

# Cross-Platform App Development Project

## Instructions and Grading Criteria

- Select ONE mobile platform that your application will be developed on (IOS or Android). Web applications will not be considered. After selecting a mobile platform, you must develop and test the React Native application for your selected platform. You are NOT required to develop for both platforms.
- In addition to the required functionality, learners are expected to use the coding conventions demonstrated in class, meaningful variable naming, and clearly organized code. Comments are helpful but not required.
- The majority of grades are assigned based on the correct completion of the required functionality.
- The user interface of your application must be reasonably polished, easy to understand, and readable. Use reasonably pleasant colors and typography

## Submission Checklist:

For your submission to be graded, you must provide a zip file of your project, and a screen recording demonstrating the functionality, you implemented.

### 1. Creating Your Project

- Projects must be created using the Expo CLI.
- When creating your project, name the project: **G#-Project**. Replace # with your group number.

### 2. Before you submit:

- Remove the node\_modules folder from your project
- After removing the node\_modules folder, zip the project.
- Name the project **G#-Project.zip**. No 7zip or rar files are accepted.

### 3. In the assignment dropbox:

- In the submission comments, note which mobile platform that your application is designed to be run on.
- Submit your zip file and screen recording to the dropbox
- If your screen recording is too large for the dropbox, then upload your screen recording to GoogleDrive or OneDrive and ensure that the link is set to: "Anyone with the link can view". Paste a link to the recording in the submission comments.

## Academic Integrity

- This is an individual assessment.
- Permitted activities: Usage of Internet to search for syntax only; usage of course materials
- Not permitted:
  - Communication with others (both inside and outside the class)
  - Discussion of solution or approaches with others; sharing/using a "reference" from someone
  - Searching the internet for full or partial solutions
  - Sharing of resources, including links, computers, accounts

## PROBLEM DESCRIPTION

In this project, you will build a React Native application that enables a user to view a list of currently playing movies and purchase movie tickets.

The application must provide screens to perform the following operations:

- **Now Playing:** Display a list of currently playing movies
- **Movie Details Screen:** Displays information about a specific movie
- **Buy Tickets Screen:** Enables the user to purchase tickets for a specific movie
- **My Purchases Screen:** Displays a list of tickets purchased by the user
- **Login Screen:** Enables the user to login or create a new account
- **Logout Screen:** Enables the user to logout of the application

Example user interfaces for each screen is provided on the course webpage. You are permitted to customise the colours, fonts, and layout; however, you should ensure that your screens are visually pleasing, easy to understand, and easily readable.

### Technical Requirements

1. Screens must be created using function based components.
2. Navigation must be managed using the **React Native Navigation** library:  
<https://reactnavigation.org/>
3. Data Persistence and Authentication must be managed using **Firebase** services. You must use the Expo Firebase library: <https://docs.expo.dev/guides/using-firebase/>
  - You are **NOT** permitted to use the React Native Firebase library (<https://rnfirebase.io/>)
3. Data persistence must be managed using **Firebase Firestore** (use the Firebase web SDK):
4. Authentication must be managed using Firebase Auth and the Email/Password Sign In method.
5. Icons should be managed using the **Expo Vector Icons** library:  
<https://docs.expo.dev/guides/icons/>
6. Data for the movies must be retrieved from the TMDb Movie API.

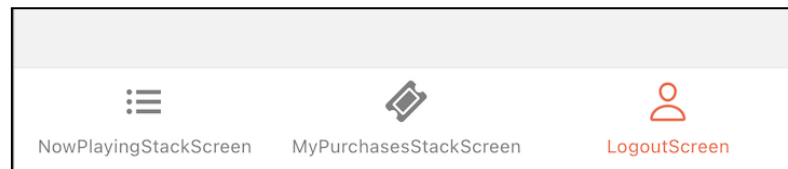
## Navigation

The main screens of the application must be displayed using a Tab Bar Navigation component. Each tab bar menu item **must** have an **appropriate name** and **icon**.

1. If there is NO user logged into the application, then the Tab Bar should display 2 menu items: **Now Playing** and **My Purchases**.



