A close up of a logo

Description automatically generated

**Факултет по компютърни системи и технологии**

**Тема: Българска литература**

**Проект по Програмиране за мобилни устройства**

Изготвили:

1. Валентин Георгиев Александров, гр. 46, ф.н. 121216074

2. Елена Венцеславова Антонова, гр. 50, ф.н. 121216165

Научен ръководител: доц. д-р. инж. Антония Ташева

Съдържание:

I. Увод

1) Защо избрахме да правим това приложение?

2) Какво има на пазара в момента?

3) Предимства и недостатъци на други приложения

4) С какво се отличаваме?

II. Проектиране

I. Увод

1) Защо избрахме да правим това приложение?

В днешно време достъпът до каквато и да е информация е по-голям от всякога, което се отразява на всеки аспект от живота ни, включително начина, по който учениците в основното училище и гимназията усвояват нови уроци. Една основна част от образованието ни е познанието върху българския език и литература. Ние сме се съсредоточили върху разнообразното творчество на различни български писатели, с цел да повишим интереса както на ученици, така и на хора с увреждания към богатото наследство на родната литература.

Решение на основния проблем - търсене на иновативен и интересен начин за усвояване на знания по българска литература чрез:

- Опростен потребителски интерфейс

- Предоставена информация за подготовка на учащите

- Лесно търсене

По-важни изгледи/екрани за визуализация на:

- различни български автори

- техните биография/творби/мотиви

- тестове върху различни биографии/творби/конкретен автор

- потребителска информация:

- статистики върху академичното представяне на потребителя

- ранглисти за резултати от различни тестове

Преходите между гореизброените екрани са плавни, осъществяват се чрез вграденото меню за навигация на всяко мобилно устройство.

2) Какво има на пазара в момента?

След дълго проучване установихме, че има приложения с аналогична цел - опознаване на българското наследство и култура под формата на конкурентна игра. Но те по-скоро са предназначени за обогатяване на българската обща култура като цяло - подбрани са разнообразни въпроси, най-вече върху историята и географията на България. Обаче, главно сме разгледали чуждестранни приложения, за да добием какви формати и стандарти на тестване са общоприети.

А) Най-известното приложение у нас с такава цел е Triviador - интелектуална игра с елементи на стратегия, създадена през 2002 г. в Унгария, стартирала в България през 2008 г.

Б) Trivia Crack - отличава се с това, че категориите с въпроси (наука, история, изкуство и т.н.) са представени чрез талисмани, които трябва да се съберат по време на играта.

В) Quiz of Knowledge - за разлика от повечето случаи, тук не се изисква интернет връзка и категориите с въпроси са 16. То служеше за наше вдъхновение в процеса на разработка.

Г) History Quiz - характеризира се с относителна простота и фокус върху една категория знания.



1. Triviador

4. History Quiz

3. Quiz of Knowledge

2. Trivia Crack

3) Предимства и недостатъци на други приложения

Обобщили сме следните плюсове и минуси на гореизброените приложения със сходно предназначение от потребителска гледна точка:

А) Предимства:

- разнообразен и увлекателен подход на представяне на различни факти и въпроси към тях

- точки за всеки верен отговор, които могат да се инвестират в изграждане на собствен виртуален свят - село, град или цяла империя (фиг. 1)

- в някои приложения времето за отговаряне на въпросите е ограничено, стимулирайки по-бързо мислене (фиг. 1 и 2)

Б) Недостатъци:

- несъобразителност с версията на мобилното устройство, т.е. приложението е „тежко“, което води до значително забавяне на заявките (фиг. 1 и 2)

- липса на източници на информация, върху която се задават въпросите

- много силна зависимост от интернет - най-честото оплакване е честопрекъсващата връзка към сървъра (фиг. 1)

4) С какво се отличаваме?

Стигнахме до заключението, че макар да са налични подобни приложения, няма такова, което конкретно се обвързва с точните знания, необходими в училище, особено в областта на литературата. Към днешна дата младите хора учат и затвърждават нова информация бързо и за кратко време поради динамичното темпо на развитие, с което светът се движи. Следователно представянето на тази информация по кратък, но ясен начин, изискващ непрестанното ѝ утвърждаване, би се увенчало с голям успех.

Също така забелязахме, че много малко продукти са насочени и към хора с увреждания, главно незрящи. Стремим се да направим своето приложение възможно най-достъпно, затова сме добавили допълнителни функционалности, спомагащи за тази цел.

