Design and Development of a Mobile App User Interface with Enhanced MVC Architecture

Fundamentals of Software Design: CS374

Antoine Gaton

November 11, 2024

Contents

[Introduction (Copy Over the Requirements) 3](#_Toc142060541)

[Model–View–Controller (MVC) Architectural Paradigm 4](#_Toc142060542)

[Definition of MVC 4](#_Toc142060543)

[The Most Important Design Concept of MVC 4](#_Toc142060544)

[Design Pattern 5](#_Toc142060545)

[Some of the Special Considerations of a Mobile App 5](#_Toc142060546)

[Design Pattern (Rationale and Proper UML Diagram) 5](#_Toc142060547)

[Prototype 6](#_Toc142060548)

[MVC Paradigm With the Design Pattern 6](#_Toc142060549)

[Code of Prototype 6](#_Toc142060550)

[Prototype Results Screenshots 6](#_Toc142060551)

[References 7](#_Toc142060552)

# Introduction (Copy Over the Requirements)

This document details the design and development of a mobile app interface that allows users to search for products within a catalog. The app utilizes the Model-View-Controller (MVC) architectural pattern for organized and modular development, with additional design patterns to enhance functionality and responsiveness. The app’s GUI includes a search bar to filter products by name, category, or price, displaying real-time results. The design considers mobile-specific features such as adaptable themes and an accessible user interface.

# Model–View–Controller (MVC) Architectural Paradigm

## Definition of MVC

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main components:

* **Model**: Manages the data and business logic of the application. In this app, PRODUCTS serves as the data model, containing a list of products with properties such as name, category, and price.
* **View**: Displays the user interface and is responsible for presenting the data from the Model to the user. The SearchScreen component in this app represents the View, which displays the search bar, results, and product details.
* **Controller**: Acts as an intermediary between the Model and View. It processes user input, updates the Model, and refreshes the View accordingly. The handleSearch function in SearchScreen acts as the Controller, filtering product data based on the search query.

## The Most Important Design Concept of MVC

The main advantage of MVC in this app is separation of concerns, where the data (Model), user interface (View), and user input handling (Controller) are managed independently. This modularity enhances the app’s maintainability and scalability, allowing for easy updates or modifications to one component without affecting the others.

# Design Pattern

## Some of the Special Considerations of a Mobile App

Mobile applications have specific requirements, such as:

* **Adaptability** to light and dark themes.
* **Responsive UI** that efficiently uses limited screen space.
* **Performance Optimization** for smooth user interactions.

These considerations guided the design, leading to the use of theming and state management to provide a seamless user experience.

## Design Pattern (Rationale and Proper UML Diagram)

In this mobile app project, we use two main design patterns to enhance the MVC structure and meet the specific requirements of mobile applications: the **Decorator Pattern** and the **Observer Pattern**. Each of these patterns serves a unique purpose in improving the app's flexibility, responsiveness, and maintainability.

1. **Decorator Pattern**
   * **Definition**: The Decorator Pattern is a structural pattern that allows behavior to be added to individual objects, dynamically, without affecting the behavior of other objects from the same class. This pattern wraps an object in a decorator class that adds or overrides functionality.
   * **Application in this App**: The useThemeColor hook applies to the Decorator Pattern by dynamically assigning theme colors to ThemedView and ThemedText components. This allows the app to adapt to different themes (e.g., light and dark mode) without modifying the core structure of the components themselves.
   * **Rationale**: The Decorator Pattern enables the app to seamlessly switch between light and dark themes by dynamically applying colors based on the selected theme. This enhances user experience by making the UI adaptable and visually consistent across different themes.
2. **Observer Pattern**
   * **Definition**: The Observer Pattern is a behavioral pattern that allows an object (subject) to notify other objects (observers) of changes in its state, automatically. Observers register with a subject to receive updates whenever the subject's state changes.
   * **Application in this App**: The Observer Pattern is indirectly applied using React’s useState hook to manage the searchQuery and results states. When the search query changes, it triggers an automatic update to the displayed results, reflecting the latest user input without manual intervention.
   * **Rationale**: This pattern allows the app to provide real-time updates to the user interface, showing filtered product results as the search query changes. This real-time responsiveness improves usability and ensures that the UI remains synchronized with the user's actions.

**UML Diagram for Decorator and Observer Patterns**

The UML diagram below illustrates how the Decorator and Observer patterns are implemented in the app.

A diagram of a software application

Description automatically generated with medium confidence

**Explanation of the UML Diagram**

1. **Decorator Pattern with useThemeColor**:
   * The useThemeColor class represents the Decorator Pattern. It provides a method, getColor, which applies theme colors to components based on the selected theme.
   * useThemeColor connects to ThemedView and ThemedText, applying the necessary color attributes (e.g., background or text color) based on the current theme (light or dark mode). This connection illustrates how the Decorator Pattern dynamically decorates these components with theme-based colors, allowing them to adapt to different themes without altering their internal logic.
2. **Observer Pattern with SearchScreen**:
   * The SearchScreen component contains searchQuery and results properties, managed with React’s useState hook. handleSearch updates searchQuery and triggers filtering of PRODUCTS, which updates the results.
   * This setup indirectly applies to the Observer Pattern. As searchQuery changes, the results are automatically updated and displayed in SearchScreen. This allows the app to reflect changes in real time, ensuring the user sees the most relevant product information based on their search.

# Prototype

## MVC Paradigm With the Design Pattern

The SearchScreen component functions as the main View, displaying the search interface and product results. It includes handleSearch as the Controller, which filters data in PRODUCTS based on the search input. The use of ThemedView and ThemedText components applies to the Decorator Pattern to adapt the UI to the theme.

## Code of Prototype

## A screen shot of a computer screen Description automatically generatedPrototype Results Screenshots

Github Repo: <https://github.com/AntoineGaton/CTU/tree/main/Fundamentals_of_Software_Design/Unit5Prototype>

# References

IBM Developer. (n.d.). *Design patterns for mobile applications*. IBM. Retrieved from https://developer.ibm.com/articles/design-patterns-mobile-applications/

GeeksforGeeks. (n.d.). *Introduction to MVC architecture*. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/mvc-design-pattern/

Smashing Magazine. (2018, January). *Ultimate guide to mobile UI patterns*. Smashing Magazine. Retrieved from https://www.smashingmagazine.com/2018/01/guide-mobile-ui-patterns/