



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

**ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И
ТЕХНОЛОГИИ
СПЕЦИАЛНОСТ КОМПЮТЪРНО И
СОФТУЕРНО ИНЖЕНЕРСТВО**

Студент: Валери Ивайлов Райков

Фак. №: 121222139

Група: 42Б

Дисциплина:

Програмиране за мобилни устройства

Тема на проекта:

Мобилно приложение за търсене и преглед на филми и
телевизионни предавания

Съдържание на документацията

1. Увод	3
2. Анализ на съществуващи разработки	3
• Netflix	3
• IMDb	4
• JustWatch	4
• TMDb (The Movie Database)	4
Заключение:	5
3. Проектиране	5
• Целева аудитория:	5
• Използвани данни:	5
• Съхранение на данните:	6
• Бизнес процеси в приложението:	6
• Как ще бъдат достъпени функционалностите от потребителя?	7
• Динамични маршрути	7
• Компоненти за подобряване на UX:	7
• Примерен user flow:	7
4. Реализация и приложение	8
• Services	9
• Hooks	13
• Context	16
• App:	18
• Components	21
• Appwrite backend:	22
5. Потребителско ръководство	25
6. Заключение	37
7. Литература	38

1. Увод

С нарастващия интерес към стрийминг платформите и онлайн съдържанието, потребителите все по-често търсят централно място, където да откриват нови филми и сериали, да следят рейтинги и да получават препоръки. Настоящият проект цели създаване на мобилно приложение, което предоставя на потребителя интуитивен и бърз начин за намиране на филми и телевизионни сериали чрез свързване с външни бази данни. Приложението е универсално и е подходящо за потребители на всякаква възраст.

2. Анализ на съществуващи разработки

В тази секция се разглеждат вече съществуващи приложения със сходна идея и цел, като се анализират техните функционалности, предимства и недостатъци.

• Netflix

Описание: Netflix е една от най-популярните платформи за стрийминг на филми и телевизионни сериали в света. Осигурява достъп до богата колекция от съдържание – както лицензирано, така и оригинално, включително световноизвестни продукции. Приложението е налично за мобилни устройства, уеб и телевизори, и предлага персонализирано потребителско изживяване.

Предимства:

- Огромна библиотека от съдържание – разнообразие от жанрове, държави и езици.
- Оригинално съдържание – Netflix създава собствени сериали и филми, често с високо качество.
- Интелигентни препоръки – базирани на историята на гледане и оценки.
- Мултиплатформена поддръжка – можеш да гледаш на всяко устройство с интернет.
- Възможност за изтегляне – съдържание за офлайн гледане.

Недостатъци:

- Платено съдържание – липса на безплатна версия, което го прави недостъпно за някои потребители.
- Ограничения по държави – не всички филми/сериали са налични навсякъде.
- Понякога твърде много съдържание – трудно е да избереш какво да гледаш.
- Няма потребителски рецензии – липсва възможност за писане на коментари и обсъждане между потребителите.

• **IMDb**

Описание: Официално мобилно приложение за достъп до огромна база данни с филми, актьори и оценки.

Предимства:

- Богата база с информация
- Много потребителски рецензии и рейтинги
- Надежден източник на данни

Недостатъци:

- Интерфейсът е претрупан
- Има реклами
- Няма персонализирани препоръки

• **JustWatch**

Описание: Приложение за търсене на филми и сериали по това къде са налични (Netflix, HBO, Disney+ и др.)

Предимства:

- Отличен филтър по платформи
- Удобен за потребители, които имат много абонаменти

Недостатъци:

- Малко информация за самите филми
- Няма потребителска регистрация или история на гледаното

• **TMDb (The Movie Database)**

Описание: Отворена платформа с богата информация, често използвана от други приложения (**включително и настоящото**)

Предимства:

- API за разработчици
- Добро визуално представяне на заглавията

Недостатъци:

- Понякога непълни данни
- По-слаба поддръжка на нови сериали

Заклучение:

Настоящото приложение е изработено с цел да съчетае възможно най-много от предимствата на горепосочените разработки и същевременно да минимизира техните недостатъци. То предлага собствен потребителски интерфейс и уникална функционалност, които ще бъдат разгледани в следващите секции от документацията.

3. Проектиране

• Целева аудитория:

Приложението е насочено към младежи и възрастни потребители, които обичат киното и търсят бърз достъп до информация за филми и сериали (както български така и чуждестранни).

• Използвани данни:

Входни данни от потребителя: Потребителят въвежда или взаимодейства със следните типове данни:

Функционалност | **Данни, които се въвеждат/генерират**

Регистрация/Вход | Email, Парола, Потребителско име

Запазване на филм | ID на филм, Заглавие, Плакат, Описание

Търсене на филм | Търсене по заглавие

Преглед на детайли|

за филм | Взема се ID от URL параметъра и се използва за fetch

Изход | Сесията се изтрива

Структури на данни (класове и обекти):

Потребител (User):

```
type User = {  
  $id: string;  
  email: string;  
  name: string;  
}
```

Филм (Movie):

```
type Movie = {
```

```

id: number;
title: string;
overview: string;
genres: string[];
poster_path: string;
release_date: string;
runtime: number;
vote_average: number;
vote_count: number;
production_companies: string[];
budget: number;
revenue: number;
}

```

Запазен филм (Saved movie):

```

type SavedMovie = {
  userId: string;
  movieId: number;
  title: string;
  poster_path: string;
  savedAt: string;
}

```

• Съхранение на данните:

Appwrite Database – използва се за съхранение на запазените филми като документи, свързани с конкретен потребител.

Appwrite Account – управлява се сесията и идентичността на потребителя.

TMDB API – използва се за извличане на детайли за филми (няма нужда от локално съхранение).

Състоянието на UI се държи в `useState`, `useEffect` и `custom hooks` като `useFetch`.

• Бизнес процеси в приложението:

Процес	Описание
Регистрация	Създаване на потребител чрез Appwrite, логване на сесия

Вход/Изход	Създаване и изтриване на сесии с Appwrite
Запазване на филм	Изпращане на POST заявка към Appwrite DB
Преглед на филм	Вземане на ID (URL), fetch (TMDB), показване на детайли
Списък със запазени	Вземане от база с филмите на потребителя
Навигация	Чрез табове и динамични маршрути в Expo Router

• Как ще бъдат достъпни функционалностите от потребителя?

Интерфейс на приложението (UI)

Навигацията и структурата на потребителския интерфейс са реализирани с Expo Router (React Navigation) и използват:

Раздел (Tab) | Екран

Home (/)	Начален екран с популярни филми
Search (/search)	Позволява търсене на филми по ключова дума
Saved (/saved)	Показва списък със запазените от потребителя филми
Profile (/profile)	Показва информация за профила и бутон за изход

• Динамични маршрути

`tabs/movies/[id].tsx` - Това е страница с детайли за филм.

ID се взима от `useLocalSearchParams()`, после се `fetch`-ват данните и се визуализират.

• Компоненти за подобряване на UX:

TabIcon – показва икона и име на всеки таб.

MovieCard, MovieInfo – визуализират данни за филм по красив и четим начин.

AlertModal – за показване на съобщения (успех, грешка и т.н.).

GoBack – бутон за връщане към предишната страница.

useModal hook – управлява показването на модалните прозорци.

• Примерен user flow:

1. Потребителят избира език и влиза → отива на "Home"
2. Разглежда филми и TV предавания → цъка на един → отваря се `movies/[id]`
3. Натиска "Save" → филмът се запазва
4. В раздел "Saved" вижда запазените си филми
5. Може да излезе от профила чрез "Profile" tab

4. Реализация и приложение

Използвани технологии:

- **React Native** - framework за изграждане на крос-платформени мобилни приложения с JavaScript / TypeScript и React. Той позволява използването на една кодова база за iOS и Android, като използва нативни компоненти.
- **Expo** - инструмент и набор от библиотеки, които улесняват разработката на React Native приложения, като предоставят готови решения за достъп до хардуерни функции (камера, GPS и др.) и бързо тестване.
- **Tailwind CSS** - е utility-first CSS framework, който позволява бързо стилизиране на уеб приложения чрез предварително дефинирани класове, вместо да се пише ръчен CSS.
- **TypeScript** - строго типизиран superset на JavaScript, който добавя опционални типове и подобрена поддръжка за големи проекти, намалявайки грешките по време на разработка.

Източници на данни:

- **TMDb API** - предоставя безплатен достъп до огромна база данни с филми, сериали и актьори, което го прави идеален за създаване на кино-приложения. Той предлага разнообразни endpoints за търсене, популярни филми, рейтинги и други данни. Може да се използва с REST заявки и има добра документация за интеграция.
- **Appwrite** - open-source backend as a service (BAAS) платформа, която предлага готови решения за потребителска аутентикация, бази данни, файлово хранилище и други cloud функции. Той улеснява разработката на уеб и мобилни приложения, като намалява нуждата от ръчно писане на сложен backend код. Поддържа множество SDK-та за различни езици и frameworks, включително JavaScript, Flutter и React Native.

Контекст:

Контекстът в React/React Native е механизъм за споделяне на данни между компоненти без необходимост от пропърти-드릴ване (props drilling). Той е полезен за глобални стойности като тема, потребителски данни, език и др. `useContext` е React Hook, който позволява лесен достъп до стойностите на контекста във функционални компоненти.

