

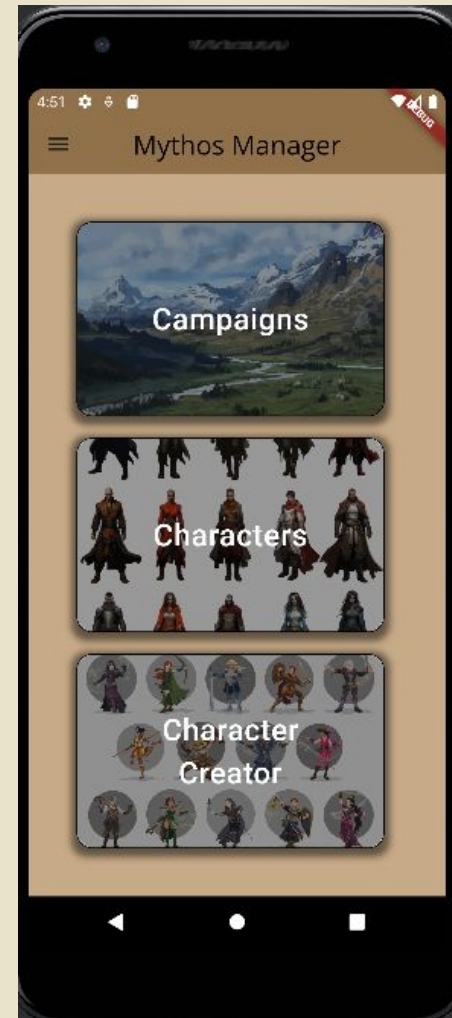
The background features a collection of abstract geometric shapes. In the top left, there is a small white square with a black border and a larger orange square. On the right side, there is a vertical black bar, an orange bar, and a teal curved line. In the bottom left, there are horizontal orange lines, a black circle, and orange and black rectangular blocks. The title 'Mythos Manager' is centered in a large, bold, brown font.

Mythos Manager

Liam, Jonathan, Shreif

Introducing Mythos Manager

- DND campaign and character manager
- Create, manage, and share characters
- Embark on campaigns with campaign and note creation





Application Overview

Characters


- Character Creation
 - ▶ Class Selection
 - ▶ Race Selection
 - ▶ Background Selection
 - ▶ Backstory and general character information
- Character list screen to view a list of all of your created characters
- Character view screen to view a specific character and all of its details

Campaign and Notes

- Campaign Creation
 - ◀ Name
 - ◀ Details
 - ◀ Campaign character
- Ability to add any specific notes about a campaign
- Campaign list screen to view all campaigns
- Campaign display screen to view a particular campaign including its details, notes, and linked character

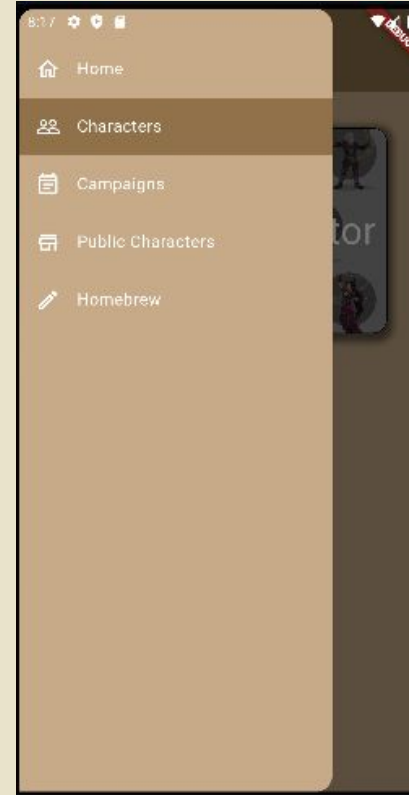


Public Characters

- After a user creates a character, they have the option to share it to all other Mythos Manager users
 - In the character display screen, users will see a share button that will make a character public when pressed
 - Users can filter public characters by class and subclass
 - Once public, anyone can view a characters details, and also use it in their campaigns
- 

App Navigation

- Homepage has 3 buttons to Campaigns, Characters, and Character Creator
- Hamburger button to open a drawer to navigate to Campaigns, Characters, Public Characters, and Homebrew (Not implemented)





Architectural Elements

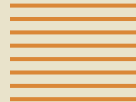
State Management - Riverpod

- Riverpod for Scalable State Management
 - ◀ Utilizes Provider, StreamProvider, and AsyncNotifierProvider for state management.
 - ◀ Implements authenticationControllerProvider to manage auth state asynchronously.
 - ◀ Facilitates clean separation of UI and business logic.
- Reactive UI Updates
 - ◀ Consumer / HookConsumer used for reactive UI components based on auth and character states.
 - ◀ ref.watch mechanism to listen to changes in state and update UI accordingly.
- Decoupled Architecture
 - ◀ State providers and controllers encapsulate Firebase operations, enhancing testability and maintenance.