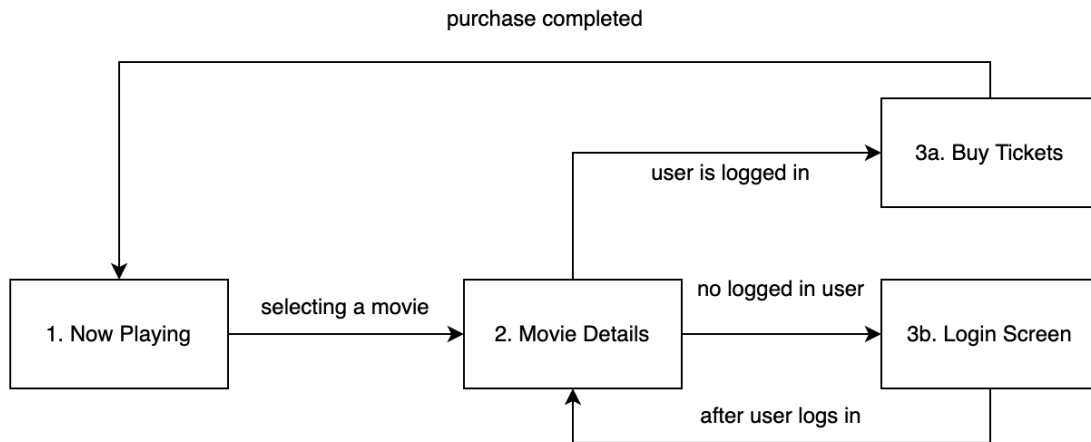
2. If there IS a logged in user, then the Tab Bar should display 3 menu items: **Now Playing**, **My Purchases**, and **Logout**



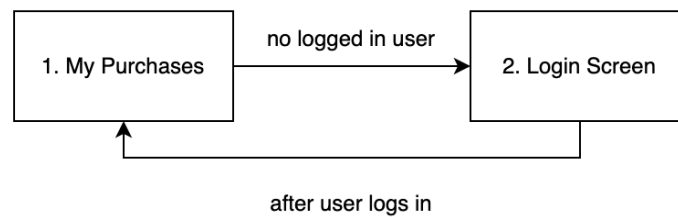
3. A tab bar menu option can be conditionally rendered by adding logic to the App.js file to determine if there is a logged in user. For example, in the below code sample, ScreenA is always displayed; but ScreenB is only displayed if there is a logged in user.

```
export default function App() {  
  // state variable to track if there is a logged in user  
  const [userLoggedIn, setUserLoggedIn] = useState(false)  
  useEffect(()=>{  
    // code to check if there is a logged in user  
  },[])  
  return (  
    <NavigationContainer>  
      <Tab.Navigator>  
        <Tab.Screen name="ScreenA" component={ScreenA} />  
        { (userLoggedIn === true ) && <Tab.Screen name="ScreenB"  
component={ScreenB} /> }  
      </Tab.Navigator>  
    </NavigationContainer>  
  );  
}
```

4. The **Now Playing** Screen is connected to other screens in the application, as follows. Use a **Stack Navigator** to connect the screens together.



5. The **My Purchases** screen is connected to other screens in the application, as follows. Use a **Stack Navigator** to connect the screens together.



## Now Playing Screen

This screen retrieves and displays a list of currently playing movies.

1. The movies must be retrieved using **The Movie Database API**

- The URL endpoint for the API is:

`https://api.themoviedb.org/3/movie/now_playing?api_key=YOUR_API_KEY&language=en-US&page=1&region=CA`

- To use the API, you will need to sign-up for an API key. After obtaining an API key, replace YOUR\_API\_KEY with your assigned key. See this link: <https://www.themoviedb.org/settings/api/request> (you must be logged into the TMDB website to view the link)

2. Display the retrieved movies in the UI as a <FlatList>. Each row of the FlatList must show the **movie name**, the **release date**, and an **icon** indicating that the row can be clicked on.