- **AuthContext.tsx / useAuth()** - използван за управление на състоянието на потребителя
- **SavedMoviesContext.tsx / useSavedMovies()** - използван за управление на запазените от потребител филми

Динамично зареждане:

- Използване на персонализиран React Hook (useFetch), който опростява извличането на данни от API или асинхронни операции.

По-интересни примери от кода:

- **Services**

api.ts: Този код е част от услуга за достъп до външен API – The Movie Database (TMDB). Основната му функция е да извлича филми по зададен текст за търсене или да връща популярни филми по подразбиране.

Съдържа:

- **Конфигурация на API (ред 3 -10)** - конфигурирано веднъж с възможност за преизползване!
- **Функция за език (ред 12–14)** - взима текущия език от i18n библиотеката
- **Функция за извличане на филми (ред 17–36) - Параметър:** обект със свойство query, което е текстът за търсене. Ако query е зададен – се търсят филми с това име. Ако няма query, се извличат популярни филми. Избира подходящия краен URL за заявката: Ако има търсене (query) – използва **/search/movie**. Ако няма – използва **/discover/movie**, сортирани по популярност (**ред 20-22**). След това се изпраща GET заявка към избрания endpoint (**ред 24-27**). Следва обработка на грешки и проверка за статус кода на получения response (**ред 29-32**). При проблем се хвърля грешка с текст за грешка, а при успех се връщат извлечените от API филми.

```

1  import i18n from '@services/i18n';
2
3  export const TMDB_CONFIG = {
4    BASE_URL: 'https://api.themoviedb.org/3',
5    API_KEY: process.env.EXPO_PUBLIC_MOVIE_API_KEY,
6    headers: {
7      accept: 'application/json',
8      Authorization: `Bearer ${process.env.EXPO_PUBLIC_MOVIE_API_KEY}` || '',
9    },
10  };
11
12  const getLanguageParam = (): string => {
13    return i18n.language === 'bg' ? 'bg-BG' : 'en-US';
14  }
15
16  // Movie functions
17  export const fetchMovies = async ({ query }: { query: string }) => {
18    const lang = getLanguageParam();
19
20    const endpoint = query
21      ? `${TMDB_CONFIG.BASE_URL}/search/movie?query=${encodeURIComponent(query)}&language=${lang}`
22      : `${TMDB_CONFIG.BASE_URL}/discover/movie?sort_by=popularity.desc&language=${lang}`;
23
24    const response = await fetch(endpoint, {
25      method: 'GET',
26      headers: TMDB_CONFIG.headers,
27    });
28
29    if (!response.ok) {
30      // @ts-ignore
31      throw new Error(`Failed to fetch movies. ${response.statusText}`);
32    }
33
34    const data = await response.json();
35
36    return data.results || [];
37  }

```

Следващата функция е за извличане на детайлна информация за конкретен филм от TMDB API, използвайки ID на филма. Иползва се обработка за грешки (try - catch). Отново се изгражда и изпраща заявка към API, проверява се дали response е валиден и при успех се преобразува в JSON формат и се връща:

```

39  export const fetchMovieDetails = async (movieId: string): Promise<MovieDetails> => {
40    const lang = getLanguageParam();
41
42    try {
43      const response = await fetch(`${TMDB_CONFIG.BASE_URL}/movie/${movieId}?api_key=${TMDB_CONFIG.API_KEY}&language=${lang}`, {
44        method: 'GET',
45        headers: TMDB_CONFIG.headers,
46      });
47
48      if (!response.ok)
49        throw new Error("Failed to fetch movie details!");
50
51      const data = await response.json();
52
53      return data;
54    } catch (err) {
55      console.log(err);
56      throw err;
57    }
58  }

```

Останалите функции от този файл са със сходна идея - извличане на български филми и след това функции за извличане на TV предавания. Идеята и кодът са сходни и затова няма да ги разглеждам по отделно.

- **appwrite.ts:** този файл съдържа функции за работа с Appwrite (backend платформа) в React Native приложение. Той използва Appwrite SDK за взаимодействие с база данни и система за потребителска аутентикация. Тук се инициализира appwrite client към текущия проект (ред 7-9) и се създават инстанции за достъп до базата и акаунтите (ред 11-12). Извършват се следните функции:
- Запазва филми в "любими".
- Показва тенденции в търсенето.
- Управлява потребителски сесии.

Функция: updateSearchCount: записва или актуализира информация в база данни за това колко пъти даден филм е бил търсен по зададен низ (query). Проверява дали вече съществува запис за търсенето (query) в базата данни, като използва **Query.equal**. Ако записът съществува, увеличава броя търсения с едно, иначе създава нов запис

```

3  const DATABASE_ID = process.env.EXPO_PUBLIC_APPWRITE_DATABASE_ID!;
4  const COLLECTION_ID = process.env.EXPO_PUBLIC_APPWRITE_COLLECTION_ID!;
5  const SAVED_COLLECTION_ID = process.env.EXPO_PUBLIC_APPWRITE_SAVED_COLLECTION_ID!;
6
7  const client = new Client()
8    .setEndpoint('https://cloud.appwrite.io/v1')
9    .setProject(process.env.EXPO_PUBLIC_APPWRITE_PROJECT_ID!);
10
11  const database = new Databases(client);
12  const account = new Account(client);
13
14  export const updateSearchCount = async (query: string, movie: Movie) => {
15    try {
16      const result = await database.listDocuments(DATABASE_ID, COLLECTION_ID, [
17        Query.equal('searchTerm', query),
18      ]);
19
20      if (result.documents.length > 0) {
21        const existingMovie = result.documents[0];
22
23        await database.updateDocument(DATABASE_ID, COLLECTION_ID, existingMovie.$id, {
24          count: existingMovie.count + 1,
25        });
26      } else {
27        await database.createDocument(DATABASE_ID, COLLECTION_ID, ID.unique(), {
28          searchTerm: query,
29          movie_id: movie.id,
30          count: 1,
31          title: movie.title,
32          poster_url: `https://image.tmdb.org/t/p/w500${movie.poster_path}`,
33        });
34      }
35    } catch (err) {
36      console.log(err);
37      throw err;
38    }
39  }

```

Функция: getTrendingMovies: Тази функция извлича списък с "трендови" (най-популярни) филми от базата данни на Appwrite. Използва се за показване на "Най-търсените филми днес" в началния екран.

```

41 export const getTrendingMovies = async (): Promise<TrendingMovie[] | undefined> => {
42   try {
43     const result = await database.listDocuments(DATABASE_ID, COLLECTION_ID, [
44       Query.limit(5),
45       Query.orderDesc('count'),
46     ]);
47
48     return result.documents as unknown as TrendingMovie[];
49   } catch (err) {}
50   console.log(err);
51   return undefined;
52 }
53

```

Функция saveMovie: Запазва филм в колекцията "запазени филми", ако вече не е запазен. Проверява дали филмът вече е запазен чрез **checkIfMovieSaved(movie.id)**. Ако не е запазен, създава нов документ в колекцията **SAVED_COLLECTION_ID**

Функция getSavedMovies: Връща всички запазени филми от базата данни.

```

68 export const saveMovie = async (movie: Movie) => {
69   try {
70     const alreadySaved = await checkIfMovieSaved(movie.id);
71
72     if (!alreadySaved) {
73       await database.createDocument(DATABASE_ID, SAVED_COLLECTION_ID, ID.unique(), {
74         movie_id: movie.id,
75         title: movie.title,
76         poster_url: movie.poster_path
77           ? `https://image.tmdb.org/t/p/w500${movie.poster_path}`
78           : null,
79       });
80     } else {
81       return;
82     }
83   } catch (err) {
84     console.error(err);
85     throw err;
86   }
87 };
88
89 export const getSavedMovies = async (): Promise<AppwriteMovie[]> => {
90   try {
91     const response = await database.listDocuments(DATABASE_ID, SAVED_COLLECTION_ID);
92     return response.documents as unknown as AppwriteMovie[];
93   } catch (err) {
94     console.error(err);
95     return [];
96   }
97 };

```

Следващите 2 функции са насочени към аутентикацията в приложението:

Функция: signUp: Създава нов потребител и го вписва автоматично (log in) след регистрация. В началото изтрива текущата сесия, ако има такава, за да се избегнат конфликти с вече активен потребител. След това създава нов потребител и автоматично го вписва:

Функция: login: Вписва съществуващ потребител в приложението. Отново изтрива текущата сесия, ако има такава, след това създава нова сесия за

вписване, ако email и паролата са коректни. Накрая връща информация за текущия потребител:

```
135 export const signUp = async (email: string, password: string, username: string) => {
136   try {
137     try {
138       await account.deleteSession('current');
139     } catch (deleteError) {}
140
141     const user = await account.create(
142       ID.unique(),
143       email,
144       password,
145       username
146     );
147
148     await account.createEmailPasswordSession(email, password);
149     return user;
150   } catch (err) {
151     console.error('Signup error:', err);
152     throw err;
153   }
154 }
155
156 export const login = async (email: string, password: string) => {
157   try {
158     try {
159       await account.deleteSession('current');
160     } catch (deleteError) {}
161
162     await account.createEmailPasswordSession(email, password);
163     return await account.get();
164   } catch (err) {
165     console.error('Login error:', err);
166     throw err;
167   }
168 }
```

и т.н.