II. Проектиране

1) Целева група

Първоначалната идея за нашето приложение произхожда от факта, че заради българската образователна система стремежът за трупане и усвояване на нови знания е силно занижен в младото поколение. Децата днес мислят коренно различно и това трябва да е приоритет в изграждането на нови методики и курсове, а не обратното.

По-нататък помислихме на коя друга група хора подобен продукт би бил полезен и идеята се доразви в още една посока. Всеки човек има право на достъп до източници на знания под каквато и да е форма. Затова има вградена “Text-to-Speech” функционалност.

2) Използвани данни:

Релационна база данни - PostgreSQL (система за управление на БД). Някои по-важни entities са за:

A screenshot of a cell phone

Description automatically generated- автори:

- творби:

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated- потребители:

3) Функционални изисквания

А) Навигация:

|  |  |
| --- | --- |
| Бутон (пренасочване) | Дестинация |
| 1. „Вход“ | Страница с главните опции - за автори, тестове или потребител |
| 2. „Автори“ | Списък с всички автори |
| 3. Снимка на автор | Страница с допълнителни препратки за конкретен автор |
| 4. „Биография“ | Кратка биогарфия на автора |
| 5. „Творби“ | Списък с творбите на автора |
| 6. Заглавие на творба | Кратък анализ на творбата |
| 7. „Мотиви“ | Списък с главните мотиви на автора, всеки с кратко описание |
| 8. „Тестове“ | Опции за тип тест - върху биографии, творби или конкретен автор |
| 9. „Биографии“ | Списък с тестове върху разлчни биографии |
| 10. „Творби“ | Списък с тестове върху различни творби |
| 11. „Автори“ | Списък с тестове, всеки върху определен автор |
| 12. „Потребител“ | Страница с бутони за допълнителна информация за потребителя (статистики/ранглисти) и опция за изход |
| 13. „Статистики“ | Страница с опции за три статистики, всяка върху резултатите от различните типове тестове |
| 14. „Ранглисти“ | Страница с опции за три ранглисти, всеки върху резултатите от различните типове тестове |
| 15. „Изход“ | Начална страница |

Б) Функционалности:

|  |  |
| --- | --- |
| Функционалност | Резултат |
| 1. Запомняне на log-нато състояние | Няма необходимост потребителят да влиза всеки път. |
| 2. Добавяне на потребителска снимка | Потребителят може да избере профилна снимка от галерията си. |
| 3. Глобална търсачка | Бързо намиране на информация. |
| 4. Търсене по ключови думи | Аналогия на клавишната комбинация „Ctrl+F“. |
| 5. Гласово търсене | Търсене без необходимост от писмено въвеждане. |
| 6. Задържане върху екрана | Активиране на “Text-to-Speech” функционалност. |

2) Нефункционални изисквания

А) Back End:

- организация чрез Spring MVC - архитектурен модел за изграждане на потребителски интерфейси. Разделя дадено софтуерно приложение на три взаимносвързани части, така че двете представяния на информация (вътрешна/външна) да са отделени.

- комбинация на различн технологии за достъп до данни чрез Spring Data

- работа с JPA-базирани repository-та за лесна реализация на CRUD (Create-Read-Update-Delete) операции чрез Spring Data JPA (Java Persistence API)

- автентикация чрез JSON Web Token

- контрол над автентикацията и авторизацията на клиента чрез Spring Security: криптиране на пароли с BCrypt

- трансформация на ООП класове (entities) в таблици на релационна БД чрез Hibernate

- пренос на информация между клиента и сървъра чрез DTO (Data Transfer Object)

- тестване на сървъра с Postman

- контрол на версиите с Git

- съхранение на изображения в Cloudinary

Б) Бази данни:

- структуриране чрез PostgreSQL

- управление чрез PG Admin

A screenshot of a cell phone

Description automatically generated- ER диаграма:

В) Front End (JavaScript - превод на Java за Android и Objective-C за iOS):

- разработка на Cross-Platform приложението с React Native, използвайки библиотеките: React Navigation, RN Elements, RN Gesture Handler, RN TTS

- сваляне и обновяване на JS библиотеки чрез npm (Node Package Manager)

- инсталиране, обновяване и изтриване на пакети за Android SDK чрез SDK Manager

- Rest параметри, Spread оператори и Lambda изрази - ES6

- по-добро разпределение на кода и прилагане на Single Responsibility Principle с ES6 модули