DND API

- Dynamic Content Fetching
 - ◀ Uses DNDAPIService to interact with the D&D API for fetching races, classes, equipment, etc.
 - ◀ Provides real-time data for character creation, enhancing user experience.
- Caching and Efficiency
 - ◀ Potential caching strategies for API calls to minimize latency and data usage.
 - ◀ Asynchronous data fetching with error handling for robust app performance.

Firebase Authentication




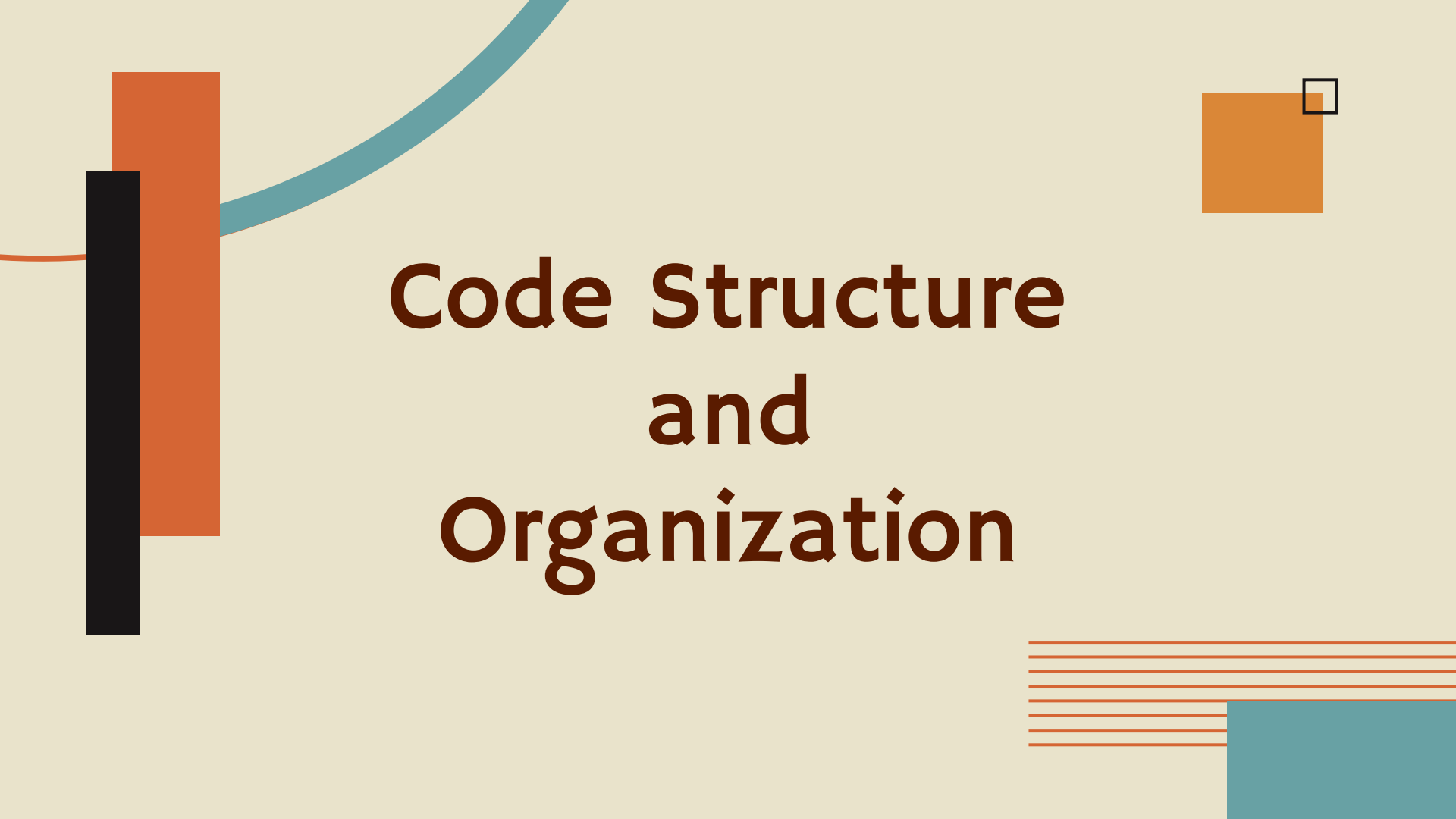
- Secure User Authentication
 - ◀ Implements sign-up, login, and sign-out functionalities using Firebase Email/Password Authentication.
 - ◀ AuthenticationService encapsulates Firebase Auth methods, simplifying auth operations across the app.
- User State Management
 - ◀ authenticationStateProvider streams auth state changes, enabling UI to react to user sign-in and sign-out.
 - ◀ Ensures secure and seamless user experience across different app components.