• Hooks

- **useFetch.ts:** потребителски (custom) React hook, наречен useFetch. Служи за централизирано и многократно използване на логика за извличане на данни (fetch). Подобна логика често се повтаря, затова създаването на такъв хук прави кода по-четим и лесен за поддръжка.

useFetch<T> приема функция, която прави заявка към API (или друга асинхронна операция) и връща:

data: резултат от заявката

loading: дали се зарежда в момента

error: евентуална грешка при извличане

refetch: функция за повторно извикване

reset: функция за нулиране на състоянието

Аргументи:

useState hooks: Съхраняват данните, състоянието на зареждане и евентуална грешка.

fetchData функцията: Изпълнява fetchFunction, задава loading = true, записва резултатите или грешка.

reset функцията: Изчиства всичко – удобно при повторни заявки или смяна на страници.

useEffect hook: Автоматично стартира заявката при първоначално зареждане, ако autoFetch = true.

Връща:

```
return {  
  data,    // резултат от заявката  
  loading, // дали е в процес на зареждане  
  error,   // ако има грешка  
  refetch: fetchData, // функция за презареждане  
  reset    // функция за нулиране  
};
```



```

3  const useFetch = <T>(fetchFunction: () => Promise<T>, autoFetch = true) => {
4      const [data, setData] = useState<T | null>(null);
5      const [loading, setLoading] = useState(false);
6      const [error, setError] = useState<Error | null>(null);
7
8      const fetchData = async () => {
9          try {
10             setLoading(true);
11             setError(null);
12
13             const result = await fetchFunction();
14
15             setData(result);
16         } catch (err) {
17             // @ts-ignore
18             setError(err instanceof Error ? err : new Error("An error occurred!"));
19         } finally {
20             setLoading(false);
21         }
22     }
23
24     const reset = () => {
25         setData(null);
26         setLoading(false);
27         setError(null);
28     }
29
30     useEffect(() => {
31         if (autoFetch)
32             fetchData();
33     }, []);
34
35     return { data, loading, error, refetch: fetchData, reset };
36 }
37
38 export default useFetch;

```

- **useModal.ts:** Този код дефинира потребителски React hook (custom hook), наречен useModal, който предоставя централизирана логика за управление на модални прозорци (modal windows) в потребителския интерфейс. Той управлява:
 - дали модалът е видим
 - заглавие на модала
 - съобщение
 - тип (икона/цвят) на модала – например info, success, warning, error

```

1  import { useState } from 'react';
2
3  type ModalType = 'info' | 'success' | 'warning' | 'error';
4
5  Tabnine | Edit | Explain
6  const useModal = () => {
7    const [modalVisible, setModalVisible] = useState(false);
8    const [modalTitle, setModalTitle] = useState('');
9    const [modalMessage, setModalMessage] = useState('');
10   const [modalType, setModalType] = useState<ModalType>('info');
11
12   const showModal = (title: string, message: string, type: ModalType = 'info') => {
13     setModalTitle(title);
14     setModalMessage(message);
15     setModalType(type);
16     setModalVisible(true);
17   }
18
19   const hideModal = () => {
20     setModalVisible(false);
21   }
22
23   return {
24     modalVisible,
25     modalTitle,
26     modalMessage,
27     modalType,
28     showModal,
29     hideModal,
30   };
31 }
32 export default useModal

```

• Context

Context API позволява споделяне на стойности между компоненти без нужда от пропсинг надолу по дървото. Използва се за:

- удостоверяване (auth)
- настройки (theme, език)
- глобални данни (напр. филми, потребител)
- **AuthContext.tsx** – Контекст за удостоверяване -> Държи информация за текущия потребител, статуса на зареждане и дали е автентикиран. При зареждане на приложението, проверява дали има влязъл потребител, като използва `getCurrentUser()` от Appwrite API. Ако има такъв – записва го в `user`, иначе – `null`. Подава контекста и децата на този провайдер имат достъп до всички стойности чрез **`useAuth()` hook**.


```

6   export const AuthProvider = ({ children }: { children: React.ReactNode }) => {
7     const [user, setUser] = useState<any>(null);
8     const [loading, setLoading] = useState(true);
9
10    useEffect(() => {
11      const loadUser = async () => {
12        try {
13          const currentUser = await getCurrentUser();
14          setUser(currentUser);
15        } catch (error) {}
16        setUser(null);
17      } finally {
18        setLoading(false);
19      }
20    };
21
22    loadUser();
23  }, []);
24
25  const value = {
26    user,
27    setUser,
28    logout,
29    loading,
30    isAuthenticated: !!user,
31  };
32
33  return (
34    <AuthContext.Provider value={value}>
35      {!loading && children}
36    </AuthContext.Provider>
37  );
38 }
39
40 export const useAuth = () => useContext(AuthContext);

```

- **SavedMoviesContext.tsx** – Контекст за запазени филми -> Управлява списък със запазени филми на потребителя. Използва useFetch хук, за да зареди филмите от getSavedMovies. Съхранява:
 - savedMovies – масив с филми
 - loading - зареждане
 - error – грешка
 - refreshSavedMovies – функция за повторно зареждане

```

1 import React, { createContext, useContext } from 'react';
2 import useFetch from '@/hooks/useFetch';
3 import { getSavedMovies } from '@/services/appwrite';
4
5 const SavedMoviesContext = createContext<any>(null);
6
7 Tabnine | Edit | Explain
8 export const SavedMoviesProvider = ({ children }: { children: React.ReactNode }) => {
9   const {
10     data: savedMovies = [],
11     loading,
12     error,
13     refetch: refreshSavedMovies,
14   } = useFetch(getSavedMovies);
15
16   return (
17     <SavedMoviesContext.Provider value={{ savedMovies, loading, error, refreshSavedMovies }}>
18       {children}
19     </SavedMoviesContext.Provider>
20   )
21 }
22
23 export const useSavedMovies = () => useContext(SavedMoviesContext);

```

• App:

Това е мястото, където се структурира логиката на маршрутизацията (routing), страниците (pages) и сървърни/клиентски компоненти.

- **_layout.tsx**: Това е основен шаблон за всички страници. Използва се като "рамка", в която се вграждат различните page.tsx.
- **(auth)/**: сегмент от маршрути, свързани с потребителската автентикация. Скобите () в Next.js и Expo Router означават "групиране без URL сегмент", т.е. те:
 - групират файлове и папки логически, но
 - не влияят на пътя (route), който се вижда в брауъра или приложението.

Тук се намират **index.tsx**, **login.tsx**, **signup.tsx** -> От тук се управляват:

- **Регистрация** – потребител въвежда email, парола и име.
- **Вход** – създаване на сесия и извличане на текущ потребител
- Преминане към основното приложение след логин
- **(tabs)/**: съдържа всички екрани, които се показват като част от таб-базирана навигация (табове долу в приложението). Това е част от т.нар. layout-based routing. Позволява на потребителя да преминава между различни екрани като „Home“, „Search“, „Saved“ и „Profile“.

Тук се намират **_layout.tsx**, **index.tsx**, **profile.tsx**, **saved.tsx**, **search.tsx** (Описани са малко по-надолу):

_layout.tsx:

```
7  const AuthAwareLayout = () => {
8    const { user, loading } = useAuth();
9
10   if (loading)
11     return null;
12
13   return (
14     <Stack screenOptions={{
15       headerShown: false,
16     }}>
17       {user ? (
18         <>
19           <Stack.Screen
20             name="(tabs)"
21             options={{ headerShown: false }}
22           />
23           <Stack.Screen
24             name="bg-movies"
25             options={{ headerShown: false }}
26           />
27           <Stack.Screen
28             name="movies/[id]"
29             options={{ headerShown: false }}
30           />
31         </>
32       ) : (
33         <Stack.Screen
34           name="(auth)"
35           options={{ headerShown: false }}
36         />
37       )}
38     </Stack>
39   )
40 }
```

(auth)/login.tsx:

```
12  const loginScreen = () => {
13    const [email, setEmail] = useState('');
14    const [password, setPassword] = useState('');
15    const [loading, setLoading] = useState(false);
16    const router = useRouter();
17
18    const {
19      modalVisible,
20      modalTitle,
21      modalMessage,
22      modalType,
23      showModal,
24      hideModal,
25    } = useModal();
26
27    const { t } = useTranslation();
```

(auth)/signup.tsx:

```
31  const handleSignup = async () => {
32    if (!username || !email || !password || !confirmPassword) {
33      showModal(t('Error'), t('Please fill in all required fields'), 'error');
34      return;
35    }
36
37    if (password !== confirmPassword) {
38      showModal(t('Error'), t('Passwords do not match'), 'error');
39      return;
40    }
41
42    setLoading(true);
43
44    try {
45      await signUp(email, password, username);
46      router.replace('(tabs)');
47    } catch (err) {
48      console.error(err);
49      showModal(t('Error'), t('Signup failed. Please try again.'), 'error');
50    } finally {
51      setLoading(false);
52    }
53  };
```