- имплементация на компонент-базирана архитектура

- асинхронни заявки (AJAX) за комуникация със сървъра

- запазване на информация в паметта на мобилното устройство с Async Storage

- асинхронно програмиране с promises

- запазване на състоянието на приложението с React State

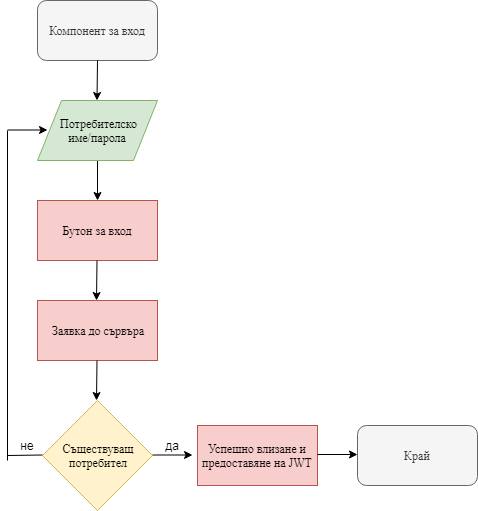
- поддръжка на Android кода с „gradlew clean“

3) Потребителски истории

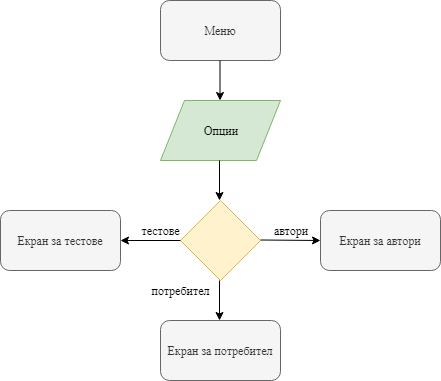
|  |  |  |  |
| --- | --- | --- | --- |
| № | като | искам да | за да |
| 1. | система (сървър) | получавам заявките с JSON Web Token в Header частта | автентикирам потребителя |
| 2. | система | държа паролите на потребителите криптирани | гарантирам сигурност на личните им данни |
| 3. | потребител | ми бъде запомнено състоянието | не се налага да влизам всеки път |
| 4. | потребител | получавам списък с български автори, с техни снимки | разгледам допълнителна информация за всеки от тях |
| 5. | потребител | мога да филтрирам авторите по техните имена | намирам конкретен автор по-лесно |
| 6. | потребител | правя тестове върху биография и творчество | си затвърдя знанията |
| 7. | потребител | се визуализира статистика на моя прогрес на базата на направените от мен тестове | следя своя напредък |
| 8. | потребител | да се визуализира ранглист | се сравнявам с другите потребители |
| 9. | потребител | слушам текстовете, когато задържам екрана | усвоя материала по-пълноценно |
| 10. | потребител | извличам информация чрез гласово търсене | имам по-приятно потребителско изживяване |

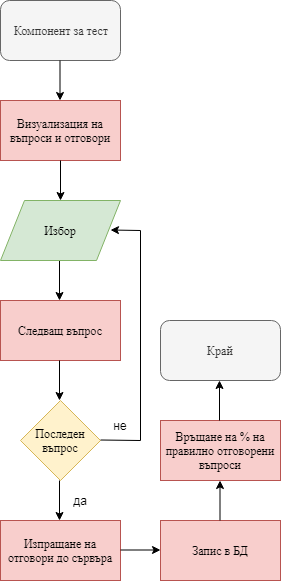
4) Workflow диаграми

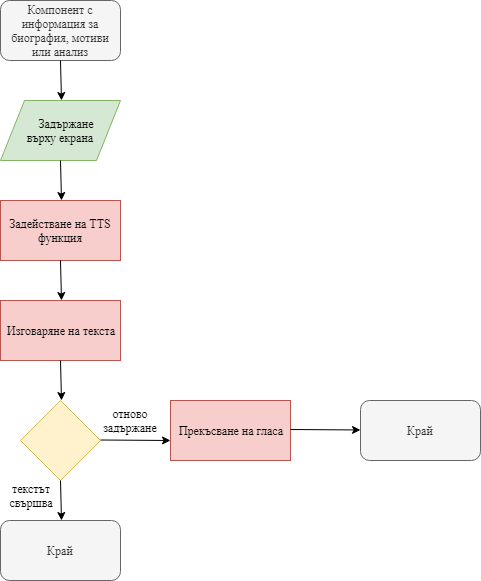
А) Вход на потребителя:



