

Факултет по компютърни системи и технологии



Проект по Програмиране за мобилни устройства **Тема: Българска литература**

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

- 1) Целева група
- 2) Използвани данни
- 3) Функционални изисквания
- 4) Нефункционални изисквания
- 5) Потребителски истории
- 6) Workflow диаграми

III. Реализация

- 1) Кратки фрагменти код (практическа реализация)

IV. Потребителско ръководство

- 1) Вход на потребителя
- 2) Главно меню
- 3) Тест
- 4) Ранглист

V. Заключение

- 1) Готов ли е продуктът за употреба?
- 2) Среща ли нуждите на потребители?
- 3) Конкурентноспособен ли е на съществуващите решения?

VI. Източници на информация

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 представа 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



2. Trivia Crack



3. Quiz of Knowledge



4. History Quiz

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

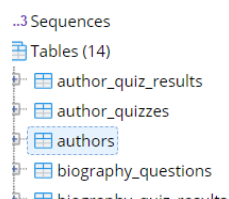
Първоначалната идея за нашето приложение произхожда от факта, че заради българската образователна система стремежът за трупане и усвояване на нови знания е силно занижен в младото поколение. Децата днес мислят коренно различно и това трябва да е приоритет в изграждането на нови методики и курсове, а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

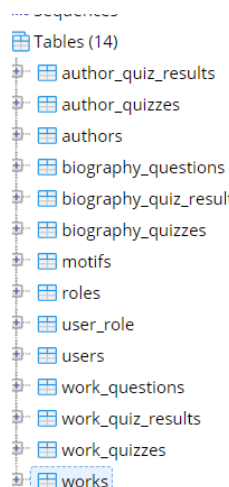
Релационна база данни - PostgreSQL (система за управление на БД). Някои по-важни entities са за:

- автори:



	id	avatar	biography	first_name	last_name
	[PK] integer	character varying (255)	text	character varying (255)	character varying (255)
1	1	https://res.cloudinary.com/...	Иван Минч...	Иван	Вазов
2	2	https://res.cloudinary.com/...	Йордан Сте...	Йордан	Йовков

- творби:



	id	description	title	author_id
	[PK] integer	text	character varying (255)	integer
1	1	Одата, също ...	Българският език	1
2	2	Повод за съз...	Една българка	1
3	3	Човешките х...	По жицата	2

- потребители:

Tables (14)				
author_quiz_result				
author_quizzes				
authors				
biography_questio				
biography_quiz_re				
biography_quizzes				
motifs				
roles				
user_role				
users				

	id	email	password	avatar
	[PK] integer	character varying (255)	character varying (255)	character varying (255)
1	1	themadhat6@gmail.com	\$2a\$10\$WcoUkxCTcxAlON...	https://res.cloudinary.com/...

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 биография 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 различни 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.

4) Нефункционални изисквания

А) Back End:

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

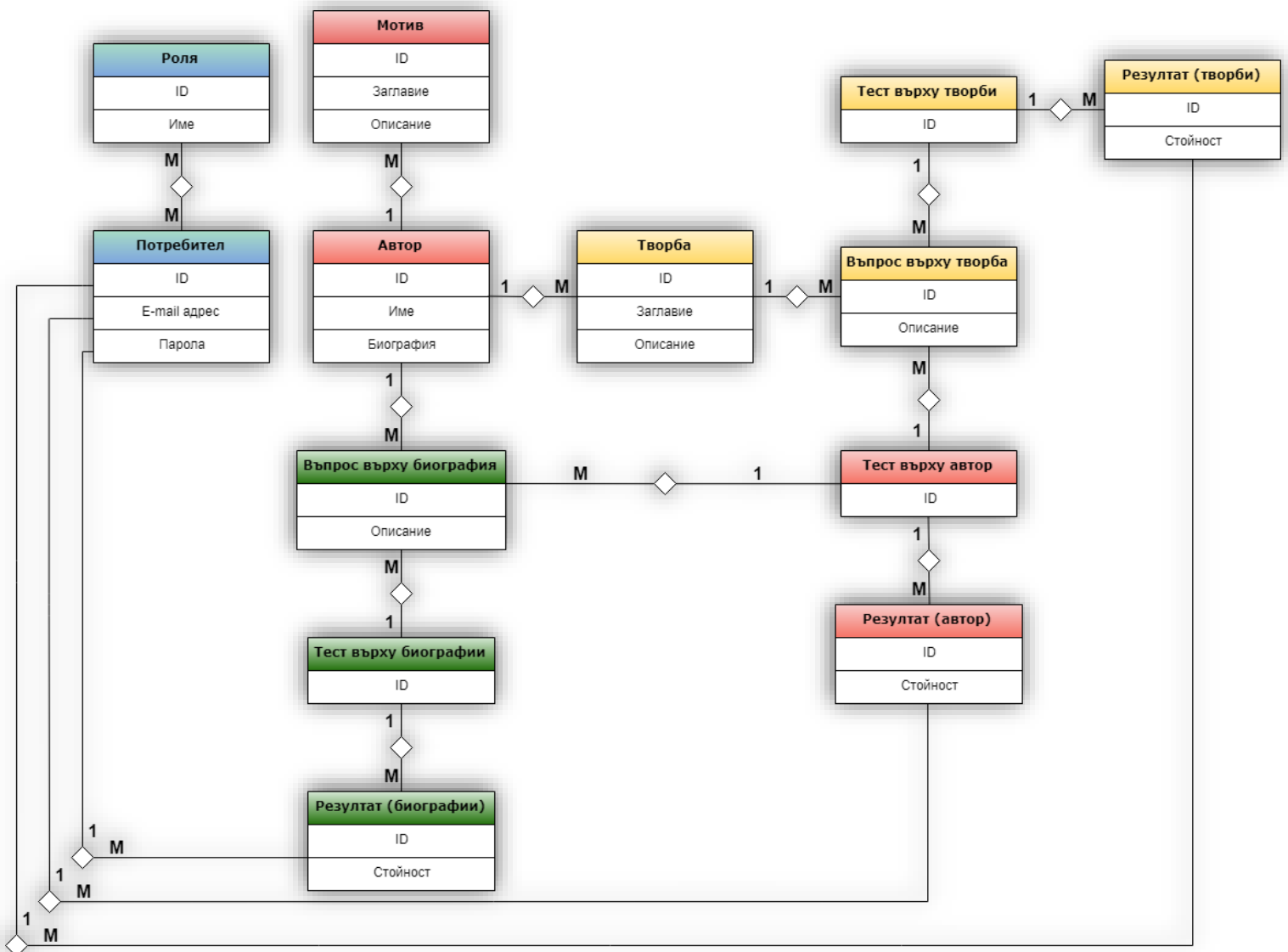
- съхранение на изображения в Cloudinary

