезлия потребител. Първоначално се проверява дали потребителят е студент или преподавател чрез сравняване на стойността typeId в Session масива със записите в таблицата userTypes. Ако потребителят е преподавател, то се връща обект от клас Room. Ако потребителят е студент, то е необходима информация за реда на студента и преподавателя. Тази информация се взема като се направи JOIN между редовете на таблицата schedule, в които участва студента и rooms въз основа, че запис в schedule отговаря за стая със запис в rooms. След това се връща информация за стаята, информация за преподавателя и реда на студента в разписанието.

* 1. **ScheduleRequestHandler.php**

Съдържа класа ScheduleRequestHandler, който имплементира функционалности, свързани с разписанието.

* + 1. getSchedule(roomId)

Връща разписанието за конкретна стая по подадено Id на стая. Прави JOIN между записите в таблицата schedule, които отговарят за съответната стая, с таблицата users въз основа на това, че запис в schedule отговаря за един студент. Това се прави с цел да се вземе името на студента и неговото id. След това от резултатите от JOIN се взема необходимата информация за студентите и се връща масив със студентите в реда, посочен в разписанието.

* 1. **StudentsRequestHandler.php**

Съдържа класа StudentsRequestHandler, който имплементира функционалностите, които са дефинирани само за студенти. .

* + 1. addToQueue(userId,roomId)

Влезлият студент се добавя в опашката за чакане на някоя стая. Приема два параметъра - id на студента и id на стаята. Първо методът търси реда на студента в разписанието на стаята. Ако студентът не принадлежи на стаята, се хвърля AuthorizationException. След това студентът се добавя в таблицата queues, съответно с неговото id, реда му в разписанието и id на стаята.

* + 1. getLink(roomId, userId)

Влезлият студент получва линк ако е негов ред. Приемат се два параметъра – id на студента и id на стаята. Прави се заявка към таблицата queues с id-то на стаята и id-то на студента. От отговора на заявката проверяваме дали редът в таблицата queues има стойност active равна на 1. В такъв случай, то влезлият студент е на ред и му се връща линк от BBB. В противен случай се връща празен стринг.

* 1. **QueueRequestHandler.php**

Съдържа класа QueueRequestHandler, който имплементира функционалностите, свързани с опашките.

* + 1. startQueue(roomId)

Стартира опашката за дадена стая. Приема параметър id на стаята. Прави заявка за update към таблицата rooms за записа за съответната стая. Актуализира стойността active за стаята да е равна на 1 и currentTime да е равно на текущото време.

* + 1. getStudentInQueue(roomId)

Връща всички студенти на опашката за подадена стая. Приема параметър id на стаята. Прави JOIN заявка между записите в queues за съответната стая и users, въз основа на това, че запис в queues съответства на един студент. Това се прави за да се вземе името на студентите. Връщаме id и име на студентите, както и id на студента, който е на ред в опашката. Студент е на ред, ако в таблицата queues записа за него има стойност за active 1.

* + 1. refreshQueue(roomId)

Връща състоянието на опашката за дадена стая. Приема параметър id на стаята. По подаденото Id прави заявка към таблицата rooms и взема записа за стаята.

1 случай – стойността на activated за стаята е 1:

Ако стойността на isInMeeting за записа има стойност 1, то в момента има студент, който е на ред. Проверяваме дали стойността currentTime(в случая времето в което е започнало срещата) на записа заедно с времето за среща за стаята са след текущото време. Ако това е вярно, то ще се върне отговор, че времето е изтекло. В противен случай ще се върне отговор, че се състои среща. Ако стойността на isInMeeting има стойност 0, то в момента няма никой на ред. Проверяваме дали стойността currentTime(в случая времето в което е приключило съответната среща) на записа заедно с времето за почива са след текущото време. Ако това е вярно, то ще се върне отговор, че почивката е приключила В противен случай ще върне отговор, че тече почивка.

2 случай – стойността на activated за стаята е 0:

Това означава, че стаята е затворена и връща подходящ отговор в json форамт.