Firestore Database


- NoSQL Database Usage: Storing and retrieving data from Firestore, a NoSQL database, for dynamic content management.
 - Real-time Data Sync: Highlighting Firestore's real-time capabilities in syncing character data across the application.
 - CRUD Operations: The CampaignRepository and CharacterRepository encapsulate CRUD operations, showcasing data manipulation for campaign and character entities.
 - Data Modeling and Retrieval: Illustrating the use of Firestore for complex queries, such as fetching characters based on user IDs or public visibility.
- 

The background features several abstract geometric elements: a thick orange vertical bar on the left, a dark blue vertical bar partially overlapping it, a teal curved line starting from the top left and arching towards the center, a small orange square with a black outline in the top right, and a series of horizontal orange lines of varying lengths in the bottom right corner, some of which overlap a teal rectangular block.

Code Structure and Organization



Overall Structure

- Feature first structure
 - Each feature broken into four categories
 - ◀ Presentation
 - ◀ Application
 - ◀ Domain
 - ◀ Data
 - MVC and Service-Repository architectural patterns used for backend code
- 

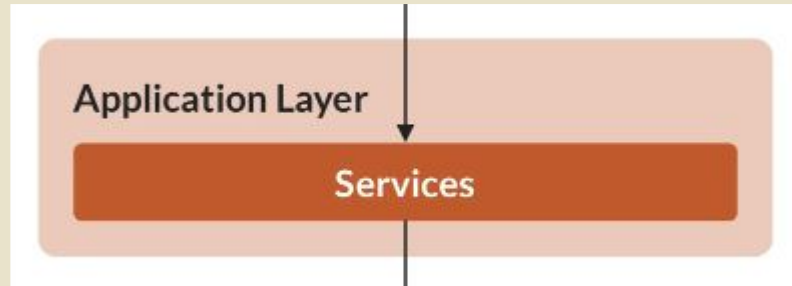
Presentation Layer

- Holds widgets in the form of screens or components
- Holds controllers to delegate operations to the application layer
- Controllers use Riverpod notifiers
 - ◀ AsyncNotifier
 - ◀ FamilyAsyncNotifier
- When a notifiers state changes within the controller it will force widgets watching the controller to rebuild



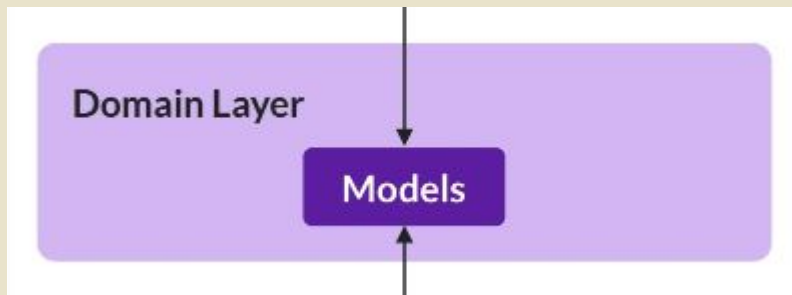
Application Layer

- Holds services which delegate operations to data layer
 - ◀ Typically have multiple repository dependencies to use in conjunction with each other
 - ◀ CharacterService, CampaignService
- Used by controllers in the presentation layer
- Services are used through a riverpod Provider