Б) Бази данни:

- структуриране чрез PostgreSQL

- управление чрез PG Admin

- 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 сървър
- 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“

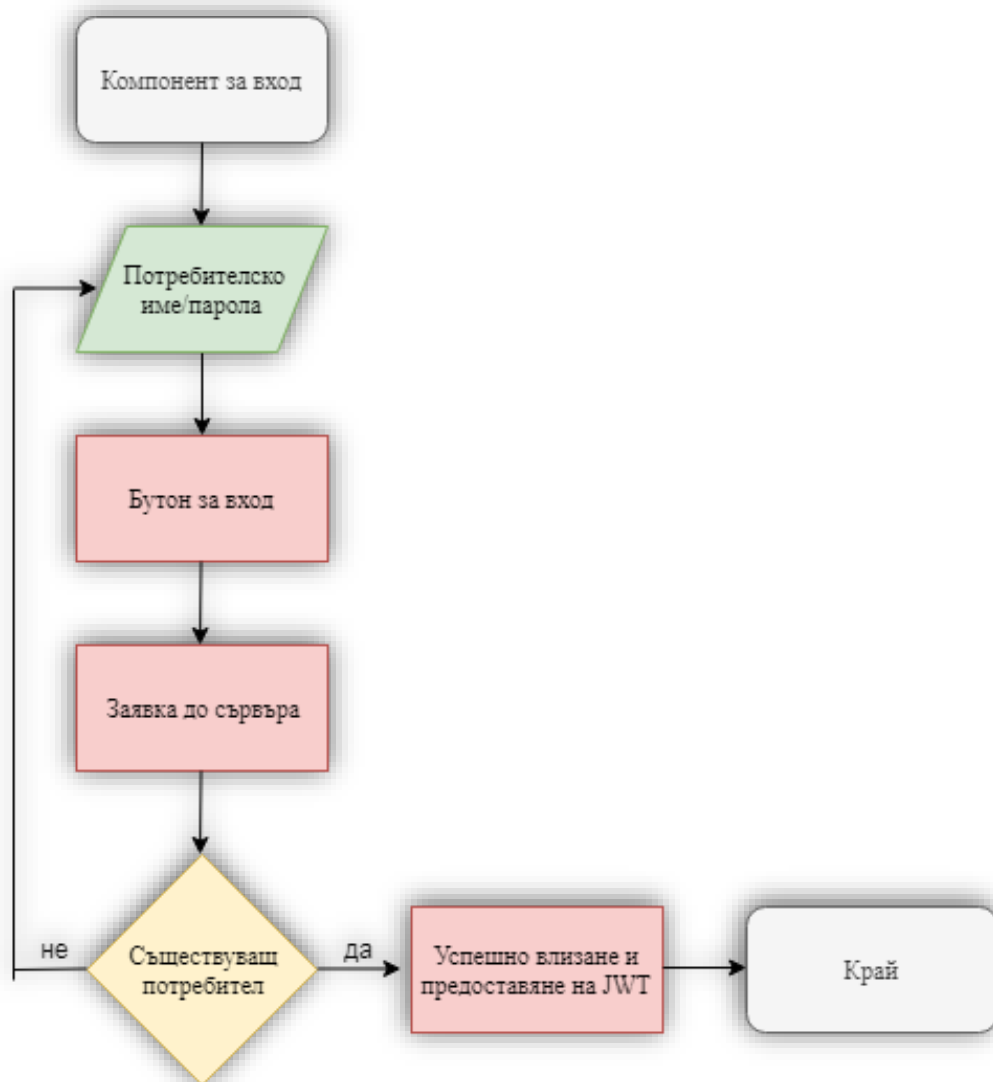
5) Потребителски истории

№	като	искам да	за да
1.	система (сървър)	получавам заявките с JSON Web Token в тяхната Header част	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	разглеждам 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	следа своя напредък
8.	потребител	да се визуализира ранглист	се сравнявам с другите потребители

