

OOAD Project 6 - Final Submission

Title: KALE, a game-based education app for quizzes

Team members: Janani Selvan, Keyur Shirgaonkar, Veena Prasad

1. Project Summary:

KALE, an e-learning platform for educational purposes that aims to perform formative assessment to gauge student's knowledge. A cloud hosted database application for a Question and Answer application with connected devices played by distributed users using the web interface. The application allows to host and create fun and with multiple players. Firebase realtime database application is used to sync data across all the clients in real time.

2.Tools and Technologies:

Programming Languages	:	Python, Flask
Web Technologies	:	CSS, HTML, React JS ,Firebase, Java Script
Database	:	Postgress
Cloud/Container Technologies Tools	:	Virtual machines, Redis
Process	:	Agile Software Development

3.Goals and Responsibilities:

The main goals of the system is to build a rich and collaborative application allowing multiple users to obtain a secure firebase realtime database. The highlighted responsibilities of the application is to persist the data locally and provides the players to get a real time experience without compromising on responsiveness.

4.Game Description:

A quiz application with two types of user where the player user joins the game with the unique pin generated by the system using singleton pin generator. The admin user hosts various types of quizzes maintained in a factory pattern. The admin user begins the game when an adequate number of players have joined the game using nicknames. The questions are spanned for a certain period of time and changed by the admin. On completion of the quiz the quiz leaderboard is presented to all types of users by listing the nickname and cumulative points.

5. Requirements:

API and functionality:

1. /api/admin/create_user

- Validate permissions
- Validates whether the username exists
- Validates the password with the one stored in the database for the corresponding user
- Add user info into the admin_user table
- If user already present, return the same

Username	Password

2. /api/admin/login

- Validate permissions
- Authenticate user and return 200 if valid.
- If user is invalid, return 404 - user not found.

3. /api/player/create_user - now

- Add the corresponding nickname of the user
- entered answers by the user
- scores will be stored for the respective players
- game_id belonging to the respective quiz
- last_correct answer is added to the database
- corresponding_answer is added to the database
- If user already present, return the same
- Payload - {pin: <unique guid>}
- Fetch the quiz id associated with pin from Redis Key value store
- Update the quiz id in player user table
- If user already present, return status code 202

nickname	entered answer	Score	game_id	streak	last_correct	answer

4. /api/user/login - now

- Validate Payload - {pin: <unique guid>}

5. /api/admin/create_quiz - via seed script

- Creates a quiz table
- Adds a category column to it
- Adds a Difficulty column to it

Category	Difficulty

Text Image Video Graphs

6. /api/admin/<quiz-id>/edit_quiz - via seed script

- List of Questions
- Update fields in the questions/metadata table using the quiz id from the admin

7. /api/all/<quiz-id>/leaderboard - now

- Sorting the player user table (PySpark) and publishing the top 3 scores.

8. /api/user/<quiz-id>/<qn-id>/post-answer - now

- Post the answer added by the student
- Adding the corresponding correct answer to that respective question
- Posting the question id for the corresponding answers
- Update the percent of students answered each option in redis key-value store

answers	correct	question_id

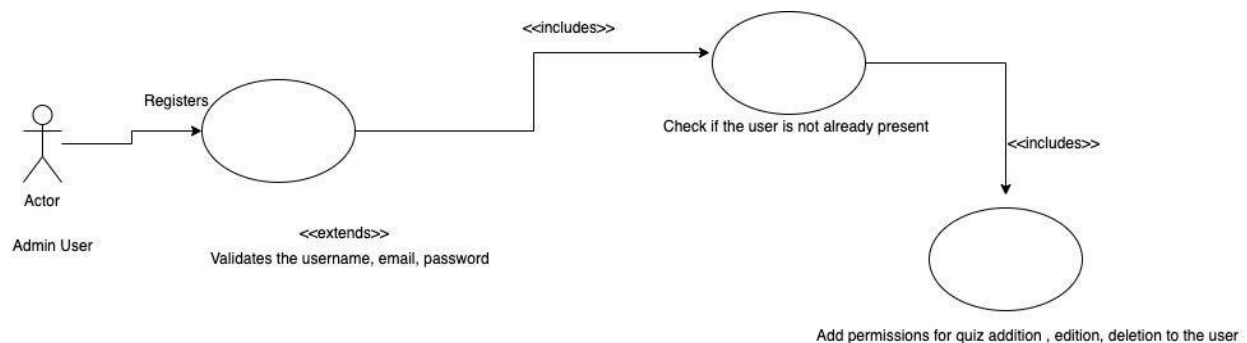
Data that'll be stored in Key value store (Redis):