(tabs)/_layout.tsx:

```

10     return (
11       <Tabs screenOptions={{
12         tabBarShowLabel: false,
13         tabBarItemStyle: {
14           width: '100%',
15           height: '100%',
16           justifyContent: 'center',
17           alignItems: 'center',
18         },
19         tabBarStyle: {
20           backgroundColor: '#0f0d23',
21           borderRadius: 50,
22           marginHorizontal: 20,
23           marginBottom: 36,
24           height: 52,
25           position: 'absolute',
26           overflow: 'hidden',
27           borderWidth: 1,
28           borderColor: '#0f0d23',
29         }
30       }}>
31         <Tabs.Screen
32           name="index"
33           options={{
34             title: "Home",
35             headerShown: false,
36             tabBarIcon: ({ focused }) => (
37               <TabIcon
38                 focused={focused}
39                 icon={icons.home}
40                 title={t('Home')}
41               />
42             ),
43           }}
44       />
45       <Tabs.Screen
46         name="search"
47         options={{
48           title: "Search",
49           headerShown: false,
50           tabBarIcon: ({ focused }) => (
51             <TabIcon
52               focused={focused}
53               icon={icons.search}
54               title={t('Search')}
55             />
56           ),
57         }}
58       />
59       <Tabs.Screen
60         name="saved"
61         options={{
62           title: "Saved",
63           headerShown: false,
64           tabBarIcon: ({ focused }) => (
65             <TabIcon
66               focused={focused}
67               icon={icons.save}
68               title={t('Saved')}
69             />
70           ),
71         }}
72       />
73       <Tabs.Screen
74         name="profile"
75         options={{
76           title: "Profile",
77           headerShown: false,
78           tabBarIcon: ({ focused }) => (
79             <TabIcon
80               focused={focused}

```

Index.tsx -> по-интересни части от кода:

```

19   const router = useRouter();
20   const { user } = useAuth();
21
22   const {
23     data: trendingMovies,
24     loading: trendingLoading,
25     error: trendingError,
26   } = useFetch(getTrendingMovies);
27
28   const {
29     data: movies,
30     loading: moviesLoading,
31     error: moviesError,
32   } = useFetch(() => fetchMovies({ query: '' }));
33
34   const {
35     data: tvShows,
36     loading: tvShowsLoading,
37     error: tvShowsError,
38   } = useFetch(() => fetchTVShows({ query: '' }));
39
40   const [showTVSeries, setShowTVSeries] = useState(false);
41
42   const {
43     modalVisible,
44     modalTitle,
45     modalMessage,
46     modalType,
47     showModal,
48     hideModal,
49   } = useModal();
50
51   const { t } = useTranslation();

```

```

(showTVSeries && (
  tvShowsLoading ? (
    <ActivityIndicator size="large" color="#0000ff" className="mt-10 self-center" />
  ) : tvShowsError ? (
    <Text>{t('Error')}: {tvShowsError?.message}</Text>
  ) : (
    <View className="flex-1 mt-5">
      <Text className="text-lg text-white font-bold mb-3">{t('Trending TV Shows')}</Text>
      <FlatList
        data={tvShows}
        renderItem={({ item }) => (
          <TVSeriesCard
            id={item.id}
            poster_path={item.poster_path}
            name={item.name}
            vote_average={item.vote_average}
            first_air_date={item.first_air_date}
          />
        )}
        keyExtractor={(item) => item.id.toString()}
        numColumns={3}
        columnWrapperStyle={{
          justifyContent: 'flex-start',
          gap: 20,
          paddingRight: 5,
          marginBottom: 10,
        }}
        className="mt-2 pb-10"
        scrollEnabled={false}
      />
    </View>
  )
)

```

Останалите tab страници са с идентична логика и функционалност спрямо конкретната нужда, затова няма да ги разглеждам по отделно.

- **movies/[id].tsx**: представлява детайлната страница за даден филм. Тя се показва, когато потребителят избере конкретен филм от списъка.
- **useLocalSearchParams()** извлича id от URL-а (пример: /movies/1234 → id = 1234).

- **useFetch()** извиква API-то за филма.
- **useAuth()** проверява дали потребителят е логнат.
- **useModal()** управлява състоянието на модал прозорци.

```

15 const MovieDetails = () => {
16   const { id } = useLocalSearchParams();
17   const { user } = useAuth();
18
19   const [
20     data: movie,
21     loading,
22     error,
23   ] = useFetch(() => fetchMovieDetails(id as string));
24   const [saved, setSaved] = useState(false);
25
26   const {
27     modalVisible,
28     modalTitle,
29     modalMessage,
30     modalType,
31     showModal,
32     hideModal,
33   } = useModal();
34
35   const { t } = useTranslation();
36
37   useEffect(() => {
38     const checkSaved = async () => {
39       if (!movie)
40         return;
41
42       try {
43         const isSaved = await checkIfMovieSaved(movie.id);
44         setSaved(isSaved);
45       } catch (err) {
46         console.error(err);
47       }
48     };
49
50     checkSaved();
51   }, [movie]);

```

• Components

Това са модули от код, които представляват части от потребителския интерфейс (UI). Всеки компонент може да бъде независимо дефиниран и използван многократно. Позволяват:

- Повторно използване
- По-добра структура и четимост на кода
- По-лесна поддръжка
- Изолация на логика (например: модал, таб, карта, форма и т.н.)
- **MovieCard.tsx**: Отделен компонент, който представлява елемента, визуализиращ картичката с филма на различни места от приложението.


```

7  const MovieCard = ({ id, poster_path, title, vote_average, release_date }: Movie) => {
8    const { t } = useTranslation();
9
10   return (
11     <Link href={`/${movies}/${id}`} asChild>
12       <TouchableOpacity className='w-[30%]'>
13         <Image
14           source={{
15             uri: poster_path
16               ? `https://image.tmdb.org/t/p/w500${poster_path}`
17               : 'https://placeholder.co/600x400/1a1a1a/ffffff.png',
18           }}
19           className='w-full h-52 rounded-lg'
20           resizeMode='cover'
21         />
22
23         <Text className='text-sm font-bold ■ text-white mt-2' numberOfLines={1}>{title}</Text>
24
25         <View className='flex-row items-center justify-start gap-x-1'>
26           {Array.from({ length: Math.round(vote_average / 2) }).map((_, index) => (
27             <Image key={index} source={icons.star} className='size-4' />
28           ))}
29           <Text className='text-sm ■ text-white font-bold uppercase'>{Math.round(vote_average / 2)}</Text>
30         </View>
31
32         <View className='flex-row items-center justify-between'>
33           <Text className='text-xs ■ text-light-300 font-medium mt-1'>
34             {release_date?.split('-')[0]}
35           </Text>
36           <Text className='text-sm font-medium ■ text-light-300'>
37             {t('Movie')}
38           </Text>
39         </View>
40       </TouchableOpacity>
41     </Link>
42   )
43 }

```

- **SearchBar.tsx:** Търсачката в приложението.

```

12  const SearchBar = ({ onPress, placeholder, value, onChangeText }: Props) => {
13    return (
14      <View className='flex-row items-center □ bg-dark-200 rounded-full px-5 py-4'>
15        <Image
16          source={icons.search}
17          className='size-5'
18          resizeMode='contain'
19          tintColor='#ab8bff'
20        />
21        <TextInput
22          onPress={onPress}
23          placeholder={placeholder}
24          value={value}
25          onChangeText={onChangeText}
26          placeholderTextColor='#a8b5db'
27          className='flex-1 ml-2 ■ text-white'
28        />
29      </View>
30    )
31  }

```

Това са по-интересните части от кода на приложението. Останалото ще се обясни на място при защитата в случай на възникнали въпроси.

- **Appwrite backend:**

В конкретното приложение се използва за следните функционалности:

- **Аутентикация (Authentication):** Позволява на потребителите да:

- Се регистрират (signUp функцията)
- Влизат в профила си (login)
- Излизат от профила си (logout)
- Получават текущия си профил (getCurrentUser)







Използва се account обект от Appwrite SDK, за да се създават и управляват сесии на потребителите.

- **Запазване на филми (Database):** Потребителите могат да запазват филми в свой списък чрез:

- saveMovie – записва филм в Appwrite база данни
- getSavedMovies – взема списък със запазени филми
- checkIfMovieSaved – проверява дали даден филм вече е запазен

Това става чрез използване на базата данни (Database service) на Appwrite – ти създаваш записи (документи) в колекция, където съхраняваш информация за всеки филм и кой потребител го е запазил.