* + 1. getNext(roomId)

Придвижва опашката. Взема следващият на ред студент и отбелязва, че той в момента е в среща. Приема параметър id на стаята. Първо проверява дали има записан ключ studentId в масива GET. Ако има, то той ще бъде взет. Ако няма, то ще бъде избран студента в опашката, който е най-напред в разписанието. Това се случва чрез заявка към таблицата queues. След като вече е избран студента то се прави нова заявка към queues. За записа на избрания студент се променя стойността active да е равна на 1, което ще означава, че в момента той е в среща и трябва да получи линк. След това се прави заявка към таблицата rooms за да се обнови currentTime с настоящото време и isInMeeting със стойността 1. В този случай currentTime обозначава врмето на началото на срещата, а isInMeeting показва, че има студент в среща.

* + 1. finishStudent(roomId)

Завършва срещата с даден студент. Приема параметър id на стаята. Изтрива студента от таблицата queues за съответната стая. След това се прави заявка към таблицата rooms за да се обнови currentTime с настоящото време и isInMeeting със стойността 0. В този случай currentTime обозначава края на времето на последната среща, а isInMeeting показва, че няма студент в среща.

В директорията exceptions се помещават изключенията.

* 1. **BadRequestException.php**

Съдържа класа BadRequestException, който наследява класа Exception. Това изключение се хвърля, когато сме подали някакви грешни данни.

* 1. **NotFoundException.php**

Съдържа класа NotFoundException, който наследява класа Exception. Това изключение се хвърля, когато даден ресурс не е намерен.

* 1. **AuthorizationException.php**

Съдържа клас AuthorizationException, който наследява класа Exception. Това изключение с хвърля, когато потребител няма права да достъпи даден ресурс.

В директорията endpoints се помещават .php файловете, към които ще се изпращат http заявките. Те от своя страна ще ги обработват като използват класовете от директория src и ще връщат подходящ отговор.

* 1. **user.php**

Endpoint, който обработва заявките за обекта User.

* + 1. GET

При GET заявка се изисква да бъде подаден като query параметър id. По подаденото id връща информация за потребителя. Извиква методът getUserById на UserRequestHandler, който връща изискваната информация. Ако всичко е наред информацията се връща във вид на json.

* + 1. POST

Извършва регистрация на потребител. Заявката трябва да съдържа информацията за новия потребител. Извиква метода createUser на UserRequestHandler. Ако всичко е наред, връща отговор в json формат, че заявката е успешна.

* 1. **session.php**

Endpoint, който обработва заявките свързани със сесията.

* + 1. GET

Връща информация за логнатия потребител. Извиква метода getUserById на UserRequestHandler. Върната стойност от метода изпраща обратно в json формат.

* + 1. POST

Логва потребител. Заявката трябва да съдържа имейл и парола на потребител. След това се извиква метода login на SessionRequestHandler . Ако всичко е успешно в масива SESSION се записва, че потребителят е логнат. Освен това се записва id и typeId потребителя. Накрая се връща отговор в json формат, че операцията е успешна.

* + 1. DELETE

Чрез тази заявка потребителят излиза от профила си. Извиква се функцията session\_destroy. Ако всичко е успешно се връща отговор в json формат, че заявката е успешна.

* 1. **room.php**

Endpoint, който обработва заявките, свързани със стаите.

* + 1. GET

Връща всички стаи на логнатия потребител. Извиква метода getUserRooms на класа RoomRequestHandler, който връща данните за стаите. После стаите се връщат като отговор в json формат.

* + 1. POST

Създава нова стая. Заявката трябва да съдържа информацията за стаята. След това се извиква метода createRoom на RoomRequestHandler. При успех той връща id на новата стая. Ако всичко е наред, то ще се върне отговор в json формат, че заявката е успешна.

* 1. **schedule.php**

Endpoint, който обработва заявките, свързани с разписанието.

8.35.2. GET

Връща разписанието за конкретна стая. В масива GET трябва да има посочен roomId. Извиква метода getSchedule на ScheduleRequestHandler, като му подава roomId от GET масива. Методът връща разписанието на стаята, което се връща като отговор в json формат.

* 1. **queue.php**

Endpoint, който обработва заявките, свързани с опашките.

* + 1. GET

Връща студентите в опашката за дадена стая. В масива GET трябва да има посочен roomId. Извиква метода getStudentsInQueue на класа QueueRequestHandler, като му подава roomId от GET масива. Върнатата му стойност се изпраща като отговор в json формат.

