Muzaffar Sharapov

Documentation:

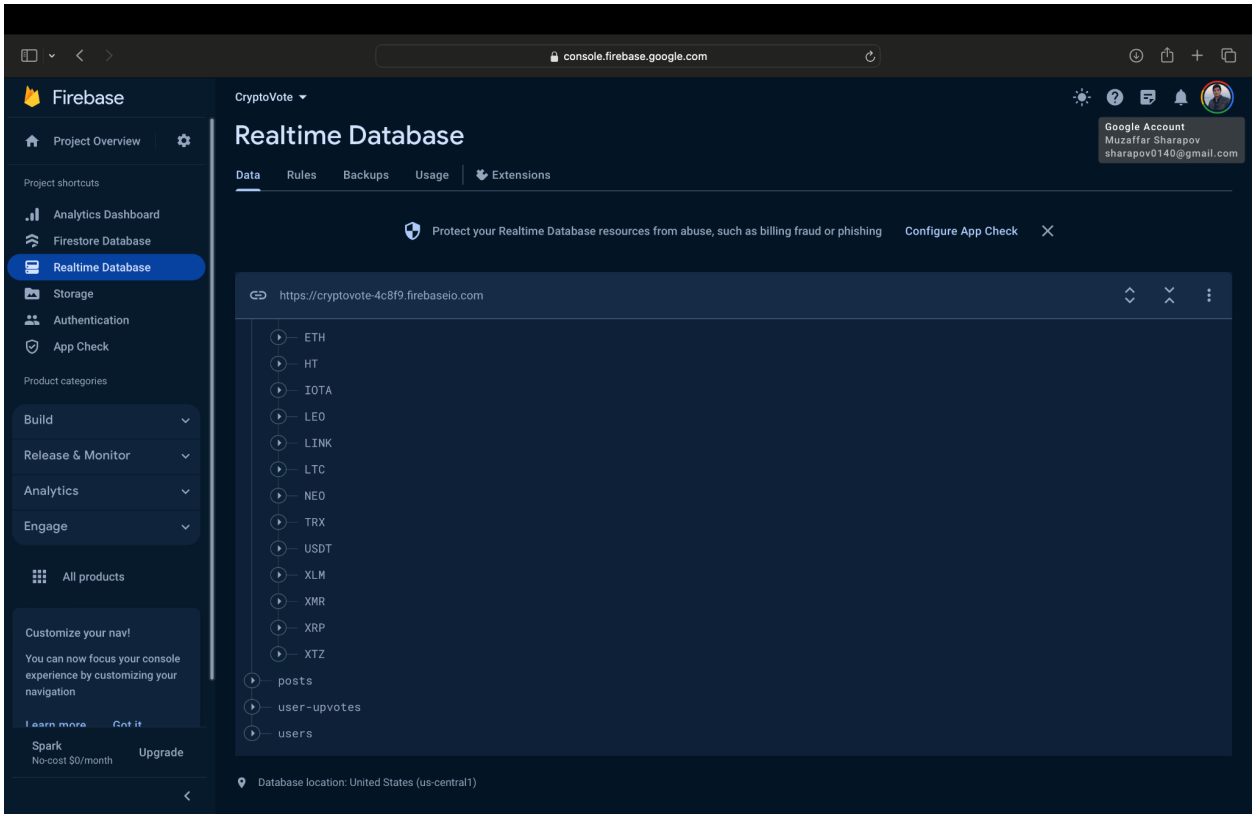
Credits and Acknowledgements

This project incorporates ideas, code snippets, or features from the following sources:

1. **API Calls Implementation**: The methodology for implementing API calls in this project was inspired by the tutorial "A simple iOS app in Swift & Xcode to get weather API" by Ordinary Jack. This tutorial provided insights into API interaction, which were adapted to access data from the CoinGecko website instead of a weather API. [View Tutorial](https://ordinaryjack.medium.com/a-simple-ios-app-in-swift-xcode-to-get-weather-api-super-simple-with-a-counter-example-in-python-94592beff707)

2. **Feature Inspiration - Instagram Clone App**: Certain features in this project, such as user signup, login, and the upvote/downvote (like/dislike) functionality, were inspired by the InstagramCloneApp created by Jedrzej D. This open-source project served as a reference for implementing similar features in our application. [View Repository](https://github.com/jedrzejd/InstagramCloneApp)