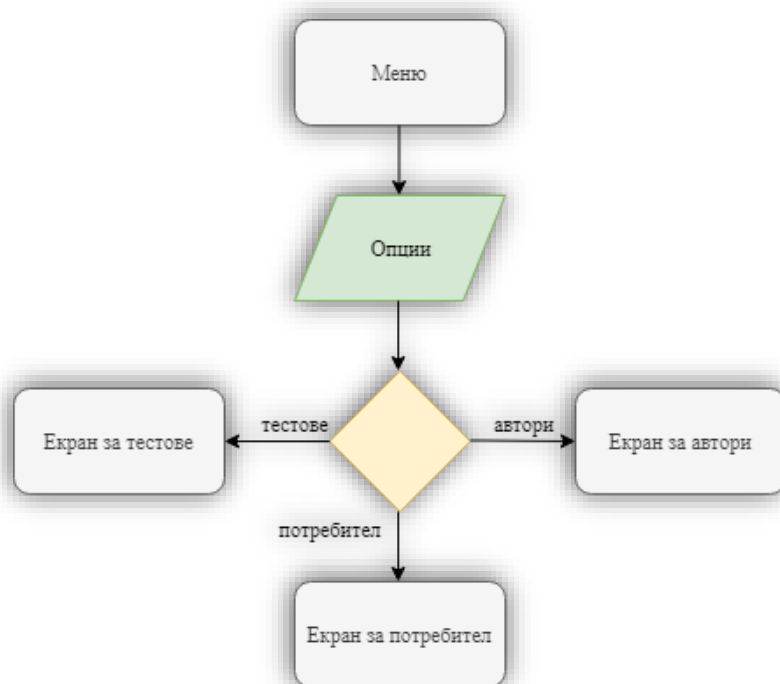
9.	потребител	слушам текстовете, когато задържам екрана	усвоя материала по-пълноценно
10.	потребител	извличам информация чрез гласово търсене	имам по-приятно потребителско изживяване

6) Workflow диаграми

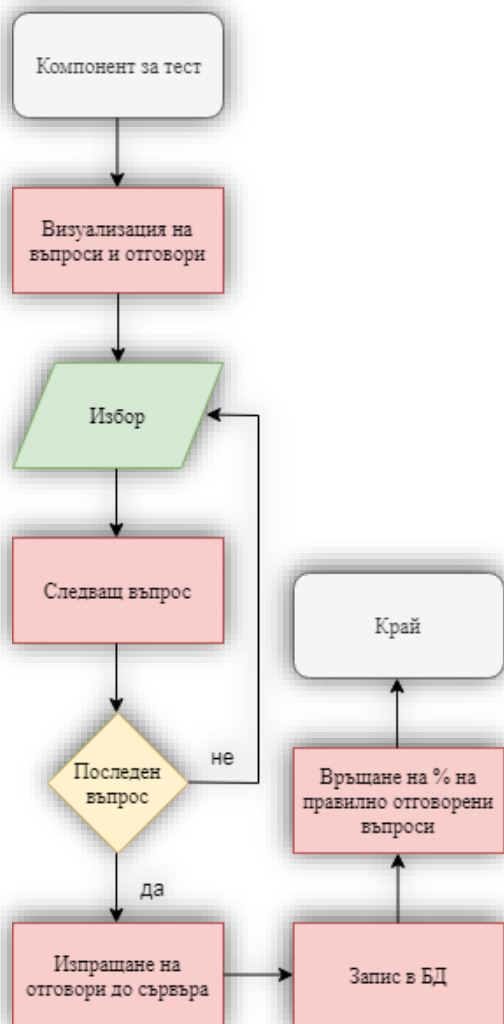
А) Вход на потребителя:



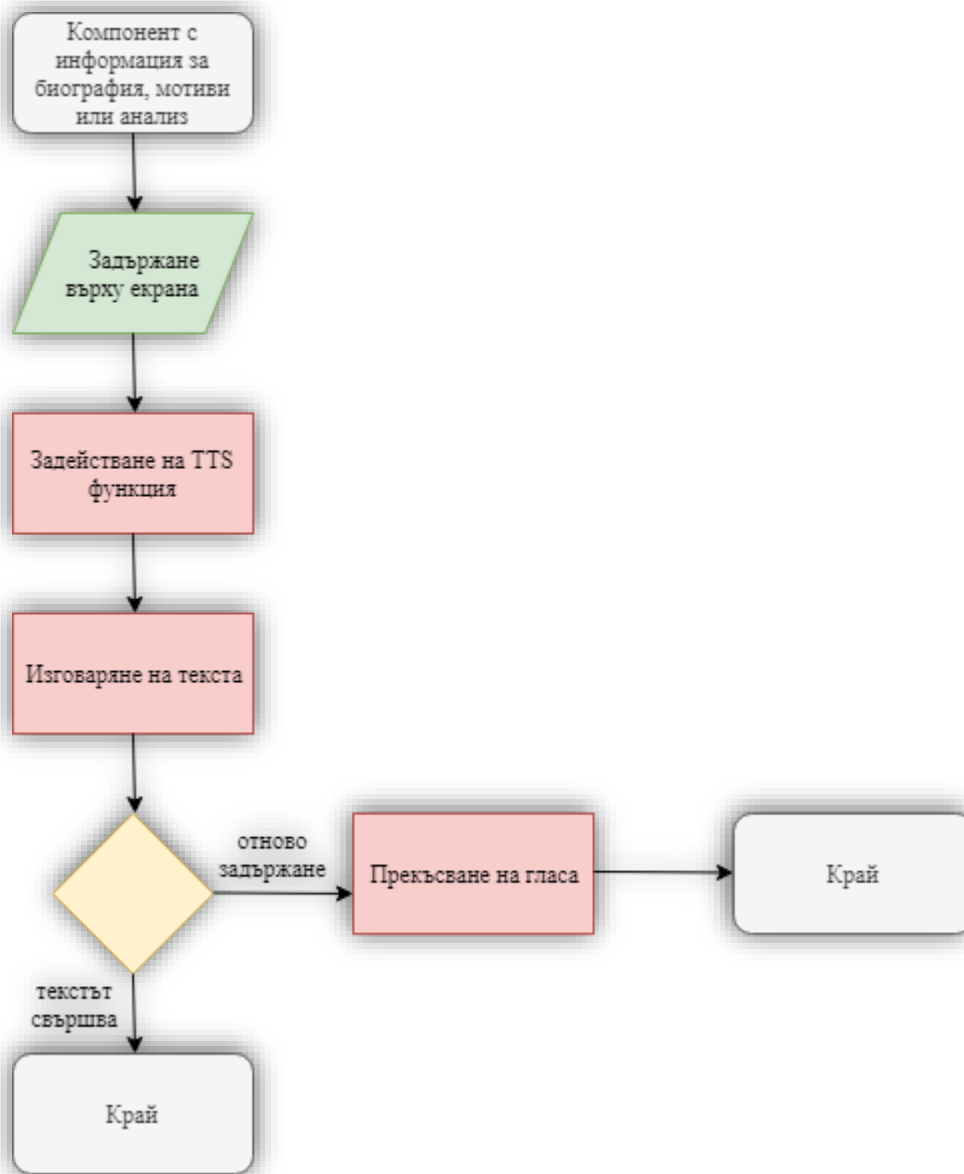
Б) Избор от основното меню:



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:

```
83 export const login = (state) => {
84   const URL = `https://thawing-eyrie-26509.herokuapp.com`;
85   return fetch(`${URL}/authenticate`,{
86     method: 'POST', // or 'PUT'
87     body: JSON.stringify(state), // data can be `string` or {object}!
88     headers:{
89       'Content-Type': 'application/json'
90     }
91   }).then((response) => response.json())
92   };
```

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:

```
33 @CrossOrigin(origins = "http://localhost:3000")
34 @PostMapping("/authenticate")
35 public ResponseEntity authorize(@Valid @RequestBody UserLoginBindingModel userLoginBindingModel, HttpServletResponse response) {
36     UsernamePasswordAuthenticationToken authenticationToken =
37         new UsernamePasswordAuthenticationToken(userLoginBindingModel.getEmail(), userLoginBindingModel.getPass());
38
39     try {
40         Authentication authentication = this.authenticationManager.authenticate(authenticationToken);
41         SecurityContextHolder.getContext().setAuthentication(authentication);
42         boolean rememberMe = (userLoginBindingModel.getRememberMe() == null) ? false : userLoginBindingModel.getRememberMe();
43
44         String jwt = tokenProvider.createToken(authentication, rememberMe);
45
46         response.addHeader(JWTConfigurer.AUTHORIZATION_HEADER, "Bearer " + jwt);
47         return ResponseEntity.ok(new JWTToken(jwt));
48     } catch (AuthenticationException ae) {
49         // log.trace("Authentication exception trace: {}", ae);
50         return new ResponseEntity<>(Collections.singletonMap("AuthenticationException",
51             ae.getLocalizedMessage()), HttpStatus.UNAUTHORIZED);
52     }
53 }
54 }
```

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

```
3 public class UserLoginBindingModel {
4     private String email;
5     private String pass;
6     private Boolean rememberMe;
7 }
```

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:

```
"token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0aGVtYWRoYXR0ZXI2QGdtYWlsLmNv"
```

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

```
127 <TextInput
128     style={styles.textBox}
129     onChangeText={({emailInput) => this.setState({email:emailInput})}
130     placeholder = "Потребителско име"
131     placeholderTextColor = "rgba(0, 0, 0, 0.3)"
132     textAlign={'center'}
133     selectionColor={"rgba(0, 0, 0, 0.4)"}
134 />
135 <Text></Text>
136 <TextInput
137     style={styles.textBox}
138     onChangeText={({passInput) => this.setState({pass:passInput})}
139     secureTextEntry={true}
140     placeholder = "Парола"
141     placeholderTextColor = "rgba(0, 0, 0, 0.3)"
142     textAlign={'center'}
143     selectionColor={"rgba(0, 0, 0, 0.4)"}
144 />
145 <Text></Text>
146 <Button
147
148
149     color = "rgba(52, 52, 52, 0.7)"
150     title = "Вход"
151     onPress = {this.onLogInButtonPress}
152 />
```