*Example of a row in the FlatList*

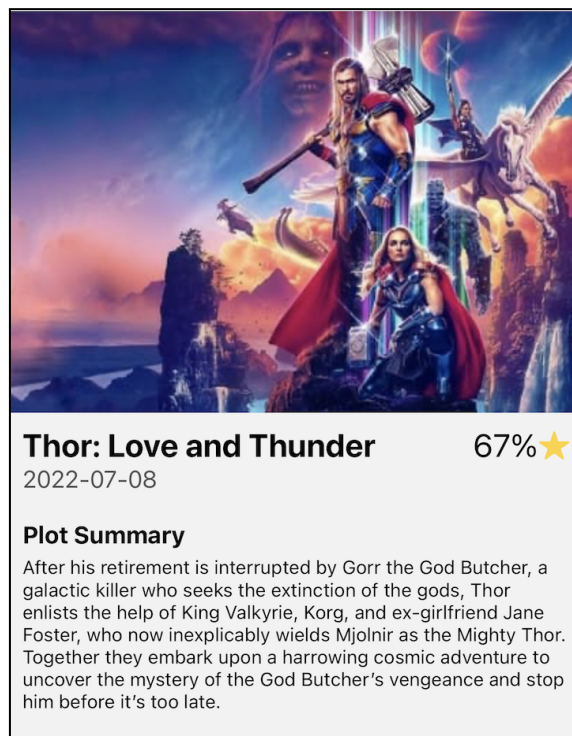
Now Playing	
Thor: Love and Thunder Release Date: 2022-07-08	>
Jurassic World Dominion Release Date: 2022-06-10	>

3. When the user clicks on a row in the FlatList, send the currently selected movie data to the **Movie Details Screen**. Then, navigate the user to the screen.

## Movie Details Screen

This screen displays details about a selected movie. The user arrives on this screen by clicking on a movie in the **Now Playing** screen.

1. Using the data sent by the Now Playing screen, display the **backdrop image**, **movie name**, **average voter rating + star icon**, **release date**, and movie **overview**.



2. The screen must display a BUY TICKETS button. This button is used by the user to purchase tickets for the currently selected movie.

A user can only purchase tickets **if they are logged in**. If the user is **NOT** logged in, then the BUY TICKETS button must be disabled, and an option provided to the user to login/create a new account.

*Example UI for logged in vs. not logged in user*

User is NOT logged in	User is Logged In
<p><b>Plot Summary</b></p> <p>After his retirement is interrupted by Gorr the God Butcher, a galactic killer who seeks the extinction of the gods, Thor enlists the help of King Valkyrie, Korg, and ex-girlfriend Jane Foster, who now inexplicably wields Mjolnir as the Mighty Thor. Together they embark upon a harrowing cosmic adventure to uncover the mystery of the God Butcher's vengeance and stop him before it's too late.</p> <p>You must be logged in to use this feature.</p> <p>Buy Tickets</p> <p>Login or Create New Account</p>	<p><b>Plot Summary</b></p> <p>After his retirement is interrupted by Gorr the God Butcher, a galactic killer who seeks the extinction of the gods, Thor enlists the help of King Valkyrie, Korg, and ex-girlfriend Jane Foster, who now inexplicably wields Mjolnir as the Mighty Thor. Together they embark upon a harrowing cosmic adventure to uncover the mystery of the God Butcher's vengeance and stop him before it's too late.</p> <p>Buy Tickets</p>

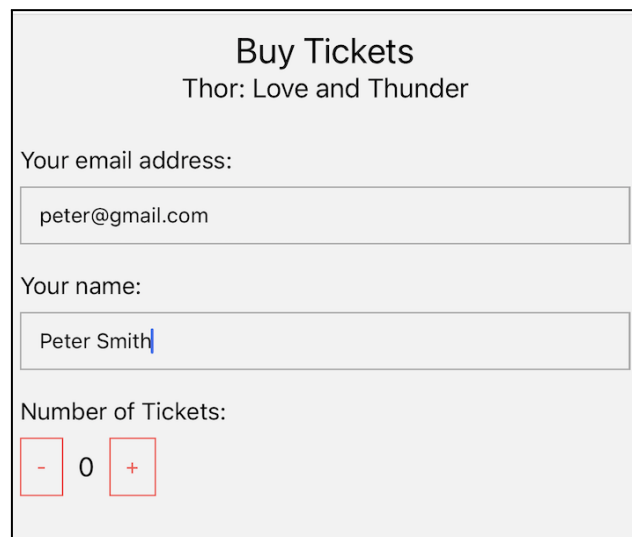
5. When the BUY TICKETS button is pressed, send the selected movie details to the **Buy Tickets** screen. Then, navigate the user to the screen.