Б) Избор от основното меню:



В) Правене на тест:

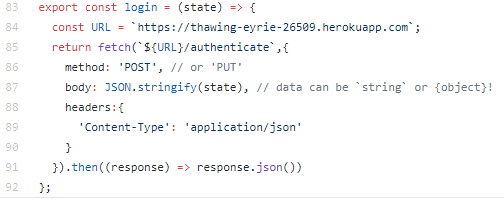
Г) Text-to-Speech функция:

III. Реализация

1) Кратки фрагменти код (практическа реализация)

А) Автентикация на потребители:

Заявка, отговорна за изпращане на потребителско име и парола за автентикация до сървъра:



В state обекта има полета:

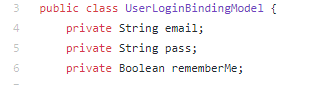
* user
* pass

Те биват изпратени до сървърът ни, който е хостнат в Heroku.

Сървърът приема заявка на /authenticate чрез следния метод от контролера UserJWTController:



Controller-ът получава UserLoginBindingModel, което е клас със следните полета:

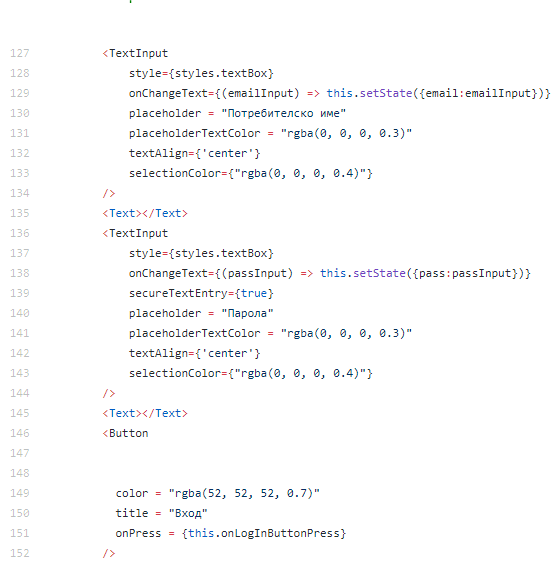


Те съвпадат с тези, които се изпращат от клиентската страна, и по-този начин Spring ги map-ва от тялото на POST заяката към класа UserLoginBindingModel.

След това се проверява дали тези email и pass принадлежат на даден потребител и ако това е така, то сървърът връща token:



В React Native това е компонентът, отговорен за изпращането на данните на потребителя:



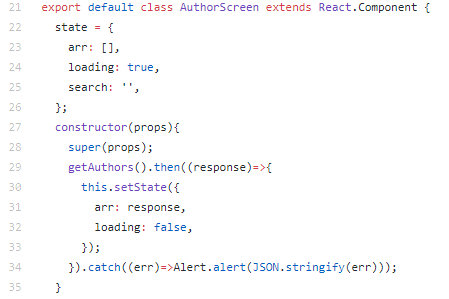
Всеки път, когато потребителят въведе нещо в полето за потребителско име или парола, се извикват callback функции, които обновяват state-ът, който ще се прикачи към заявката до сървъра при натискането на бутонът вход, който от своя страна ще извика следната функция:



onLogInButtonPress() е декларирана като асинхронна arrow функция (ламбда израз), защото ще използва sendRequest(), което от своя страна връща promise и е необходимо да го await-нем, докато приключи.

Това става асинхронно и event loop-ът на JS не се задръства.

Б) Показване на български автори и филтрирането им по име:



AuthorScreen е отговорен за изпращане на заявка до сървъра, която ще му върне масив от обекти, съдържащи:

* Имена
* ID
* URL към снимка на автора

State обектът запазва този масив и обновява състоянието си от режим на зареждане в режим на готовност.

Това е функцията, която прави GET заявка до сървъра:



От глобалния обект взимаме token, който е получен при автентикацията. Прикачваме го към header частта на заявката и я изпращаме. Конфигурирали сме всички ресурси на сървъра да изискват автентикация.

От страна на сървъра, тази заявка минава през Spring филтър:



Той проверява дали token-ът е валиден.