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:

```
78     sendRequest = () => {
79         this.setState({loading:true});
80         login(this.state)
81         .then((response) => {
82             this.setState({loading:false});
83             if(JSON.stringify(response).includes("Bad credentials")){
84                 Alert.alert("Няма такъв потребител.");
85                 return;
86             }
87             this.setState({ token: response.token });
88             StoreGlobal({
89                 type:'set',
90                 key:'token',
91                 value: `${response.token}`
92             });
93             this._storeData('email',this.state.email);
94             this._storeData('pass',this.state.pass);
95             this._storeData('save','true');
96             this.props.navigation.push('Details');
97         }).catch(
98             (error) => {
99                 this.setState({loading:false});
100                 Alert.alert('Wi-fi error');
101             }
102         );
103     };
104     onLogInButtonPress = async () => {
105         await this.sendRequest();
106     };
```

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:

```
21 export default class AuthorScreen extends React.Component {
22   state = {
23     arr: [],
24     loading: true,
25     search: '',
26   };
27   constructor(props){
28     super(props);
29     getAuthors().then((response)=>{
30       this.setState({
31         arr: response,
32         loading: false,
33       });
34     }).catch((err)=>Alert.alert(JSON.stringify(err)));
35   }
}
```

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:

```
9 export const getAuthors = () => {
10   const token = StoreGlobal({
11     type: 'get',
12     key: 'token'
13   });
14   const URL = "https://thawing-eyrie-26509.herokuapp.com/authors";
15   return fetch(URL, {
16     method: 'get',
17     headers: new Headers({
18       'Authorization': `Bearer ${token}`,
19       'Content-Type': 'application/json'
20     })
21   }).then((res) => res.json());
22 };
```

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:

```
33     @Override
34     public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
35         throws IOException, ServletException {
36         try {
37             HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
38             String jwt = resolveToken(httpServletRequest);
39             if (StringUtils.hasText(jwt) && this.tokenProvider.validateToken(jwt)) {
40                 Authentication authentication = this.tokenProvider.getAuthentication(jwt);
41                 User user = (User)authentication.getPrincipal();
42                 System.out.println(user.getEmail()+"1996");
43                 System.out.println(user.getRoles().toString());
44
45
46                 SecurityContextHolder.getContext().setAuthentication(authentication);
47             }
48             filterChain.doFilter(servletRequest, servletResponse);
49         } catch (ExpiredJwtException eje) {
50             log.info("Security exception for user {} - {}",
51                 eje.getClaims().getSubject(), eje.getMessage());
52
53             log.trace("Security exception trace: {}", eje);
54             ((HttpServletRequest) servletResponse).setStatus(HttpStatus.UNAUTHORIZED);
55         }
56     }
```

Той проверява дали token-ът е валиден.

След това заявката стига до AuthorController:

```
15     @RestController
16     @RequestMapping("/authors")
17     public class AuthorController {
18         private AuthorService authorService;
19         private WorkService workService;
20
21         @Autowired
22         public AuthorController(final AuthorService authorService, final WorkService workService) {
23             this.authorService = authorService;
24             this.workService = workService;
25         }
26
27         @GetMapping("/{id}")
28         public ShowAuthorDTO getAuthorById(@PathVariable("id") Integer authorId){
29             return this.authorService.findById(authorId);
30         }
31
32         @GetMapping("")
33         public List<ShowAuthorDTO> all(){
34             return this.authorService.findAll();
35         }
36     }
```

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 извиква метод findAll() на AuthorService, който е имплементиран от UserServiceImpl по следния начин:

```
12  @Service
13  public class AuthorServiceImpl implements AuthorService{
14      private AuthorRepository authorRepository;
15
16      @Autowired
17      public AuthorServiceImpl(final AuthorRepository authorRepository) {
18          this.authorRepository = authorRepository;
19      }
20
21      @Override
22      public List<ShowAuthorDTO> findAll() {
23          List<Author> authorEntities = authorRepository.findAll();
24          List<ShowAuthorDTO> authorDTOS = new ArrayList<>();
25          for(int i = 0; i < authorEntities.size(); i++){
26              Author currentAuthor = authorEntities.get(i);
27              ShowAuthorDTO newAuthorDTO = entityToDTO(currentAuthor);
28              authorDTOS.add(newAuthorDTO);
29          }
30          return authorDTOS;
31      }
32
```

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

```
8  public interface AuthorRepository extends JpaRepository<Author,Integer> {
9      List<Author> findAllBy();
10     Author getAuthorById(final Integer id);
11 }
```

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:

```
9  @Entity
10 @Table(name = "authors")
11 public class Author {
12     private Integer id;
13     private String firstName;
14     private String lastName;
15     private String biography;
16     private List<Motif> motifs;
17     private List<Work> works;
18     private List<BiographyQuestion> biographyQuestions;
19     private String avatarURL;
20 }
```

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):

```
3  public class ShowAuthorDTO {
4      private Integer key;
5      private String firstName;
6      private String lastName;
7      private String biography;
8      private String url;
9  }
```

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:

```
List<ShowAuthorDTO>
```

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

Тази функция е отговорна за тази задача:

```

49   renderItem = ({ item, index }) => {
50     if (item.empty === true) {
51       return <View style={[styles.item, styles.itemInvisible]} />;
52     }
53     return (
54       <TouchableOpacity
55         style={{margin: 10}}
56         onPress={() => {
57           this.props.navigation.push('Author',{
58             key: item.key,
59             firstName: item.firstName,
60             lastName: item.lastName,
61             biography: item.biography,
62             URL: item.url}
63           );}}>
64       <Image
65         key={index}
66         style={{width: 100, height: 100}}
67         source={{uri: item.url}}
68       />
69       <Text style={styles.authorName}>{item.firstName+" "+item.lastName}</Text>
70     </TouchableOpacity>
71   );
72 };