6. If the LOGIN button is pressed, send the user to the **Login Screen**. After logging in, the user should be **returned to this screen** (Movie Details Screen)

## Buy Tickets Screen

This screen enables the user to purchase tickets to a selected movie.

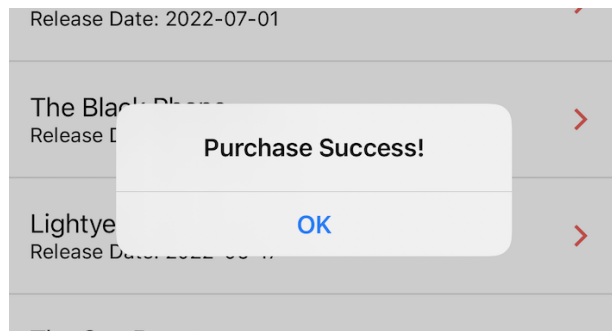
1. When the screen loads, display **the movie name**, and a form that enables the user to enter their purchase details. The form must capture the user's **email address**, **name**, and **the number of tickets** the user wants to purchase.



The form is titled "Buy Tickets" with the movie name "Thor: Love and Thunder" below it. It contains three input sections: "Your email address:" with a text box containing "peter@gmail.com", "Your name:" with a text box containing "Peter Smith", and "Number of Tickets:" with a numeric input field showing "0" and red minus/plus buttons.

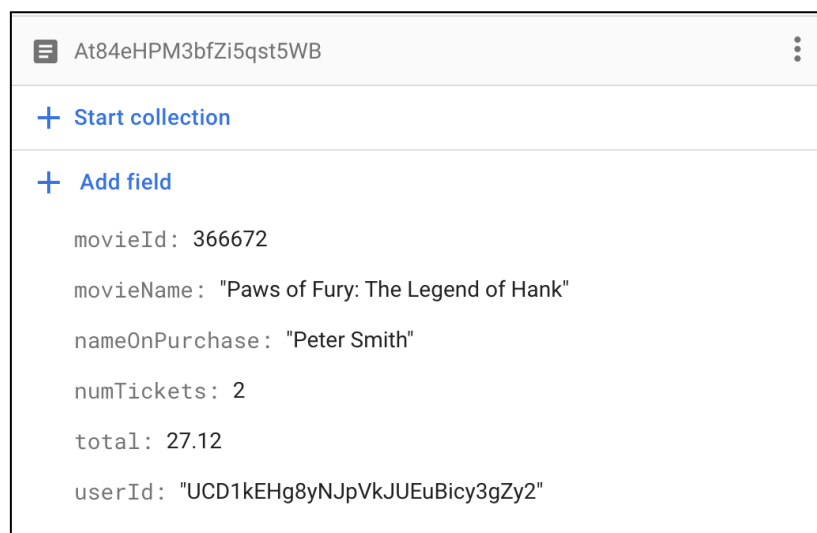
2. The **email address** field should be automatically populated with the logged in user's email address.
3. The **name** field should be manually filled in by the user.
4. The **default** number of tickets is 0. Pressing the - and + button will increase/decrease the displayed number of tickets by 1.
5. If the number of selected tickets is greater than 0, display an **order summary**. The summary should display: the **name** of the movie, the **number of selected tickets**, the **subtotal**, **tax**, and **total** for the purchase.
  - The price of a single movie ticket is \$12
  - Sales tax is 13%
6. If the number of selected tickets is 0, hide the order summary.
7. When the user presses the CONFIRM PURCHASE button, **save the purchase** to Firestore, **display a success message**, and **redirect** the user back to the **Now Playing** screen.








8. The following purchase details must be saved to Firestore:

- The movie id, as specified by the API
- The name of the movie
- The name on the purchase (this is the value entered in the name form field)
- The number of tickets
- The total paid
- The user's id, as assigned by Firebase Authentication



## My Purchases Screen

This screen displays information about the user's previous ticket purchases.