- **Appwrite/auth** - потребителите, регистрирани (влезли) в приложението:

NAME	IDENTIFIERS	STATUS	ID	LABELS	JOINED	LAST ACTIVITY
 MeggiePhil	meggieph@gmail.com	unverified	 User ID		Apr 11, 2025, 10:44	Apr 11, 2025
 Valeri_R	valeryraikov2@gmail...	unverified	 User ID		Apr 11, 2025, 10:22	Apr 14, 2025
 Valery	valeryraikov@gmail.c...	unverified	 User ID		Apr 10, 2025, 10:17	Apr 10, 2025

- **Създадените колекции:**

<input type="checkbox"/>	COLLECTION ID	NAME	CREATED	UPDATED
<input type="checkbox"/>	67f0d7d10001a9042383	saved_movies	Apr 5, 2025, 10:12	Apr 5, 2025, 10:52
<input type="checkbox"/>	67ed5689000aac10cbb6	metrics	Apr 2, 2025, 18:23	Apr 2, 2025, 18:26

- **saved_movies** колекция:

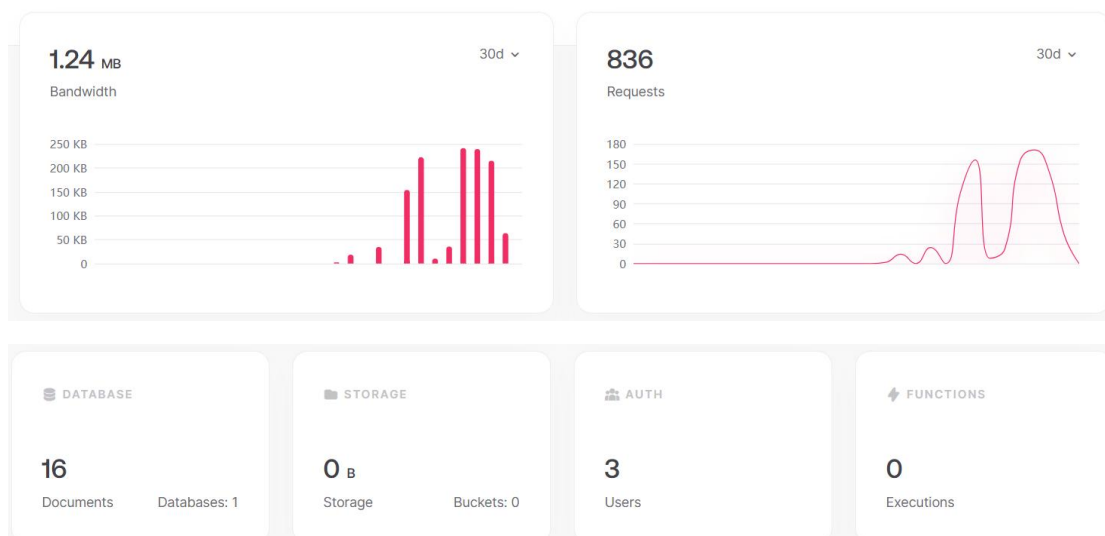
<input type="checkbox"/>	Document ID	poster_url	movie_id	title	Created
<input type="checkbox"/>	67f8d07...b41c73e	https://image.tmdb.org/t/p/w500/6FRFlogh3zFnVWn7Z6zcYnlbRcX.jpg	1197306	A Working Man	Apr 11, 2025, 11:19
<input type="checkbox"/>	67f41dd...d564cf8	https://image.tmdb.org/t/p/w500/z42kyMizgDKetejusZgucSQsQ4I.jpg	136797	Need for Speed	Apr 7, 2025, 21:47
<input type="checkbox"/>	67f406e...d652836	https://image.tmdb.org/t/p/w500/4wkpglgZMYpiSMdThgdqS1TSQc.jpg	1195430	Deva	Apr 7, 2025, 20:09
<input type="checkbox"/>	67f0e2a...1e5c277	https://image.tmdb.org/t/p/w500/nUI80er0ouSDgupOHaaZAntjUOK.jpg	818893	Infiltration	Apr 5, 2025, 10:58
<input type="checkbox"/>	67f0e14...320e988	https://image.tmdb.org/t/p/w500/1bhlezUxvLM9r66yIf1i6EDVJ6R.jpg	1229730	Carjackers	Apr 5, 2025, 10:52

- metrics колекция - показва колко пъти даден филм е търсен от потребителите и се използва за извличане на набиращите популярност филми (те се определят именно от броя търсения за всеки филм!)

<input type="checkbox"/>	Document ID	searchTerm	count	poster_url	movie_id	title
<input type="checkbox"/>	67f4cd4...91ad131	It	3	https://image.tmdb.org/t/p/w500/9E2y5Q7WICVNEhP5GiVTJhEhx1o...	346364	It
<input type="checkbox"/>	67f41dd...6063945	Need for speed	1	https://image.tmdb.org/t/p/w500/z42kyMizgDKetejusZgucSQsQ4I.j...	136797	Need
<input type="checkbox"/>	67f0d6a...82b9d80	Fast	1	https://image.tmdb.org/t/p/w500null	556682	Fast
<input type="checkbox"/>	67ee97d...0a21993	В сърцето на машината	1	https://image.tmdb.org/t/p/w500/rgX8xO4QMGMDTixCa66Y4zttp...	938436	In the
<input type="checkbox"/>	67ee97d...212939c	В сърцето	1	https://image.tmdb.org/t/p/w500/rgX8xO4QMGMDTixCa66Y4zttp...	938436	In the
<input type="checkbox"/>	67ee97d...1e8d751	В сърц	1	https://image.tmdb.org/t/p/w500/nUI80er0ouSDgupOHaaZAntjUOK...	818893	Infiltr

- Workflow диаграми и анализ на данните за заявки към сървиса:

movieApp 67ecfc27002ff2beeebc

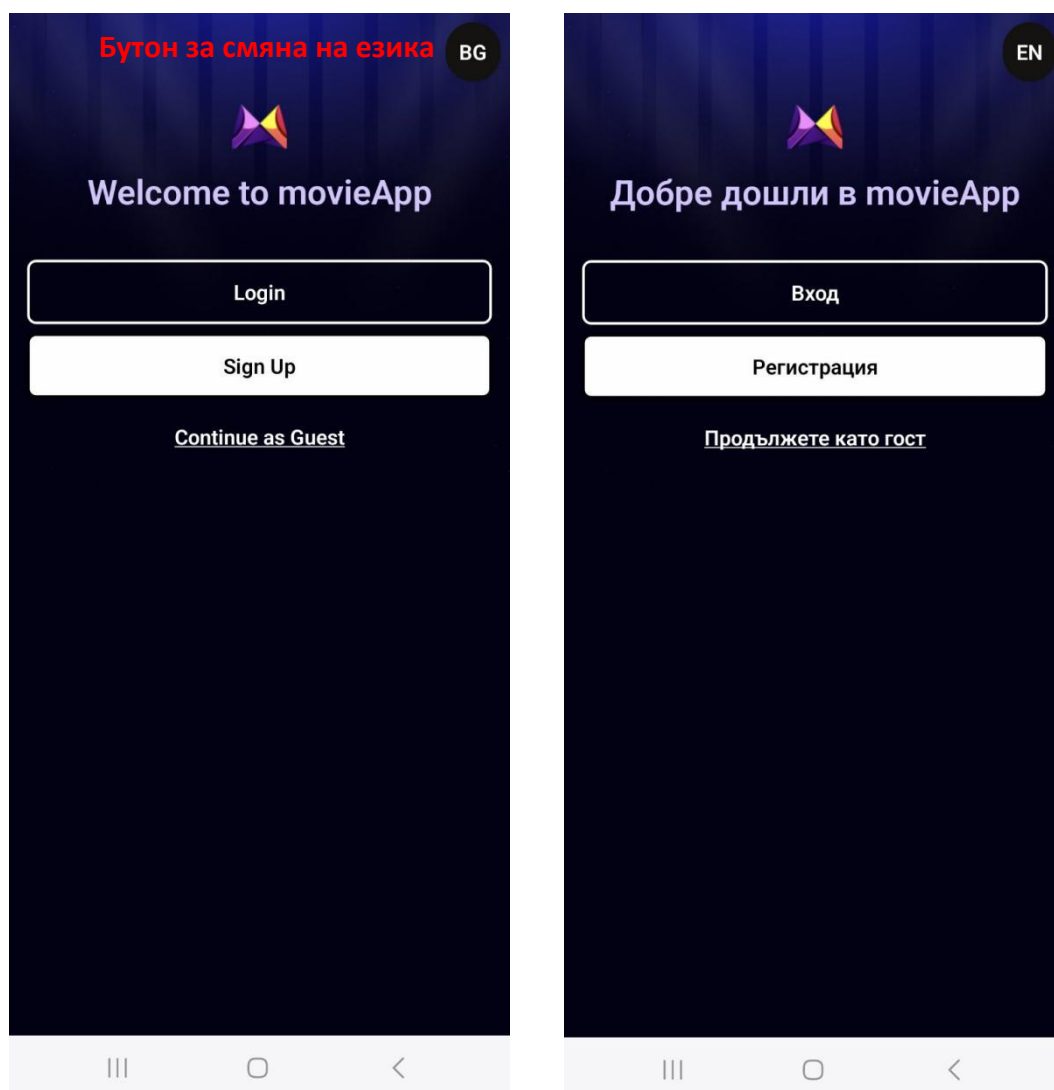


5. Потребителско ръководство

В тази секция се разглежда самото приложение и как Вие потребителите да го ползвате правилно.