I used Firebase to store data:



Description: The file **CryptoVote**.swift contains the CryptoVote class, a subclass of UITableViewController. This class is tailored for displaying a list of cryptocurrency posts in a table view. It includes a search bar for filtering posts and implements the UISearchBarDelegate and FeedCellDelegate for interactive features. Key functionalities of this class involve:
Upvoting Posts: Users can upvote cryptocurrency posts. This feature uses Firebase for user authentication to ensure secure and valid voting actions.
User Interface Setup: The class sets up key UI elements like the search bar in its viewDidLoad method and adjusts for dark mode settings.
Fetching and Displaying Posts: It fetches posts from Firebase, sorts them by the number of upvotes, and displays them in the table view.
Search Functionality: Users can search through the posts using the integrated search bar.
Refresh Mechanism: The class has a pull-to-refresh feature that updates the list of posts.

User Authentication Handling: It includes methods to manage user authentication status, such as showing alerts for email verification and prompts for user sign-up or log-in, ensuring a secure user experience.

The **CryptoList**.swift file focuses on displaying cryptocurrency information. The CryptoList class, inheriting from UITableViewController and conforming to UISearchBarDelegate, manages a table view to present cryptocurrency data. Its main functionalities include downloading JSON data from an API, parsing this data into structured models (cryptoCoinCapJson and cryptoCoinCapJsonPeriod), and dynamically displaying it in the table view. The class also integrates a search bar for filtering cryptocurrencies based on user input, enhancing interactivity. It handles user interface considerations like dark mode compatibility and customizes the appearance of the navigation and search bars. Additional methods like handleRefresh are implemented for updating the table view data, ensuring the information remains current. The class thus serves as a key component in the CryptoVote app, offering a detailed and searchable interface for cryptocurrency statistics.

The **MainTabBarController** is designed to manage the main navigation within the app through a tab bar interface. It customizes the appearance and behavior of the tab bar and the view controllers associated with each tab, providing a structured and user-friendly navigation system within the app.

The **SignUpVC** class in a file is a UIViewController tailored for facilitating user registration in an application using Firebase for backend operations. This class employs UIKit for its user interface components.
User Input Fields: The class includes several UITextField instances for user information input such as email, password, full name, and username. These fields are designed with rounded borders and a light background. The fields are connected to a method (formValidation) that enables the sign-up button when all necessary information is entered.
Sign Up Button: A UIButton, once enabled, triggers the handleSignUp method to register the user using Firebase authentication. The button remains disabled until all input fields are validly filled.
Terms and Privacy Policy: The interface incorporates UILabels and UIButtons to guide users to the application's terms of service and privacy policy, ensuring users are informed about these policies before signing up.
Navigation Management: Additional UIButtons are provided for existing users to navigate to the login screen or to postpone the sign-up process.
View Layout: The configureViewComponents method organizes the UI elements within the view, creating a coherent layout that includes the input fields, sign-up button, terms and privacy information, and navigation options.
Theme Compatibility: The class includes logic to adjust UI elements' appearance based on the current theme (dark or light) set in the device.

The **LoginVC** class in ViewController.swift is a UIViewController that facilitates user login functionality. The class, incorporating UIKit and Firebase libraries, features a user interface for email and password input, login, and account management options.
UI Elements: It includes a logo container, email and password text fields, a login button, and options for account creation and password reset.
Logo Container: Displays an image representing the app logo, centered within a custom UIView.
Text Fields: Two text fields are used for email and password input. They are styled with rounded borders and a light background.
Login Button: This button is enabled when both email and password fields contain text. It triggers the login process when clicked.
Account Management: Additional buttons are provided for users who don't have an account ('Sign Up'), need to reset their password ('Forgot Password?'), or wish to delay login ('Log In later').
Login Process: The handleLogin method attempts to authenticate the user using Firebase. It includes error handling for incorrect credentials and account locking after multiple failed attempts.
Form Validation: The class includes a method to validate form input and enable the login button accordingly.
UI Configuration: The configureViewComponents method organizes the UI elements into a stack view for a structured layout.