* + 1. POST

Стартира дадена опашка. В заявката трябва да се съдържа id на стаята. Извиква метода startQueue на класа QueueRequestHandler, като му подава id-то от заявката. Връща отговор за успешна операция в json формат.

* 1. **students.php**

Endpoint, който обработва заявките, свързани със студентите.

* + 1. GET

Проверява дали влезналият студент е на ред в опашката. Извиква метода getLink на StudentsRequestHandler. В зависимост дали студентът е на ред, ще се върне подходящата върната стойност на метода в json формат.

* + 1. POST

Вкарва влезналият в профила си студент в опашката. В заявката трябва да се съдържа id на стаята. След това се извиква метода addToQueue на StudentsRequestHandler с параметри id-то от заявката и id-то на потребителя от масива SESSION. Ако влизането е успешно и не се е хвърлило изключение то се връща отговор за успешна операция в json формат.

* 1. **comments.php**

Endpoint, който обработва заявките за коментарите.

* + 1. GET

Взема коментарите за определена стая. В масива GET трябва да има roomId. След това се извиква метода getComments на CommentRequestHandler. Той връща коментарите за дадена стая, след което те се изпращат като отговор в json формат.

* + 1. POST

Публикува коментар. Заявката трябва да съдържа съдържанието на коментара и id на стаята, на която принадлежи. Извиква метода createComment на CommentRequestHandler с параметри информацията от заявката и id-то на съответния потребите от масива SESSION. Ако операцията е успешна, то ще се върне отговор в json формат.

* 1. **getQueueState**

Endpoint за проверка на състоянието на опашка.

* + 1. GET

Взема текущото състояние на опашка. Като query параметри трябва да е подадено id на стаята. Извиква метода refreshQueue на QueueRequestHandler с параметър Id-то на стаята. Върнатата стойност от метода се изпраща като отговор в json формат.

* 1. **pauseQueue**

Endpoint за спиране на срещата с текущия студент в опашката за дадена стая.

* + 1. GET

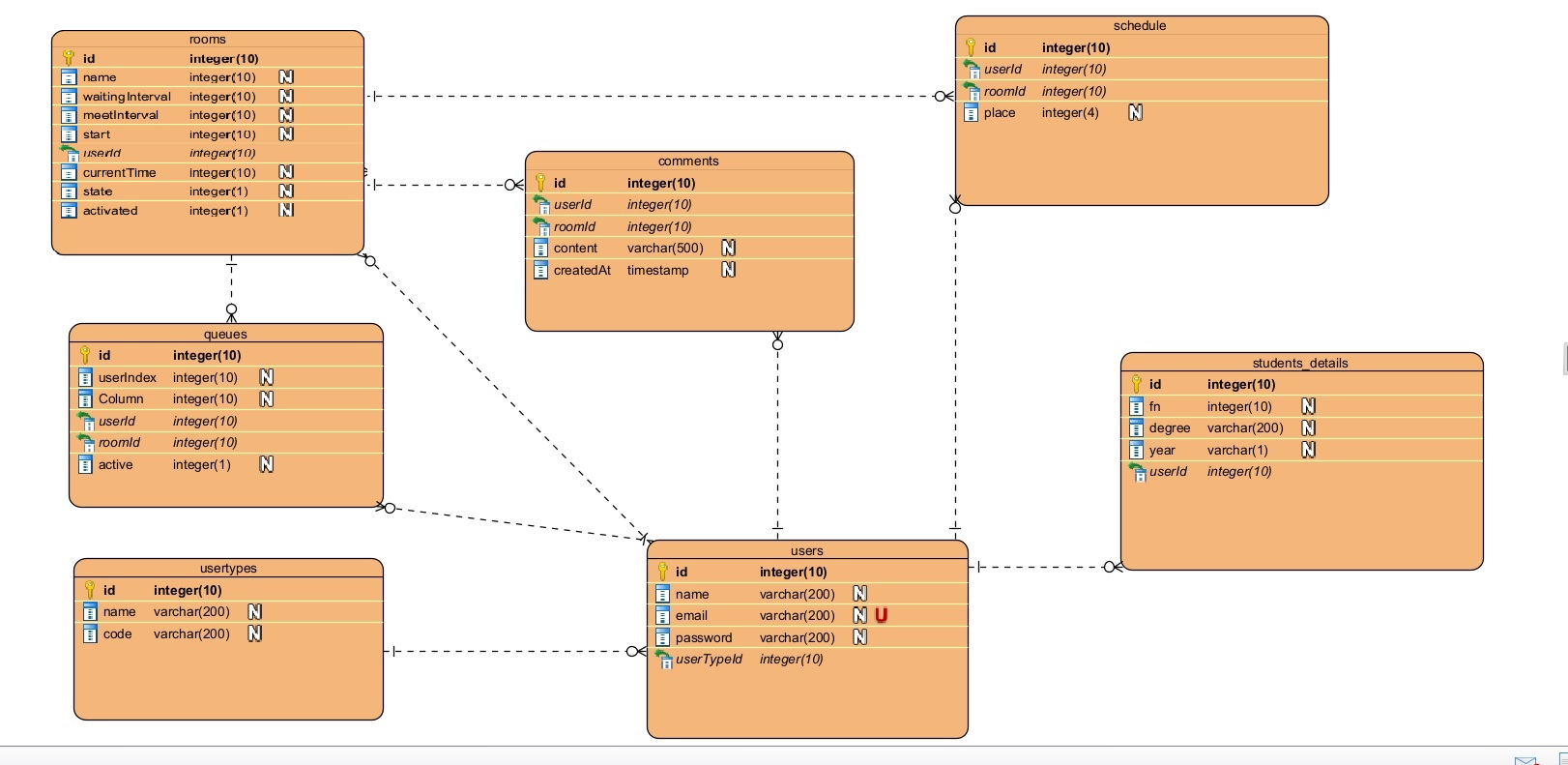
Завършва текущата среща в опашката за подадена стая. Като query параметри в GET масива трябва да е подадено Id на стаята. Извиква метода finishStudent на QueueRequestHandler с параметър подаденото Id. Ако операцията е успешна се връща отговор в json формат.