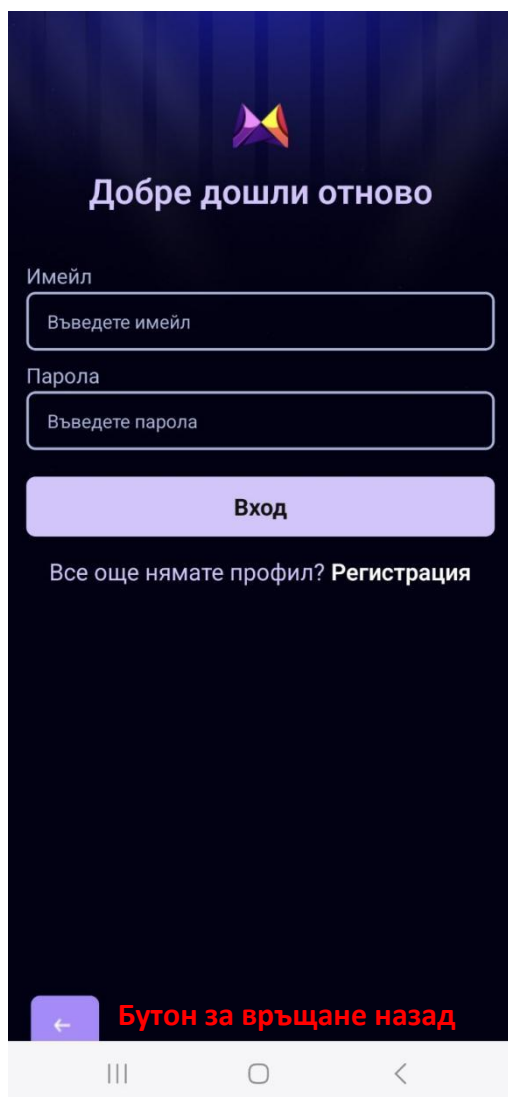
1. Потребителят отваря приложението:

При първоначално стартиране на приложението потребителят вижда следния екран: По подразбиране езикът на приложението е настроен на английски, но за наше улеснение ще го сменим на български (след като веднъж изберем езика, цялото приложение се превежда автоматично):



Потребителят има следните 3 възможности, за да продължи напред :

- Вход (за вече съществуващ профил)
- Регистрация (създаване на нов профил)
- Продължете като гост (използване на приложението без регистрация, но с ограничени възможности):



Добре дошли отново

Имейл

Въведете имейл

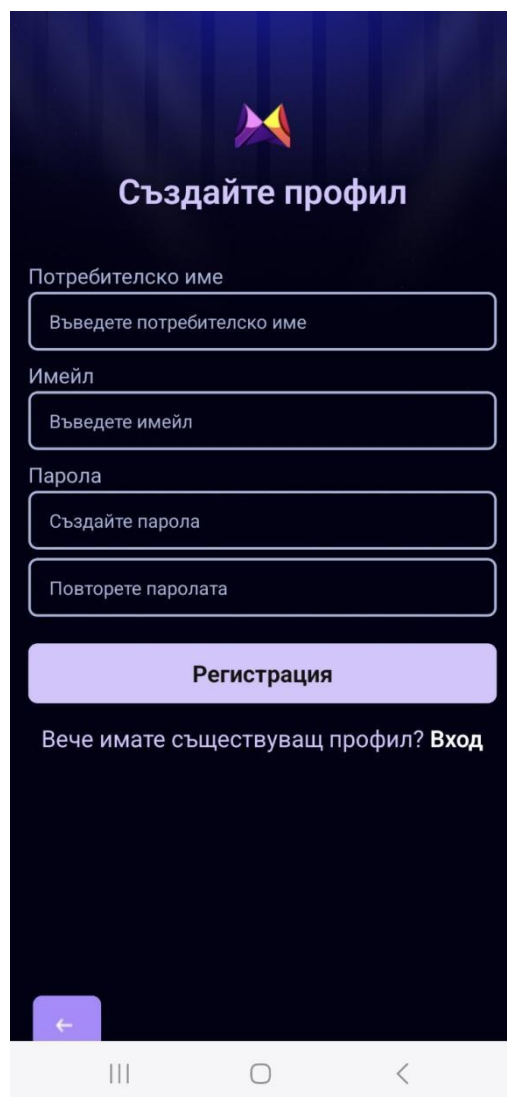
Парола

Въведете парола

Вход

Все още нямате профил? [Регистрация](#)

← Бутон за връщане назад



Създайте профил

Потребителско име

Въведете потребителско име

Имейл

Въведете имейл

Парола

Създайте парола

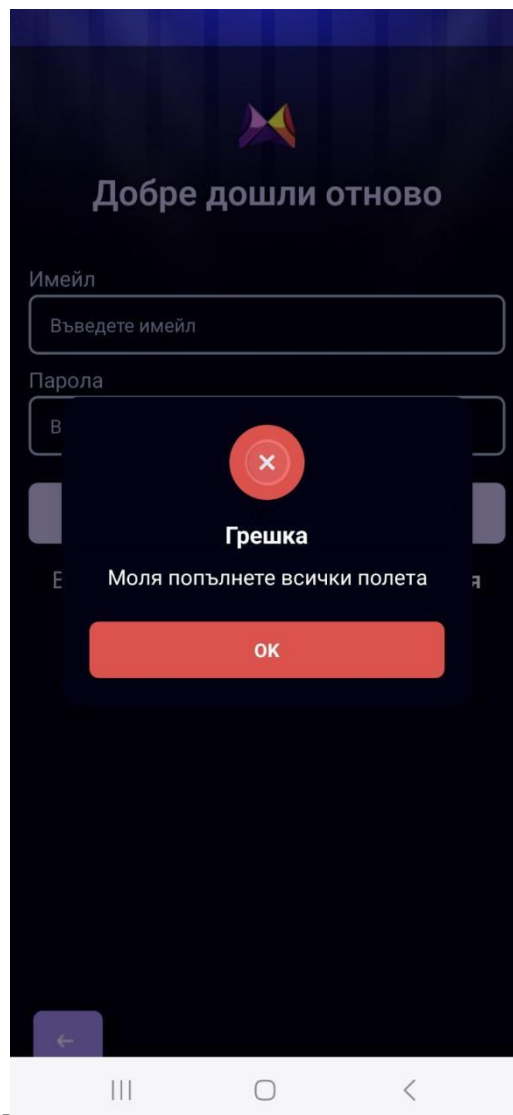
Повторете паролата

Регистрация

Вече имате съществуващ профил? [Вход](#)

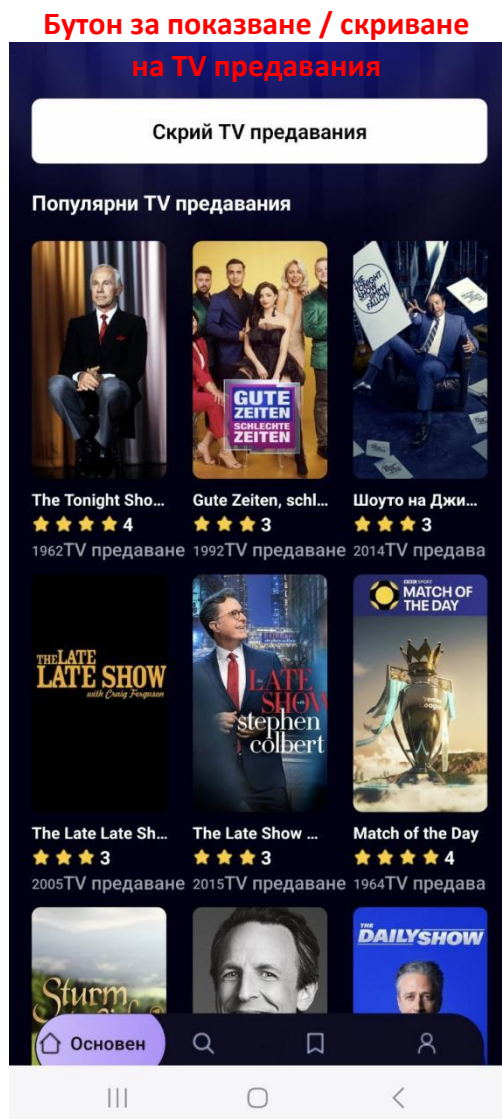
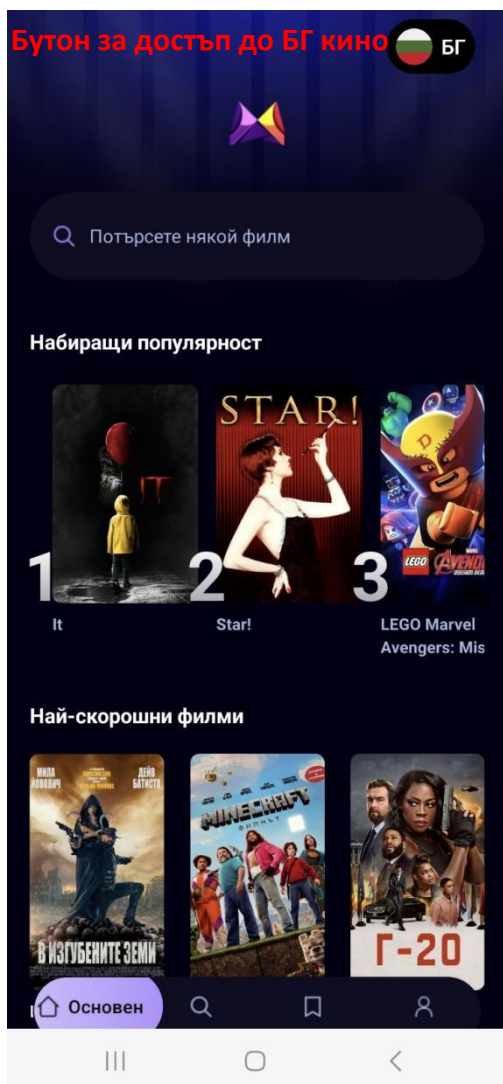
←

При въвеждане на грешни данни (невалиден имейл или парола, несъответствие с данните, различие в паролите, непопълнени полета и т.н) се показва специално предназначения за грешки модал. Пример:

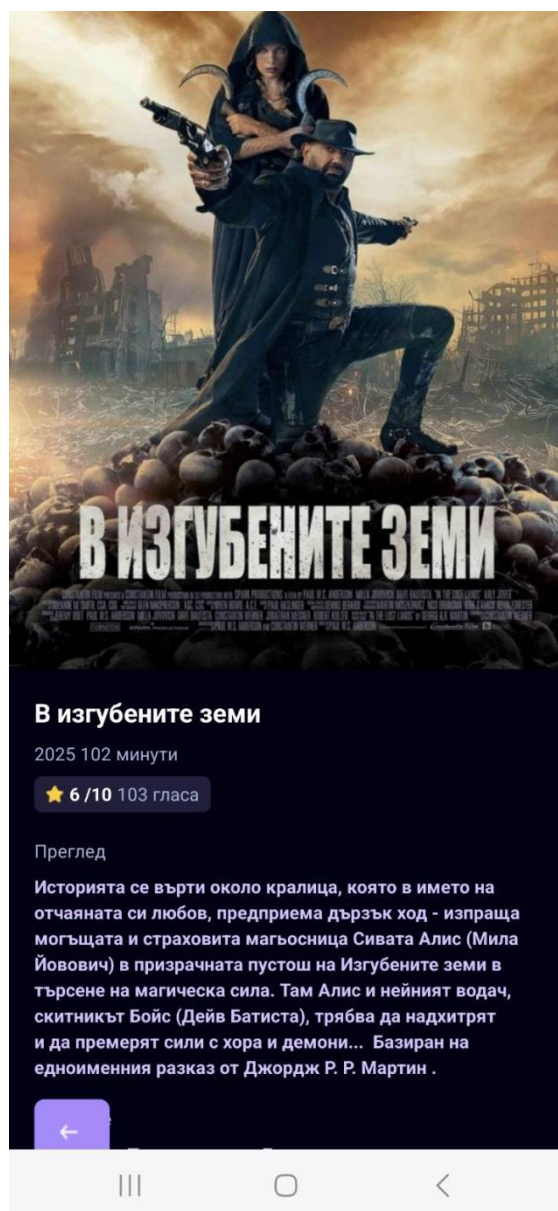


Приемаме, че потребителят е решил да достъпи приложението като гост (без вход в профил):

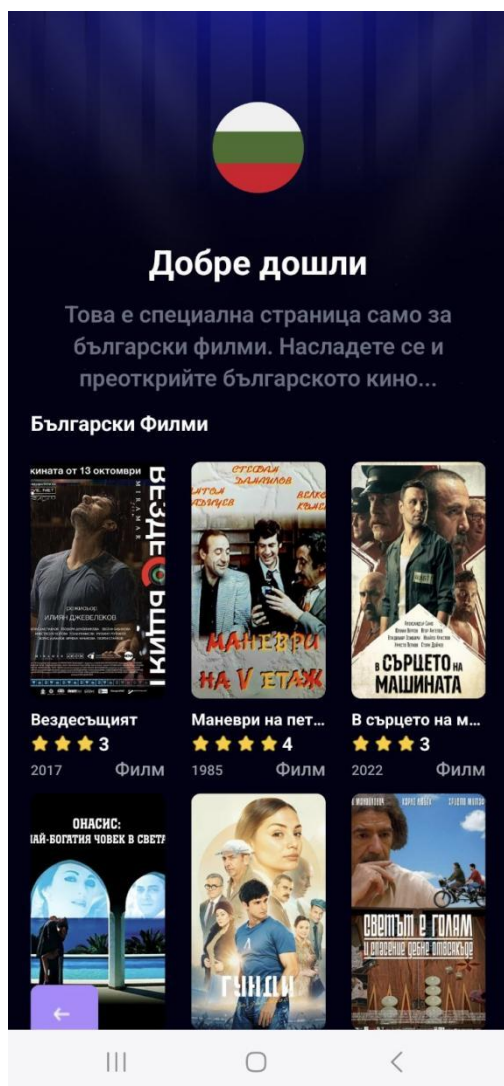
- Основен екран - потребителят разполага с бутон за достъп до страницата с български филми и предавания, търсачка, набиращи популярност филми, последно излезли филми и възможност за показване / скриване на TV предавания



При натискане на даден филм / предаване, гост потребителят вижда специална страница само за конкретния филм / конкретно предаване, на която се показва цялата необходима информация достъпна от външния API сървис (title, release date, runtime, vote_average, vote_count, overview, genres, budget, revenue, production companies и т.н):



Страницата, която е отделена само и единствено за българско кино, изглежда по следния начин:



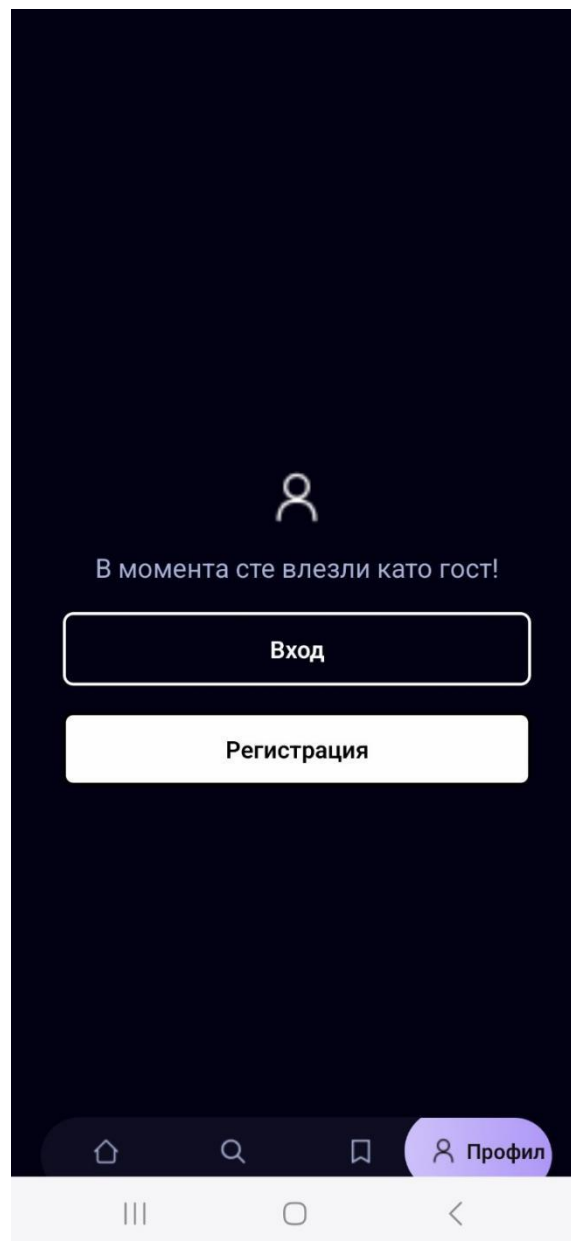
Следващия екран, който е достъпен чрез навигацията в дъното на екрана, е екрана с търсачката, където потребителите имат да възможност да търсят филми по техни избор:



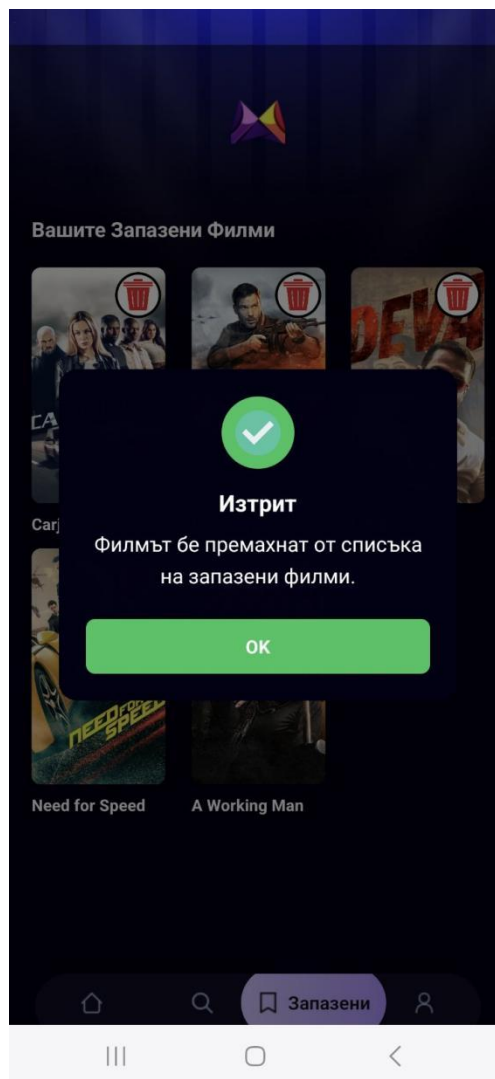
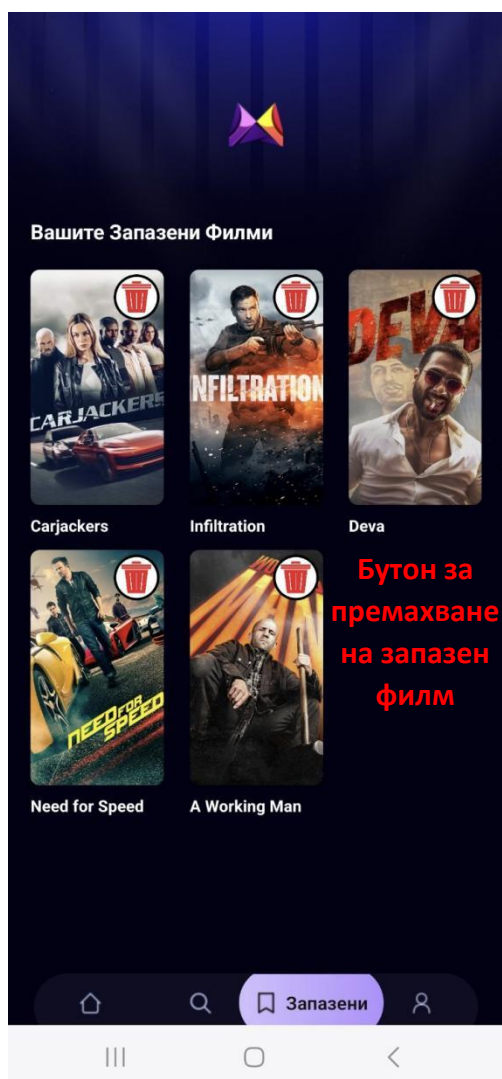
Следващия екран е този със запазените от потребителя филми (В момента той е неактивен / нефункционален, тъй като сме влезли като гости):



