TP02 – Creating a Videogame Information Mobile Application

David Gogue, Nikolaj Wochnik (Chaney), Mitch Fade  
Team XX , CS624 Full Stack Development – Mobie, MSCS, City University of Seattle

[chaneyjeff@cityuniversity.edu](mailto:chaneyjeff@cityuniversity.edu)

[goguedavidwilliam@cityuniversity.edu](mailto:goguedavidwilliam@cityuniversity.edu)

[fademitch@cityuniversity.edu](mailto:fademitch@cityuniversity.edu)

1. **Introduction**

For the TP02 assignment a mobile application with Create Read Update and Delete (CRUD) functionality was created and then queries to the RAWG videogame database API were made from the application to populate information about the videogame with data from this API service. The application contains a homepage that lists all of the videogames on the application, a page to add a new videogame and a videogame card page which includes details about an individual videogame that includes data populated from the RAWG database. A delete button is also on items on the homepage to delete videogames.

**2.** **Creating a Mobile Application with CRUD Capability**

A mobile application with CRUD capabilities was developed in React Native using expo. Currently an in-memory CRUD system was created to view and modify the data. The code for the homepage is in the index.tsx file while the code for the page with the form to add new items is in the add.tsx file and the page for the form to edit items is in the edit.tsx file. The file for the card page for individual videogames is in the item.tsx file.

**Overview of ItemContext.tsx:**

The ItemContext.tsx file manages a shared list of items[] and exposes the CRUD operations addItem, updateItem and deleteItem. Each item is an object with the following attributes:

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Data type |
| id | Random text that acts as a uuid identifier for an entry | String |
| Name | The name of the videogame | String |
| Description | Details about the videogame – optional | String |

The ItemContextType variable defines which data to use and which CRUD methods can interact with the data. The UseItems() custom hook is used to access the context so that any component can have easy access to the context. Custom hooks can be used to share logic between separate components and can be written for more specific purposes than Reacts built in hooks such as UseState() or UseEffect() (*Reusing Logic With Custom Hooks – React*, n.d.) The ItemProvider() contains the functions for CRUD operations including addItem, updateItem and deleteItem. These modify items in the items[] list.

**Use of ItemContext.tsx in the other files:**

The UseItems() hook is called in the add.tsx file to use the addItems method that is defined in the ItemContext.tsx file. Similarly the edit.tsx file uses the UseItems() hook to utilize the updateItem function defined in ItemContext.tsx. The code for the homepage in index.tsx uses the UseItem() hook to retrieve the items and deleteItem methods since this page shows a list of all the entries and has a button to delete entries.

**Demonstration of the CRUD enabled mobile application:**

The mobile application was tested on the Expo Go mobile app on an iPhone 15 with an iOS version of 18.4.1. The homepage shows a list of videogames that are in the application:

A screenshot of a phone

AI-generated content may be incorrect.

The form to add a new videogame is shown:

A screenshot of a phone

AI-generated content may be incorrect.

The form to edit a videogame entry is shown:

A screenshot of a computer

AI-generated content may be incorrect.

The card page to view an individual videogame is shown:

A screenshot of a video game

AI-generated content may be incorrect.

**Note:** The card page is populated with data from the RAWG database which will be explained in the next section.

**3.** **Integrating RAWG queries on the game card page**

The RAWG database of videogames provides an API to return data about videogames by name and includes entries for over 350,000 videogames. This API service is free for projects with less than 100,000 monthly active users or less than 500,000 page views per month (RAWG Video Games Database API, n.d.). An API call is made to the RAWG database in the index.tsx file using the URL:

https://api.rawg.io/api/games?key=${API\_KEY}&search=${encodeURIComponent(name as string)}

The API key was obtained from the RAWG videogame database website and entered in the index.tsx file. It is important to manage API keys securely and omit them from being directly committed to code repositories. A UseEffect() hook was used to make a call to the API. The UseEffect() hook is useful for managing side effects that are outside of the normal rendering flow of the application such as ftching data from API’s (GeeksforGeeks, 2025). Conditional logic is then used to return an activity indicator if a query to the API has not finished (ActivityIndicator, 2025), text that the game is “not found” if it is not in the RAWG database or details about the game if it is found. The Data about the game that is returned includes the games name, an image of the game, the games popularity rating, the date it was released and the genre of game.

**4. Future Improvements**

This mobile application currently only supports in memory storage which causes data to be deleted if the application host is stopped. React Native Asynchronous storage will be used to persist data after the application is stopped. Asynchronous storage creates a key-value based data storage system that serves as an abstraction on top of mobile platform specific database technology that allows code to be written for multiple platforms without needing to interact with platform specific databases (Siddique, 2023).

**4. REFRERENCES**

*ActivityIndicator · React Native*. (2025, April 14). <https://reactnative.dev/docs/activityindicator>

GeeksforGeeks. (2025, April 12). *ReactJS UseEffect Hook*. GeeksforGeeks. https://www.geeksforgeeks.org/reactjs-useeffect-hook/

*RAWG Video Games Database API*. (n.d.). <https://api.rawg.io/docs/>

*Reusing Logic with Custom Hooks – React*. (n.d.). <https://react.dev/learn/reusing-logic-with-custom-hooks>

Siddique, M. A. (2023, October 21). AsyncStorage in React Native: The Complete Guide - Muhammad Abubakar Siddique - Medium. *Medium*. <https://medium.com/@dev.abubakarsiddique/asyncstorage-in-react-native-the-complete-guide-deaa2b3b3665>

**Appendices and Annexures**