След това заявката стига до AuthorController:



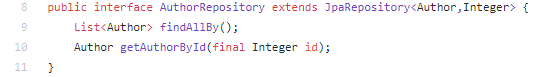
Той използва добрите практики от Spring и ни позволява да използваме dependency injection за следните услуги:

* AuthorService
* WorkService

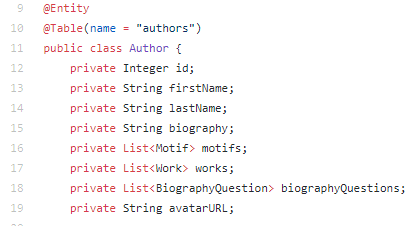
Контролерът на route /authors приема заявката и извиквка метод findAll() на AuthorService, който е имплементиран от UserServiceImpl по следния начин:



Неговата роля е и да извлече списък на всички автори от AuthorRepository. Има dependency, което е интерфейс, играещ ролята на връзка към базата данни:

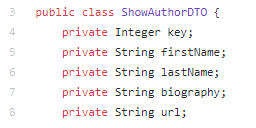


JPARepository ни предоставя готови методи за работа с каквато и да е релационна база данни. По този начин ние не пишем SQL и сме много по-ефикасни. Това Repository работи със следното entity за автор:



Освен данните, които потребителят изисква, авторите имат и списъци от своите мотиви, творби и биографични въпроси върху тях.

Тъй като това entity има прекалено много информация за нуждите на заявката, създаваме следното DTO (Data Transfer Object):



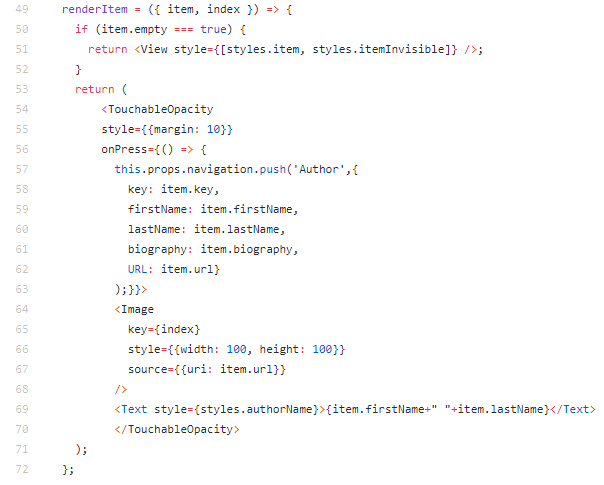
Използвайки този подход, получаваме още едно преимущество - избягване на circular dependency, когато връщаме данните под формата на JSON!

Това е причината Author Service и Controller да връщат:

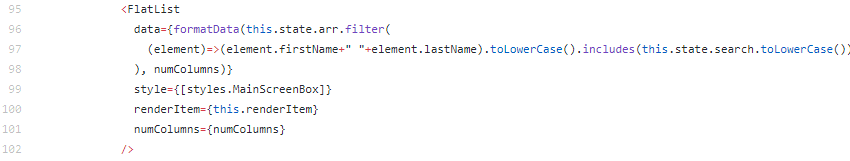


В клиентската страна авторите се визуализират като снимки, под които са прикачени техните имена.

Тази функция е отговорна за тази задача:

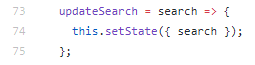


И се извиква в render() функцията:



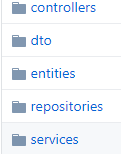
Входните данни се филтрират спрямо state обекта, от който се взима текста на търсачката.

Всеки път когато се въвежда текст в търсачката, се извиква:



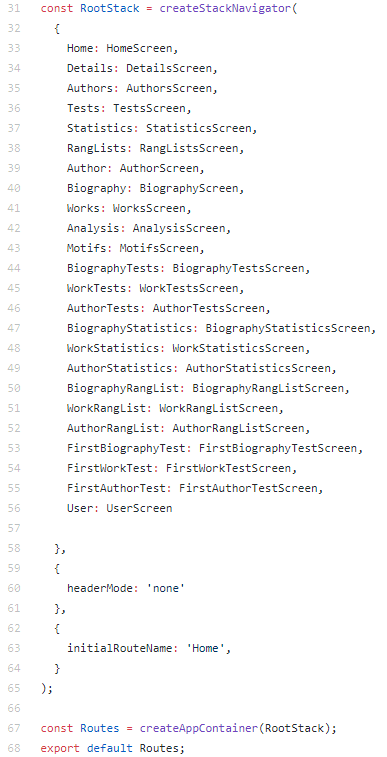
След това се извиква render() функцията и тя визуализира търсения автор спрямо въведеното от потребителя в търсачката име. Facebook са оптимизирали този процес и той по никакъв начин не влоша бързодействието на приложението.