Последния таб от навигацията е потребителския профил, с който потребителя е влезнал. Тъй като в момента сме влезли като гости, този екран изглежда така:



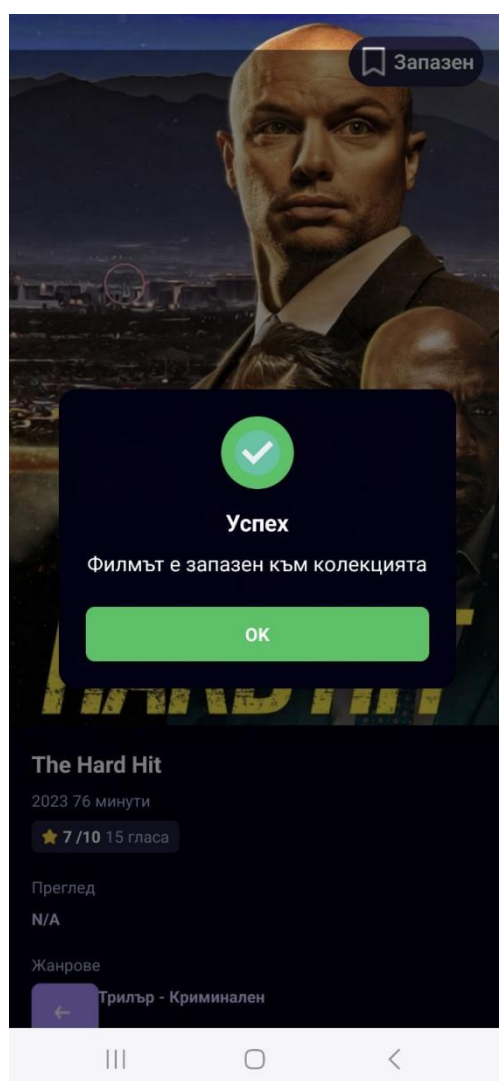
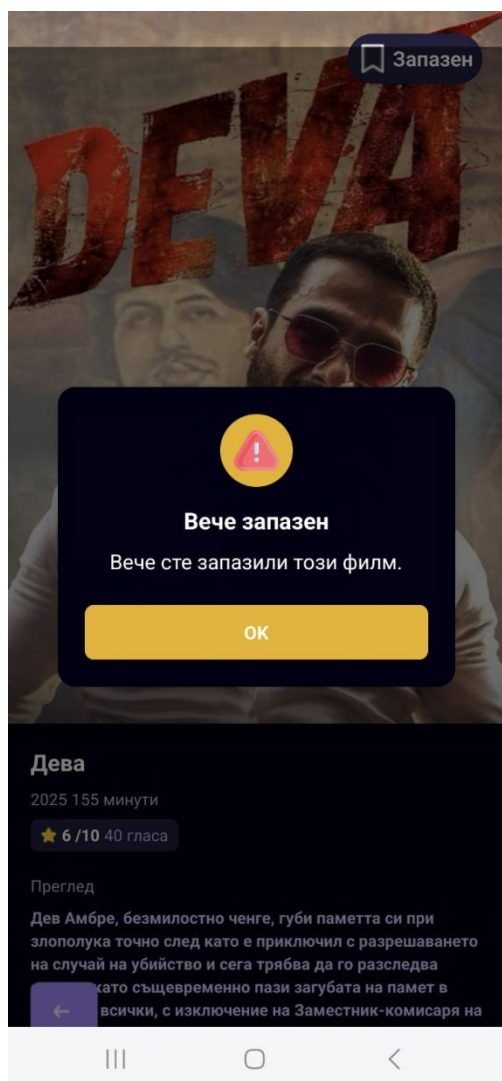
Сега да разгледаме функционалността на приложението при логнат потребител. След успешен вход, потребителят има достъп до страницата със запазените от него филми:



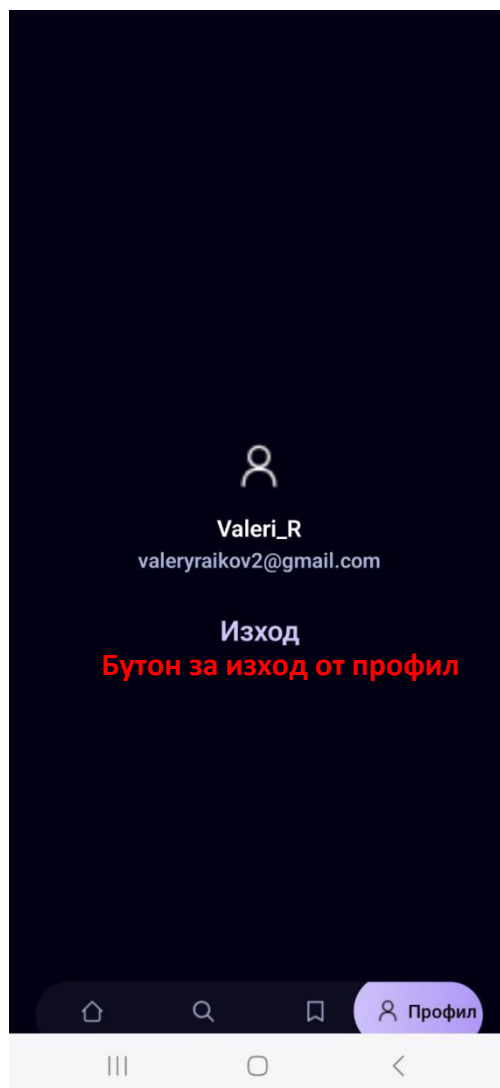
Освен това, когато логнат потребител досъпи страницата с детайлите за конкретен филм, има възможност да го запази в своя списък на запазени:



Ако филмът е вече запазен (както е в случая) получаваме съобщение, че филмът е запазен, а ако не е запаен, съобщение за успех и филмът се добавя към списъка със запазени:



Останалите разлики в приложението след успешен вход или регистрация на потребител са: Бутон, индикиращ успешен вход, който при натискане препраща към “Профил” страницата, която показва детайлите на влезлия потребител:



6. Заключение

Разработеното приложение изпълнява всички основни функционалности, които са заложи в проекта, като предоставя надеждно и стабилно потребителско изживяване. Интерфейсът е достатъчно атрактивен, но същевременно удобен и лесен за използване от потребителя. За допълнително улеснение и по-добро потребителско преживяване, приложението поддържа 2 езика - български и английски. Връзката с външните API услуги е стабилна и всички потенциални грешки и проблеми, които биха могли да се появят, са прихванати и съответните грешки се показват в персонализиран модал за грешки. Продуктът е подходящ за реална употреба и може да бъде разширен с допълнителни функции като любими, рейтинги, списъци за гледане и други.

7. Литература

- **React Native Official Docs** – <https://reactnative.dev>
- **Expo Documentation** – <https://docs.expo.dev>
- **TypeScript Documentation** - <https://www.typescriptlang.org/docs/>
- **Tailwind CSS** – <https://tailwindcss.com>
- **TMDb API Documentation** - <https://developer.themoviedb.org/reference/intro/getting-started>
- **Appwrite Docs** – <https://appwrite.io/docs>
- **i18next Documentation** - <https://www.i18next.com/>
- **YouTube tutorials** - <https://www.youtube.com/watch?v=f8Z9JyB2EIE>,
<https://www.youtube.com/watch?v=sm5Y7Vtuihg>,
https://www.youtube.com/watch?v=YSUmzHH_OMg,
https://www.youtube.com/watch?v=_JDeJgsU-bI,
<https://www.youtube.com/watch?v=JUUXzdbzN0>,
<https://www.youtube.com/watch?v=r88z8nrk8Ww>,
<https://www.youtube.com/watch?v=DTQ7EceGACM>
- **Stack Overflow**
- **AI tools**

Можете да достъпите проекта в моя GitHub профил:

<https://github.com/ValeryRaikov/react-native-projects/tree/main/movieApp>