MyPurchasesScreen	
Your Tickets	
	Minions: The Rise of Gru Num Tickets: 2 Total Paid: 27.12
	Paws of Fury: The Legend of Hank Num Tickets: 2 Total Paid: 27.12
	Thor: Love and Thunder Num Tickets: 4 Total Paid: 54.24

1. When the screen loads, display a list of purchases made by the **currently logged in user** in a **FlatList**. The list of purchases must be retrieved by **querying** Firestore for any purchases made by the currently logged in user (use the UID of the currently logged in user as the WHERE clause of the query)

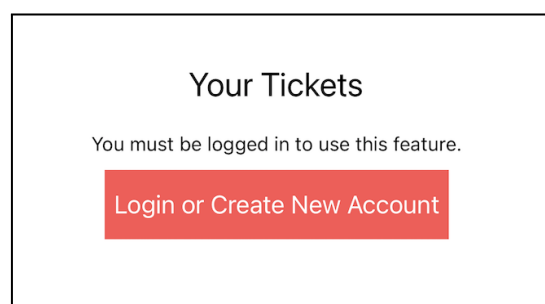
Here are examples of how to query a Firestore collection:

- Define the query:  
[https://firebase.google.com/docs/firestore/query-data/queries#simple\\_queries](https://firebase.google.com/docs/firestore/query-data/queries#simple_queries)
- Execute the Query:  
[https://firebase.google.com/docs/firestore/query-data/queries#execute\\_a\\_query](https://firebase.google.com/docs/firestore/query-data/queries#execute_a_query)

2. Each row of the FlatList must display an **icon**, the **name of the movie**, the **number of purchased tickets**, and the **total paid**. Ensure the **total paid** is a different text color than the other text in the row.

3. If there is no user logged in, then provide the user a way to LOGIN. After logging in, the user should be returned to this screen (My Purchases).

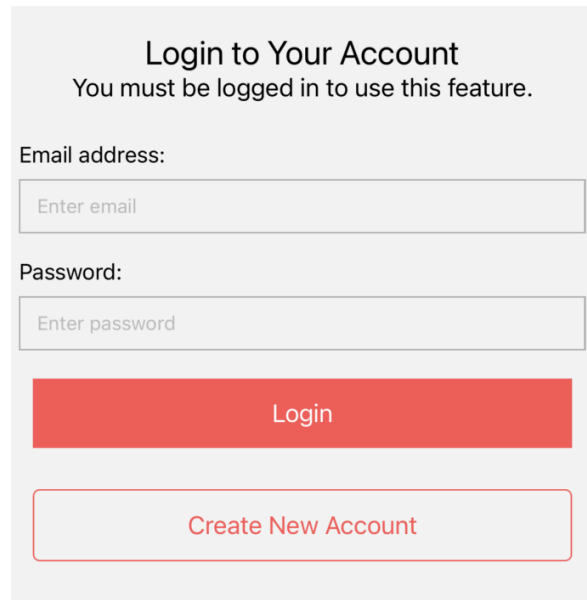
An example UI is:



## Login Screen

You must provide a **one (1) login screen** that is used by the application whenever the user is required to login.

1. The screen must be used for both login and new account creation. An example UI is:

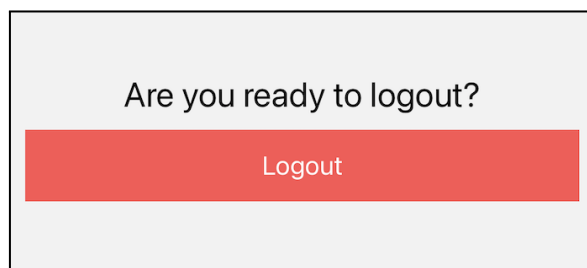


The image shows a login screen with a light gray background. At the top, the title "Login to Your Account" is centered in a bold black font. Below the title, a subtitle "You must be logged in to use this feature." is centered in a smaller black font. Underneath, the label "Email address:" is followed by a text input field with the placeholder text "Enter email". Below this, the label "Password:" is followed by another text input field with the placeholder text "Enter password". At the bottom of the form, there are two buttons: a solid red button labeled "Login" and a white button with a red border labeled "Create New Account".

2. This screen is **not available** as a Tab Bar menu item.
3. User account creation and login must be managed by Firebase Authentication. Use the **Email/Password** sign in method.

## **Logout Screen**

You must provide a screen that enables the user to logout of their account. Here is an example UI:



The image shows a logout screen with a light gray background. In the center, the text "Are you ready to logout?" is displayed in a black font. Below this text is a solid red button labeled "Logout".

1. User logout must be managed by Firebase Authentication.
2. After the user logouts, navigate the user to the **Now Playing** screen.