```

И се извиква в render() функцията:

```

95   <FlatList
96     data={formatData(this.state.arr.filter(
97       (element)=>(element.firstName+" "+element.lastName).toLowerCase().includes(this.state.search.toLowerCase()))
98     ), numColumns)}
99     style={[styles.MainScreenBox]}
100     renderItem={this.renderItem}
101     numColumns={numColumns}
102   />

```

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

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

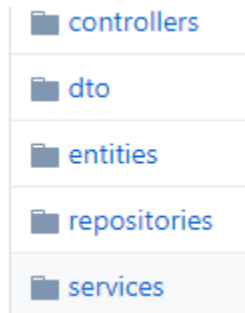
```

73   updateSearch = search => {
74     this.setState({ search });
75   };

```

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.

```

31  const RootStack = createStackNavigator(
32    {
33      Home: HomeScreen,
34      Details: DetailsScreen,
35      Authors: AuthorsScreen,
36      Tests: TestsScreen,
37      Statistics: StatisticsScreen,
38      RangLists: RangListsScreen,
39      Author: AuthorScreen,
40      Biography: BiographyScreen,
41      Works: WorksScreen,
42      Analysis: AnalysisScreen,
43      Motifs: MotifsScreen,
44      BiographyTests: BiographyTestsScreen,
45      WorkTests: WorkTestsScreen,
46      AuthorTests: AuthorTestsScreen,
47      BiographyStatistics: BiographyStatisticsScreen,
48      WorkStatistics: WorkStatisticsScreen,
49      AuthorStatistics: AuthorStatisticsScreen,
50      BiographyRangList: BiographyRangListScreen,
51      WorkRangList: WorkRangListScreen,
52      AuthorRangList: AuthorRangListScreen,
53      FirstBiographyTest: FirstBiographyTestScreen,
54      FirstWorkTest: FirstWorkTestScreen,
55      FirstAuthorTest: FirstAuthorTestScreen,
56      User: UserScreen
57    },
58    {
59      headerMode: 'none'
60    },
61    {
62      initialRouteName: 'Home',
63    }
64  );
65
66
67  const Routes = createAppContainer(RootStack);
68  export default Routes;

```

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

```

56     onPress={() => {
57       this.props.navigation.push('Author',{
58         key: item.key,
59         firstName: item.firstName,
60         lastName: item.lastName,
61         biography: item.biography,
62         URL: item.url}
63     );}}>

```

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:

```

17   constructor(props){
18     super(props);
19     const { navigation } = this.props;
20     const key = navigation.getParam('key', 'NO-ID');
21     const URL = navigation.getParam('URL', 'NO-URL');
22     const firstName = navigation.getParam('firstName', 'NO-FIRST-NAME');
23     const lastName = navigation.getParam('lastName', 'NO-LAST-NAME');
24     const biography = navigation.getParam('biography', 'NO-BIOGRAPHY');
25     this.state = {
26       key: key,
27       URL: URL,
28       firstName: firstName,
29       lastName: lastName,
30       biography: biography
31     };
32   }

```

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:


```

47         onLongPress={()=>{
48             Tts.getInitStatus().then(() => {
49                 Tts.speak(this.state.biography.toString());
50             });
51         }}
52         >{this.state.biography.toString()}</Text>

```

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:

```

54     async _buttonClick(){
55         try{
56             var spokenText = await SpeechAndroid.startSpeech("Speak yo", SpeechAndroid.BULGARIA);
57             ToastAndroid.show(spokenText , ToastAndroid.LONG);
58         }
59     }

```

Д) Правене на тестове:

State на компонента:

```

26     state = {
27         id: 0,
28         loading: true,
29         array: [],
30         answers: [],
31         chosenAnswer: 0,
32         currentQuestionIndex: 0,
33     };

```

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

Ето я заявката към сървъра:

```

68 export const getBiographyQuestionsByQuizID = (biographyQuizID) => {
69   const token = StoreGlobal({
70     type: 'get',
71     key: 'token'
72   });
73   const URL = `https://thawing-eyrie-26509.herokuapp.com/author/biography-quiz/${biographyQuizID}/question`;
74   return fetch(URL, {
75     method: 'get',
76     headers: new Headers({
77       'Authorization': `Bearer ${token}`,
78       'Content-Type': 'application/json'
79     })
80   }).then((res) => res.json());
81 };

```

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

```

32 @GetMapping("/biography-quiz/{quizId}/question")
33 public List<ShowBiographyQuestionDTO> getQuizQuestions(@PathVariable("quizId") final Integer biographyQuizId){
34     BiographyQuiz biographyQuiz = this.biographyQuizService.getBiographyQuizByID(biographyQuizId);
35     return this.biographyQuestionService.getQuestionsByBiographyQuiz(biographyQuiz);
36 }
37 }

```

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:

```

4 public class ShowBiographyQuestionDTO {
5     private Integer id;
6     private String question;
7     private String rightAnswer;
8     private String wrongAnswer1;
9     private String wrongAnswer2;
10    private String wrongAnswer3;

```

Това е услугата, която намира дадения biographyQuiz по ID:

```

26 @Override
27 public BiographyQuiz getBiographyQuizByID(final Integer biographyQuizID) {
28     return this.biographyQuizRepository.findOne(biographyQuizID);
29 }