* 1. **processQueue**

Endpoint за вземане на следващ потребител за среща от опашката.

* + 1. GET

Взема потребител от опашката и го прави активен. Като query параметри в GET масива трябва да е подадено Id на стаята. Извиква метода getNext на QueueRequestHandler с параметър подаденото Id на стаята. Ако операията е успешан се връща отговор в json формат.



Фигура 12 – графика на базата данни

## Приноси на студента, ограничения и възможности за бъдещо разширение

* 1. **Приноси на студентите**

**Николай Георгиев:**

Работил по повечето frontend логика и стиловете на проекта, началната страница, frontend на страницата за опашката, свързване на frontend с backend и тестване на свързването, backend за коментарите и backend за графика.

**Ерик Здравков:**

Работил по повечето backend логика на проекта, упрвалението на грешките, свързване на базата данни със сървърните скриптове, panel.js и по frontend частта на коментарите в страницата на опашката,

* 1. **Възможности за бъдещо развитие**

В следващи версии на проекта може да бъдат добавени следните подобрения и функционалности:

* редакция на създадените стаи
* нареждане в опашка в стая без използване на график
* генериране на реални линкове, които да са различни за отделните студенти

## Какво научих

**Николай Геолтиев:**

По време на работата по проекта се научих да използвам display:flex и display:grid, чрез които многократно се уселсни процесът по позициониране на HTML елементите. Също така се запознах с fetch и работата с асинхронни функции, чрез които да изращам заявки към сървърни скриптове.

Придобих умения за свързавне на frontend и backend частта на една система и за работа с PHP за извършване на комуникация с база данни.

Упражних и работата с DOM елементи в JavaScript и използването на Event Listener-и за следене на потребтилеска активност.

**Ерик Здравков:**

По време на работата по проекта се научих да работя с HTML, CSS, JavaScript и PHP. Научих се как се прави интеграция с база данни.

Разбрах как става връзката между клиент и сървър и REST архитектурата. Научих се да използвам load registers и exception handlers.

Разбрах как работят сесиите. Упражних DOM манипулацията и се запознах с асинхронните заявки в javascript.

## Използвани източници