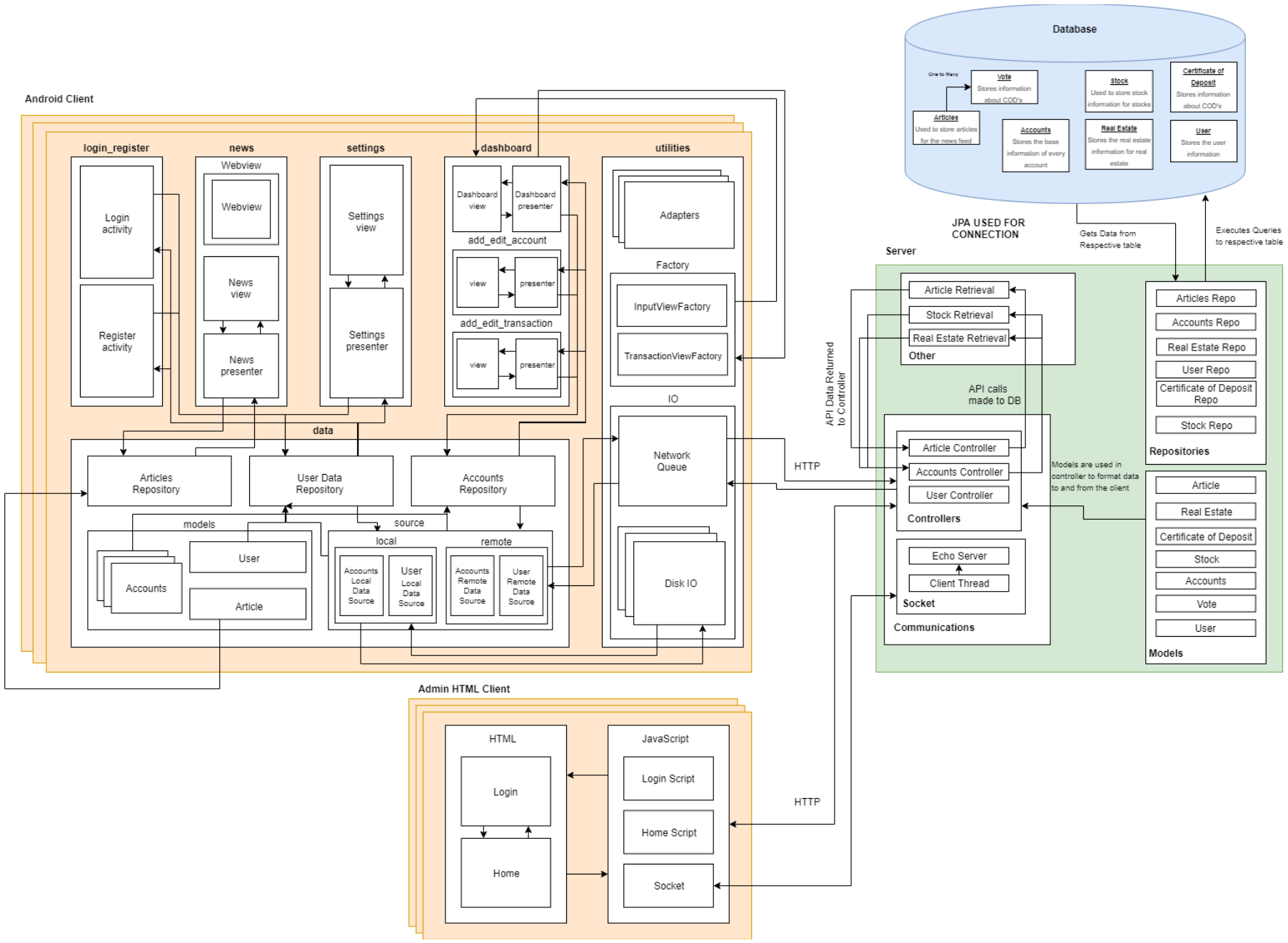
BLOCK DIAGRAM

TEAM 10
Griffin Stout
Kyle Marek
Michael Davis
Zachary Mohling

NetWorthr


Android Client
login_register
Login activity
Register activity
news
Webview
Webview
News view
News presenter
settings
Settings view
Settings presenter
dashboard
Dashboard view
Dashboard presenter
add_edit_account
view
presenter
add_edit_transaction
view
presenter
utilities
Adapters
Factory
InputViewFactory
TransactionViewFactory
IO
Network Queue
Disk IO
data
Articles Repository
User Data Repository
Accounts Repository
models
Accounts
User
Article
source
local
Accounts Local Data Source
User Local Data Source
remote
Accounts Remote Data Source
User Remote Data Source
HTTP
Admin HTML Client
HTML
Login
Home
JavaScript
Login Script
Home Script
Socket
HTTP
Database
One to Many
Vote
Stores information about COD's
Articles
Used to store articles for the news feed
Accounts
Stores the base information of every account
Stock
Used to store stock information for stocks
Real Estate
Stores the real estate information for real estate
Certificate of Deposit
Stores information about COD's
User
Stores the user information
JPA USED FOR CONNECTION
Gets Data from Respective table
Executes Queries to respective table
Server
API Data Returned to Controller
Article Retrieval
Stock Retrieval
Real Estate Retrieval
Other
API calls made to DB
Article Controller
Accounts Controller
User Controller
Controllers
Models are used in controller to format data to and from the client
Echo Server
Client Thread
Socket
Communications
Articles Repo
Accounts Repo
Real Estate Repo
User Repo
Certificate of Deposit Repo
Stock Repo
Repositories
Article
Real Estate
Certificate of Deposit
Stock
Accounts
Vote
User
Models

Server

The server application is a restful API. There are four main components of the server application. The models are essentially objects that are formatted to be inserted into their respective table. Each model has its own table in the database. The controllers handle requests that are received from the client and returns data from the database via the repositories.There are three controllers to handle requests regarding accounts, articles, and users. The repositories are used to make queries to the database and grab data for the controllers. Every account has a table in the database. In order to use API's to get real time stock data, articles, and real estate prices, we made helper classes called ArticleRetrieval, StockRetrieval, and RealEstateRetrieval to make calls to the API's we are using and return the data.

Client