The **Post** class, defined in the Post.swift file of the CryptoVote application, encapsulates the properties and behaviors of a cryptocurrency-related post. It integrates Firebase for backend operations and uses UIKit for image handling.
Properties: The class includes several properties to represent a post:
caption: A string for the post's caption.
symbol: A string identifier, likely representing a cryptocurrency symbol.
upvotes: An integer tracking the number of upvotes a post has received.
price: A double representing the price of the cryptocurrency.
price_usd: A string indicating the USD price of the cryptocurrency.
didUpvote: A boolean indicating whether the current user has upvoted the post.
image: A UIImage, potentially for displaying related visuals.
postId: A string identifier for the post.
Initialization: The class initializer takes a symbol and a dictionary. It sets the post's properties based on the dictionary's contents, which likely come from a Firebase database snapshot.
Upvote Logic: Includes methods adjustUpvotes and removeUpvote to handle upvoting functionality. These methods interact with Firebase to update the post's upvote count and the user's upvote status.
adjustUpvotes: Adds or removes an upvote for the post, updates the Firebase database accordingly, and triggers a completion handler with the updated upvote count.

removeUpvote: Removes an upvote from the post, updates the Firebase database, and triggers a completion handler with the updated upvote count.

The **CryptoStats** struct is designed to encapsulate data related to cryptocurrencies, particularly for use in an application that interacts with Firebase for data storage and retrieval. This struct is structured to hold and decode cryptocurrency information, fetched from an external API or a Firebase database.

The **User** class, designed for use in an application with Firebase integration, represents a user entity in the app. It is structured to manage user-related data and interactions within the app's social network-like functionalities.
Properties:
username: A string storing the user's username.
name: A string holding the user's full name.
profileImageUrl: A string containing the URL of the user's profile image.
uid: A string identifier unique to each user.

Initialization:
The initializer takes a uid and a dictionary. It sets the user's properties based on the dictionary's values, which originate from Firebase database snapshots.

The code snippet defines a protocol **SectionType** and two enums, SettingsSection and SocialOptions, intended for managing settings in an application with Firebase integration. These constructs are typically used to organize and display various options in a settings or configuration section of the app.
SectionType Protocol:
This protocol defines a property containsSwitch indicating whether a section contains a toggle switch.
It also requires conforming types to provide a description (via CustomStringConvertible), which is used for displaying the section's title or label.
SettingsSection Enum:
Represents different sections in the app's settings. Currently, it includes only one case, Social.
Implements CustomStringConvertible to provide a textual description for each section, which aids in user interface rendering.
The description for Social is set as "Settings".
SocialOptions Enum:
Enumerates various options available under the Social section of settings.
Cases include report, termsOfUsage, termsOfPrivacy, and logout.
Implements SectionType, meaning it provides a containsSwitch boolean (always false here, indicating these options don't have toggle switches).

The description provides text for each option. Notably, the logout option dynamically changes to "Log In" or "Log Out" depending on the user's authentication status in Firebase.

The **UIView+Layout.swift** file contains an extension for UIView, a fundamental class in UIKit. This extension provides convenience methods to simplify the process of setting up Auto Layout constraints for views.
Purpose: The primary purpose of this extension is to streamline the layout configuration of UI elements within an app, reducing the boilerplate code typically associated with Auto Layout.
Methods in the Extension:
anchor(...): This method allows setting top, bottom, leading, and trailing constraints relative to other views, as well as setting fixed width and height. It returns an AnchoredConstraints struct containing the active constraints. This method is versatile, allowing for comprehensive layout specifications with minimal code.
fillSuperview(...): This method sets the view to fill its superview's bounds with optional padding. It is particularly useful for views that need to occupy the entire space of their parent view.
centerInSuperview(...): Centers the view in its superview and optionally sets a specific width and height.
centerXInSuperview() and centerYInSuperview(): These methods center the view either horizontally or vertically within its superview.
constrainWidth(...) and constrainHeight(...): These methods apply fixed width and height constraints to the view.
Struct AnchoredConstraints:
This struct holds references to active layout constraints. It includes optional properties for top, leading, bottom, trailing, width, and height constraints.
Auto Layout Management:
Each method sets translatesAutoresizingMaskIntoConstraints to false, which is necessary for Auto Layout to work correctly. It then activates the constraints that have been set up.

The **CryptoVoteCell** class, a subclass of UITableViewCell, is designed to display cryptocurrency-related data in a table view, particularly in an application that involves voting or rating cryptocurrencies.