В сървъра всеки модул като Автори, Потребители, Тестове, Резултати има следните компоненти:

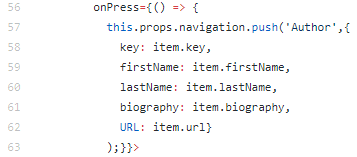


В) Навигация в приложението

Става посредством routes.js. В него е записано как се достъпва всеки екран от клиентската страна. Това позволява лесно надграждане на сегашната логика, защото ако имаме нужда от нов екран, то просто добавяме в routes.js, и вече спокойно може да го използваме в приложението.



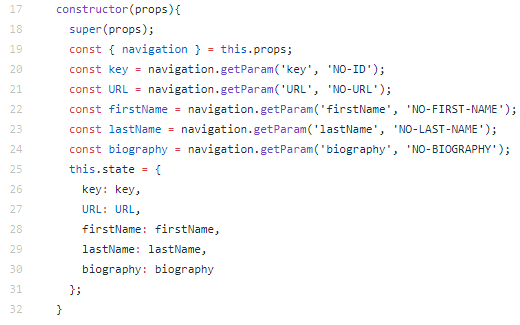
Навигирането между екраните става чрез this.props.navigation.push(‘screenName’,objectData);



В този пример screenName е ‘Author’, което в routes.js отговаря на екрана за автори (AuthorScreen.js).

objectData е обект, който изпращаме на екрана и към който ще навигираме.

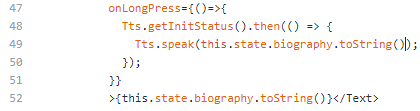
Ето как се получават данните в дестинационния екран:



След като извлечем данните, ги запазваме в state, за да ги използваме из компонента.

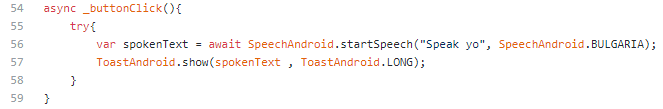
Г) Text-to-Speech

Това е пример как използваме функционалността Text to Speech:



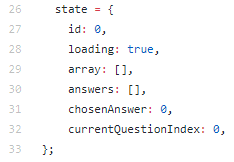
В горния фрагмент, като се задържи екрана за по-дълго време, се активира callback функцията, която ще стартира Text-to-Speech функционалността и ще изговори подадения текст.

Това е функцията, с която позволяваме на потребителя да изкаже какво би искал да търси в приложението:



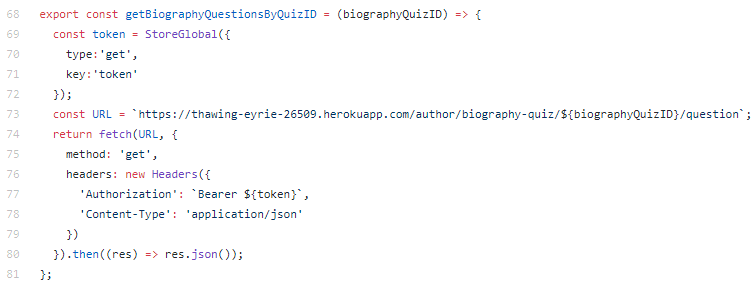
Д) Правене на тестове:

State на компонента:

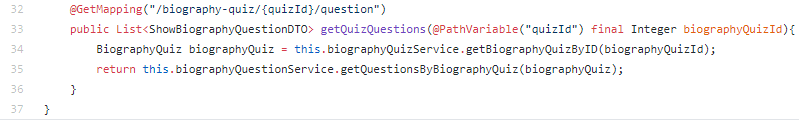


В array ще вкараме въпросите, дошли от заявката към сървъра, а пък в answers ще запишем отговорите от теста.

Ето я заявката към сървъра:

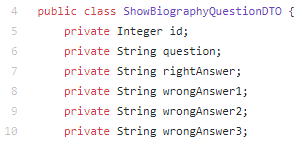


По id на даден quiz правим заявка, която ще извлече всички негови въпроси.  
Заявката се приема от BiographyQuizController на метода:

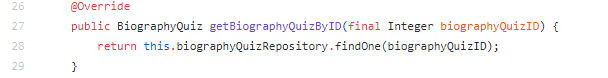


Неговата роля е да извика услуга, която ще намери посочения biographyQuiz по ID и след като го вземе, ще извлече всички въпроси, които се отнасят към него, и ще ги върне като лист от Data Transfer Object.