The client is an Android app. There are six main modules login/registration, news, settings, dashboard, utilities, and data. Login/registration contains the activities for users to login as well as register a new account. The news module contains the news fragment to view article cards as well as a web view to view each article individually. Settings has its own fragment to display the user's information as well as change the password, email, and logout. Dashboard also contains a fragment which displays the user's accounts as well as adding or editing accounts and transactions. The utilities module contains the utilities that the fragments and views need to function correctly. Finally, the data module contains user models as well as repositories for articles, user data, and accounts.

MVP Design Pattern

The Model-View-Presenter design pattern is the most common design pattern used with the Android framework. We implemented this design pattern to separate our data and business logic layers from the views. A passive view constraint decoupled our classes, improved our ability to test, and enhanced readability. This nonfunctional requirement allows a code base to grow and change.

Our data layer holds our account models and the repositories for accounts and user data. The views and presenters are agnostic to the business logic, and interact with the data model only through an interface. Whether the data layer does CRUD operations locally or with a remote machine, that doesn't involve our passive views. Once example that this approach improved the development process was when we implemented the modal screens (add_edit_account and add_edit_transaction). These views must be generated since they're different for every type, and until we implemented the MVP design pattern, it was too challenging (and too ugly) to involve the type-specific logic.

Tables and Fields

Accounts

- AccountID
- label (what user names their assets and liabilities)
- type (account type)
- isActive(whether the user deleted it or not)
- transactions(any change to the accounts value)

Articles (one to many with votes)

- userId
- url
- title
- description
- urlToImage
- isActive
- keyword(word attributed to the article to load for some users)

Stocks

- accountId
- ticker

Users

- lastname
- firstname
- email
- salt(random bytes appended to password for security)
- password(hashed password)
- type(admin or not)

Vote (many to one with article)

- Article_id(the article this vote is attributed to)
- userId(who voted)
- vote(1, 0, or -1)

Real Estate

- accountId
- address
- city
- state

Certificate of Deposit

- accountId
- maturityDate