**Practical Assessment**

**Background:**Your proposal to use Angular as a front-end solution has been accepted by the company director and the board of external stakeholders. Initially, the company have decided to create a basic prototype application that they will release to gauge interest from specific focus groups. Therefore, to begin with, the back end of this application will be emulated using Firebase, and data for the application will be gathered from a free third-party API.

You have been tasked with creating this application prototype using the following REST API: <https://developers.themoviedb.org/3/getting-started/introduction>

**Instructions:**Create a web app based on the above scenario, using Angular as a front-end framework and Firebase to provide persistent data storage and authentication features. Your application should demonstrate the following characteristics:

* Be user-friendly and aesthetically pleasing, implementing clear user instructions, feedback, and error-handling throughout.
* Appropriate animation should be used to enhance the user experience where appropriate.
* This app should function on all screen sizes.
* overall usability of the application, including user-friendliness, responsiveness, usability, and performance.

Please complete the following tasks to implement your application (note, the tasks are not necessarily presented in the order in which you would choose to complete them, this is up to you).

**Task 1: Homepage and Movie Search:**Using the provided API, implement the application’s homepage based on the following specification:

* The homepage should show a list of upcoming/trending movies.
* The user should be provided with a form which allows them to search for a movie based on its title.
* A result-set of movies should be provided to users based on the title they entered. This result-set should show basic information about the movie.

**Task 2: Persistently Store a User’s Subscriptions:**Upon registration/successful authentication, a user should be able to specify a list of active memberships to paid streaming services (see below for a full list to support). These choices should be persistently stored using Firebase.

* A user should later be able to view and manage (add, remove etc.) from this list.

**Task 3: Movie Details:**Every movie should have a dedicated route/component which is used to show detailed information about that movie.

* The exact format of this is up to you; look at the API documentation, see what’s available and what you want to include. At minimum it is expected that basic information such as: title, age-rating, review/ratings & plot be included.

**Task 4: User Specific Results:**If a user has registered an account with the application, and has specified specific subscriptions services, all results should be limited to show only what is freely available to that user.

* The homepage’s trending/up-coming feature should show only what is available to that user.
* The search feature should be limited to freely available results whilst a user is authenticated and has specified active subscriptions.
* A great solution will not just show which movies are available to watch, but also which services these movies are available on. You may wish to show this on the movie details component.
* Note, availability is based on watch region, and this prototype needs only support availability in the UK.

**Task 5: Authentication:**Visitors to the application should have the ability to register an account and login, you will achieve this using Firebase.

* You are welcome to implement social authentication strategies, but there ***must*** be an option to register with an email and password. Note, you will not be penalised if you choose not to implement social authentication.
* You will need to consider auto-login and auto-logout features and deal with these appropriately.
* You will need to consider how you want to handle protected routes.

**Task 6: Optimisation:**To increase the application’s performance, segment functionality into clearly defined and well organised module files. Utilise lazy loading to increase application performance.

**Supported Subscription Services:**Initially, it has been decided that the following subscriptions services will be supported by this application (this is not an exhaustive list, but will be sufficient for the prototype):

* Netflix
* Sky Go
* Now TV
* Amazon Video
* Disney Plus

Your submission must include:

* Full application project directory with the node modules directory removed.
* Firebase login details to use to see the real-time database showing data present for at least two users and any authentication rules present.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** |  |  |  |  |  |
| **Task 1**  Homepage shows trending/up-coming movies and provides a search form to search for movies based on title. | Search functionality is user-friendly and of excellent quality, including good user feedback for success, error, loading etc.  Explicit and well-formed typing is used on all HTTP calls.  HTTP calls fully handle error cases.  Rxjs operators are used to a high degree of competency. | | | | |
| **Task 2**  Allow the user to specify which streaming services they have an active subscription with, persist this data using Firebase. | User data is associated with Firebase user ID correctly and authentication rules are in-place to limit access to this user solely.  It is easy and clear to later add or remove an active subscription service.  Excellent error-handling and feedback is in-place for the area of the application which allows the user to manage their active subscriptions. | | | | |
| **Task 3**  Leverage the supplied API to create a movie details page which provides additional information about a selected movie. | Movie details page makes excellent use of the provided API, displaying various information pieces that are formatted well, clearly and broken down into distinct sections.  API calls are implemented correctly using correct typing and asynchronous task management throughout. | | | | |
| **Task 4**  Limit the results shown on the application to only show movies which are freely available to authenticated users with subscriptions defined | All results, for all aspects of the application are tailored specifically to the user’s saved subscriptions.  It is obvious when viewing a movie which services specifically offer the movie.  All API calls are handled excellently with proper HTTP typing and good use of Rxjs operators.  Non-authenticated users and those with no specified services are shown a general list of results. | | | | |
| **Task 5**  Implement a login and registration system that uses Firebase’s authentication REST API. | Full and complete Firebase Authentication system implemented excellently.  Registration/login forms handle all error cases well with excellent feedback.  Auto-login and auto-logout features are implemented and function correctly with good feedback provided on logout.  Forms are validated to limit all common user errors with excellent feedback provided.  Protected routes are blocked, and proper user feedback is leveraged to provide clear feedback when a protected route is visited by a non-authenticated user. | | | | |
| **Task 6**  Improve application performance using module organisation and lazy loading. | All imports, exports and declarations have been placed in the correct module to provide a fully optimised application.  Application fully implements lazy loading to increase performance. | | | | |
| Create a user-friendly, aesthetic application. This application should demonstrate good usability, error handling and clear user feedback throughout. | Provided app is user-friendly throughout and aesthetically pleasing.  Routing strategy used is sensible and clear error handling is present to deal with 404 scenarios, including great user feedback.  Application loses no features, functionality, or usability on smaller screen sizes.  Animations are utilised excellently to provide an engaging user experience.  Asynchronous tasks are all managed correctly to avoid unnecessary memory leakage | | | | |