```

А това е услугата, която по даден biographyQuiz намира всички въпроси към него:

```
37     @Override
38     public List<ShowBiographyQuestionDTO> getBiographyQuestionsByAuthorQuiz(AuthorQuiz authorQuiz) {
39         List<BiographyQuestion> biographyQuestions = this.biographyQuestionRepository.findAllByAuthorQuizOrderByIdAsc(authorQuiz);
40         return entityToDTOList(biographyQuestions);
41     }
42 }
```

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

```
44     private ShowBiographyQuestionDTO entityToDTO(final BiographyQuestion biographyQuestion){
45         return new ShowBiographyQuestionDTO(biographyQuestion.getId(), biographyQuestion.getQuestion(),
46             biographyQuestion.getRightAnswer(), biographyQuestion.getWrongAnswer1(), biographyQuestion.getWrongAnswer2(),
47             biographyQuestion.getWrongAnswer3());
48     }
49     private List<ShowBiographyQuestionDTO> entityToDTOList(final List<BiographyQuestion> biographyQuestionEntities){
50         List<ShowBiographyQuestionDTO> showBiographyQuestionDTOS = new ArrayList<>();
51         for (BiographyQuestion workQuestionEntity : biographyQuestionEntities) {
52             showBiographyQuestionDTOS.add(entityToDTO(workQuestionEntity));
53         }
54         return showBiographyQuestionDTOS;
55     }
56 }
```

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

```
35     constructor(props){
36         super(props);
37         const { navigation } = this.props;
38         const id = navigation.getParam('biographyTestID', 'NO-ID');
39     }
40 }
```

Ще го подадем на заявката:

```
40     getBiographyQuestionsByQuizID(id)
41     .then(
42         (response) => {
43             this.setState({
44                 id: id,
45                 loading: false,
46                 array: this.transformArray(response)
47             });
48         }
49     );
```

и обновяваме състоянието на компонента.

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

```
122     return (
123       <ImageBackground
124         source={require('../../images/Moleskin.png')}
125         style={{width: '100%', height: '100%'},styles.container}}>
126       <ScrollView>
127         { this.renderAnswers(this.state.array,this.state.currentQuestionIndex) }
128       <Button
129         style={{marginTop: 10}}
130         title="Следващия въпрос:"
131         onPress={() => {
132           const newIndex = this.state.currentQuestionIndex+1;
133           this.setState({currentQuestionIndex: newIndex});
134           if(newIndex >= this.state.array.length){
135
136             Alert.alert('Done');
137           }
138         }}
139       />
140     </ScrollView>
141   </ImageBackground>
142 );
```

Ето и функцията, която рендира въпрос по зададен индекс:

```

85   renderAnswers = (questions, questionIndex) =>
86   {
87     questionIndex = Number(questionIndex);
88     const currentQuestion = questions[questionIndex];
89
90     return (
91       <View
92         style={styles.container}>
93         <Text>{currentQuestion.question}</Text>
94         <Button
95           title={currentQuestion.answers[0]}
96           onPress={() => this.state.answers.push(1)}
97         />
98         <Button
99           title={currentQuestion.answers[1]}
100          onPress={() => this.state.answers.push(2)}
101        />
102        <Button
103          title={currentQuestion.answers[2]}
104          onPress={() => this.state.answers.push(3)}
105        />
106        <Button
107          title={currentQuestion.answers[3]}
108          onPress={() => this.state.answers.push(4)}
109        />
110      </View>
111    );
112  };

```

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:

```

36   @PostMapping("/biography-quiz/{quizId}")
37   public Double uploadBiographyResult(@PathVariable("quizId") final Integer biographyQuizId,
38                                     @RequestBody final BiographyQuizAnswerResultDTO answersDTO,
39                                     Principal principal){
40     User user = this.userService.findUserEntityByUserName(principal.getName());
41     BiographyQuiz biographyQuiz = this.biographyQuizService.getBiographyQuizById(biographyQuizId);
42     List<BiographyQuestion> biographyQuestions = biographyQuiz.getBiographyQuestions();
43     return this
44       .biographyResultService
45       .saveResult(answersDTO,biographyQuestions,user,biographyQuiz);
46   }
47 }

```

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.