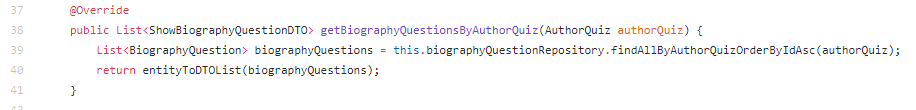
Ето какво представлява ShowBiographyQuestionDTO:



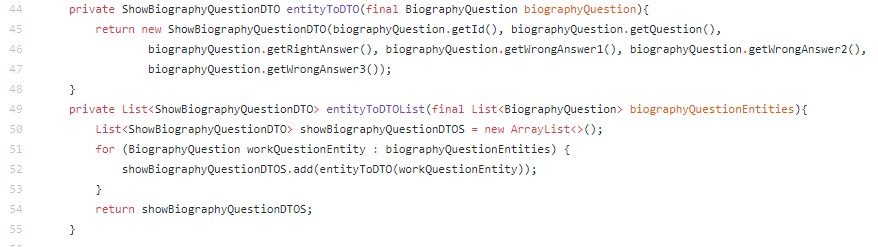
Това е услугата, която намира дадения bigraphyQuiz по ID:



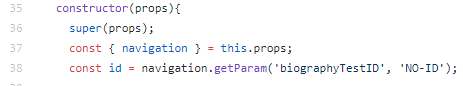
A това е услугата, която по даден bigraphyQuiz намира всички въпроси към него:



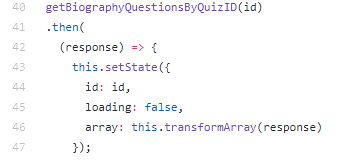
Конвертирането на BigraphyQuestion, което е entity, към ShowBiographyQuestionDTO става чрез следните два метода:



В компонента от клиентската страна получаваме id-то на избрания тест:



Ще го подадем на заявката:

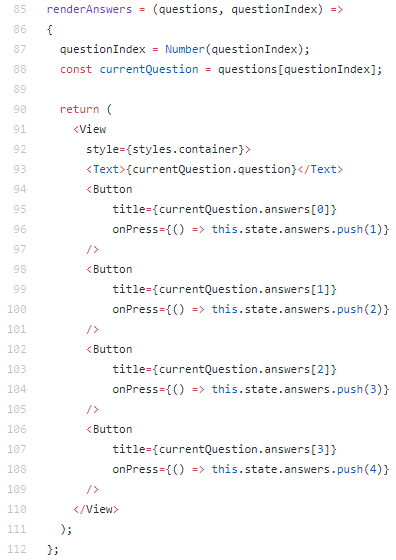


и обновяваме състоянието на компонента.

Сега може да визуализираме тези въпроси върху екрана, като взехме решение да ги показваме един по един:

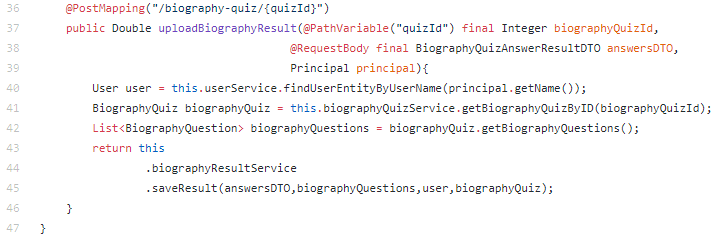


Ето и функцията, която рендира въпрос по зададен индекс:



Когато потребителят избере даден отговор, той се добавя като индекс в масив.

С POST заявка изпращаме отговорените въпроси на сървърът, който ги приема в BiographyResultController:



Ролята на uploadBiographyResult() метода е да извика услуга, която да намери теста, и да му вземе всичките въпроси.

В услугата biographyResultService се подават отговорените въпроси, въпросите от базата данни, потребителят, който ги е отговорил, и тестът. Услугата прави проверка колко верни отговора има потребителят, пресмята неговия резултат, запазва го в базата данни и връща процента на клиентската страна, която ще му го визуализира.

IV. Потребителско ръководство

1) Вход на потребителя:



Влизане с имейл и парола

2) Главно меню:

Избор на трите главни опции