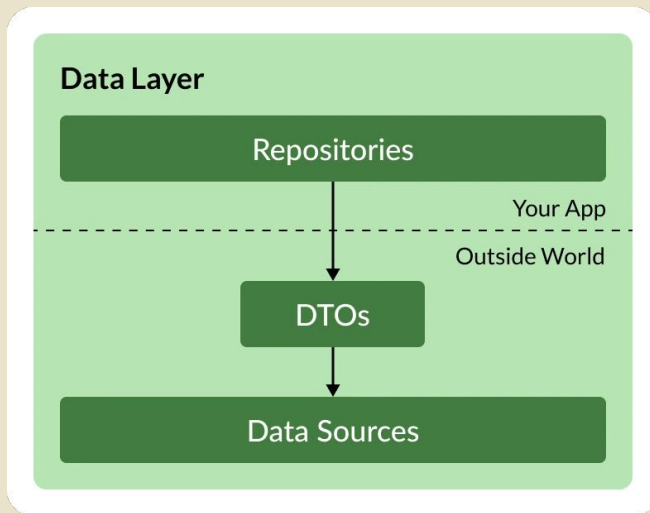
Domain Layer

- Holds data models
 - ▶ Character
 - ▶ Campaign
 - ▶ Note
- Used as DTO's for storing data within Firestore Database
- Each model has converter methods **toFirestore** and **fromFirestore**

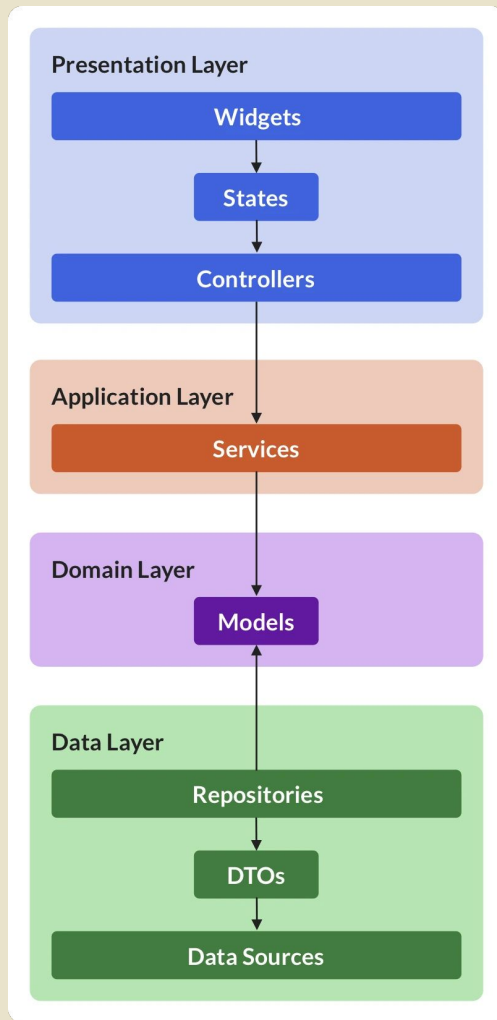


Data Layer

- Holds repositories that delegate CRUD operations to the outside world
 - ▶ Firebase Authentication
 - ▶ Firestore Database
 - ▶ DND API
- Used by services in the application layer
- Repositories are used through a riverpod provider




Basic Structure



Directory Structure & Naming Convention


```
lib/  
├── features/  
│   ├── feature_name/  
│   │   ├── application/  
│   │   │   └── name_service.dart  
│   │   ├── data/  
│   │   │   └── name_repository.dart  
│   │   ├── domain/  
│   │   │   └── name.dart  
│   │   └── presentation/  
│   │       ├── controllers/  
│   │       │   └── name_controller.dart  
│   │       └── screens/  
│   │           ├── components/  
│   │           │   └── component_name.dart  
│   │           └── name_screen.dart
```



Planned Features




Homebrew

- Would have allowed users to create their own races, subraces, classes, etc.
 - Users would be able to apply these homebrew creations to their own characters
 - Like character sharing, users would have been able to share their homebrew creations to other Mythos Manager users
- 



Character Editing

- Allow user to edit their character overtime as they wish
 - Would have utilized same multi page system for creating a character to edit
- 



Campaign Groups

- Add ability to create a group campaign, users can invite each other to a campaign
 - Character for a campaign can be seen by others, except for each others backstory
 - Notes can be shared
 - Group chat
 - Calendar for planning
- 