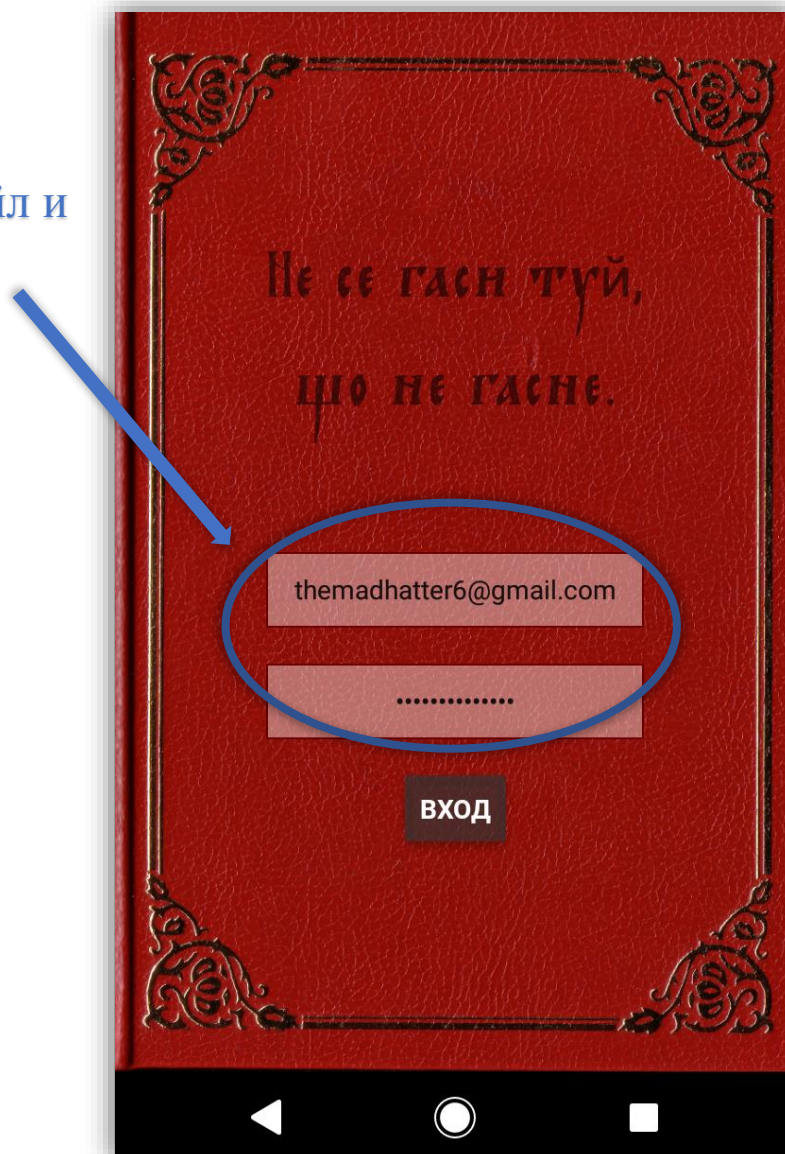
Линк към **GitHub Repository** на проекта:

<https://github.com/ValkaHonda/Literature>

IV. Потребителско ръководство

1) Вход на потребителя:

Влизане с имейл и
парола



2) Главно меню:

Избор на трите
главни опции



3) Тест:



2. Иван Вазов е роден в град:

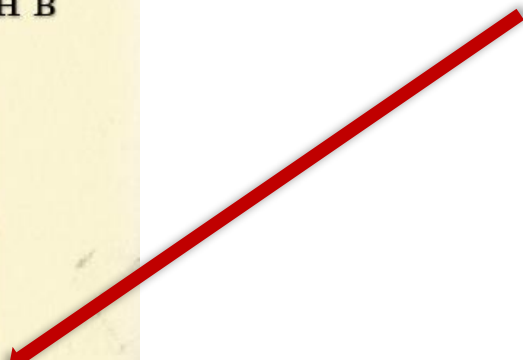
А) Калофер

Б) София

В) Сопот

Г) Карлово

Избор на правилен
отговор



Ранглист (Биографии)



4) Ранглист:

Класация на
потребителите по
резултати от тестове

V. Заключение

1) Готов ли е продуктът за употреба?

Продуктът е в готовност за експлоатация, като предстои базата данни да се напълни с огромни количества от информация. Сървърът е хостнат и всяко устройство може да го достъпва.

Продуктът е Open Source проект, което позволява откриване на бъгове и неточности, които могат да бъдат оправени от нашия екип или от всеки, който иска да допринесе за развитието на приложението.

2) Среща ли нуждите на потребители?

Приложението е насочено към ученици, които изучават българска литература и се подготвят за зрелостни изпити и матури. Предоставя допълнителна функционалност, с помощта на която хора с увредено зрение също имат възможност да се запознаят с българското културно наследство. Освен добър източник на информация, приложението предоставя ефективен начин за упражнение и по-добро усвояване на материала, като потребителите могат да следят своя прогрес и да се сравняват в класация с други хора.

3) Конкурентноспособен ли е на съществуващите решения?

На пазара няма друго приложение, което да е така специфично насочено към българската литература. Голямо предимство е удобната работа с огромни количества анализи и биографии, чрез търсачки и опростен интерфейс. Потребителят се наслаждава на прекрасно изживяване, докато научава повече за родната ни литература.

VI. Източници на информация:

<https://spring.io/docs>

<https://spring.io/projects/spring-security>

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

<https://hibernate.org/orm/documentation/5.4/>

<https://jwt.io>

<https://stackoverflow.com/questions/130794/what-is-dependency-injection>

https://www.w3schools.com/js/js_es6.asp

<https://medium.freecodecamp.org/how-to-use-es6-modules-and-why-theyre-important-a9b20b480773?gi=58ad6eb706ee>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

<https://facebook.github.io/react-native/docs/getting-started>

<https://facebook.github.io/react-native/docs/network>

<https://reactnavigation.org/docs/en/getting-started.html>

<https://www.pgadmin.org/docs/>

<https://www.smartdraw.com/entity-relationship-diagram/>

<https://tallyfy.com/workflow-diagram/>

<http://www.agilemodeling.com/artifacts/userStory.htm>

<https://devcenter.heroku.com/categories/reference>

<https://developer.android.com/studio/command-line/sdkmanager>

<https://stackoverflow.com/questions/34005713/difference-between-clean-gradlew-clean>