1. (PIN, (quiz_id, created_by))
2. (question_id, (op1 %, op2 %, op3 %, op4 %))

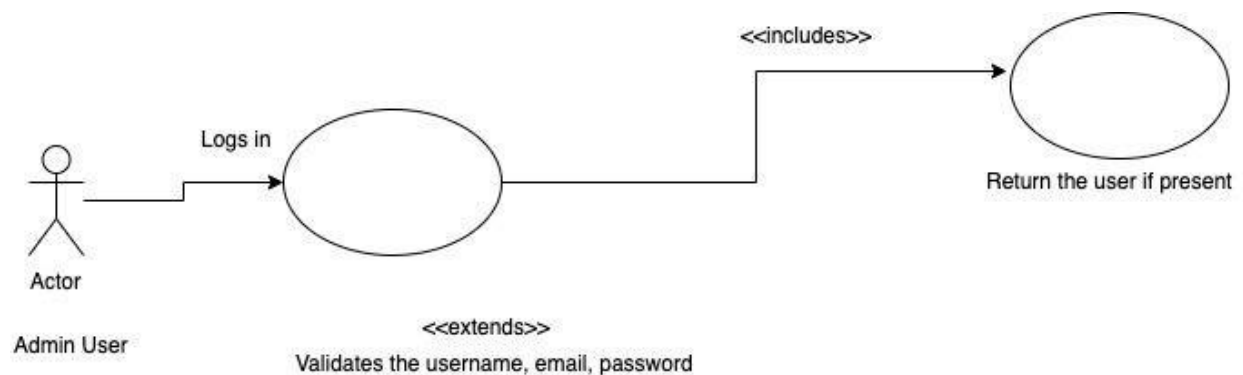
Difference between Project 4 and Project 6 functionality:

We have changed the schema for project 6 although we have kept the API calls the same. For admin creation of user we have changed schema to include username and password only. Similarly for players we have changed schema to include nickname, entered answer, score game id , streak , last_correct, answer. Similarly, while creating quiz we have changed schema to include category and difficulty. Lastly for verification of the answers we have changed the schema to include user answers, correct answers and their respective question id.

6) Use Cases:

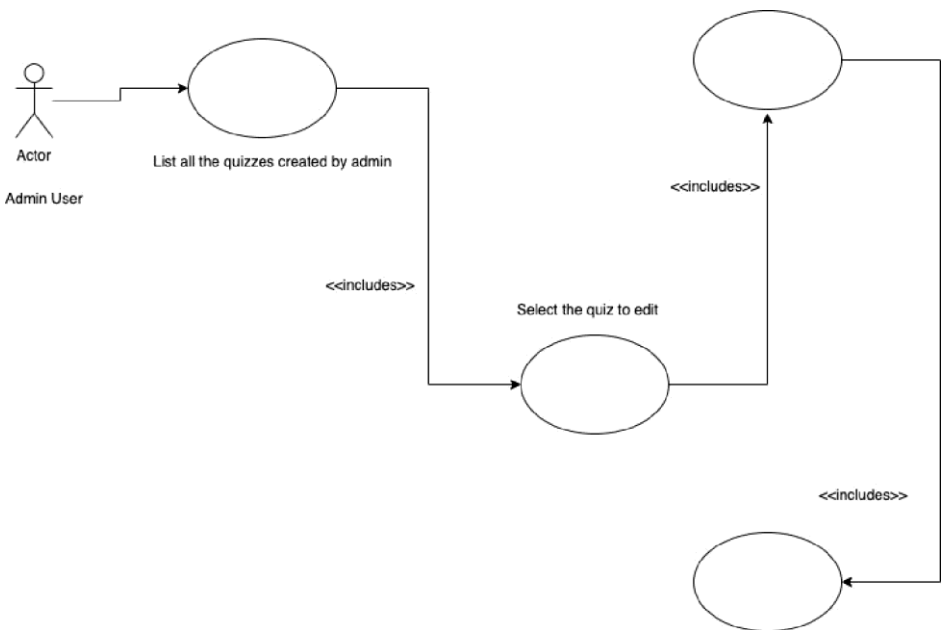


Admin User Registration Scenario



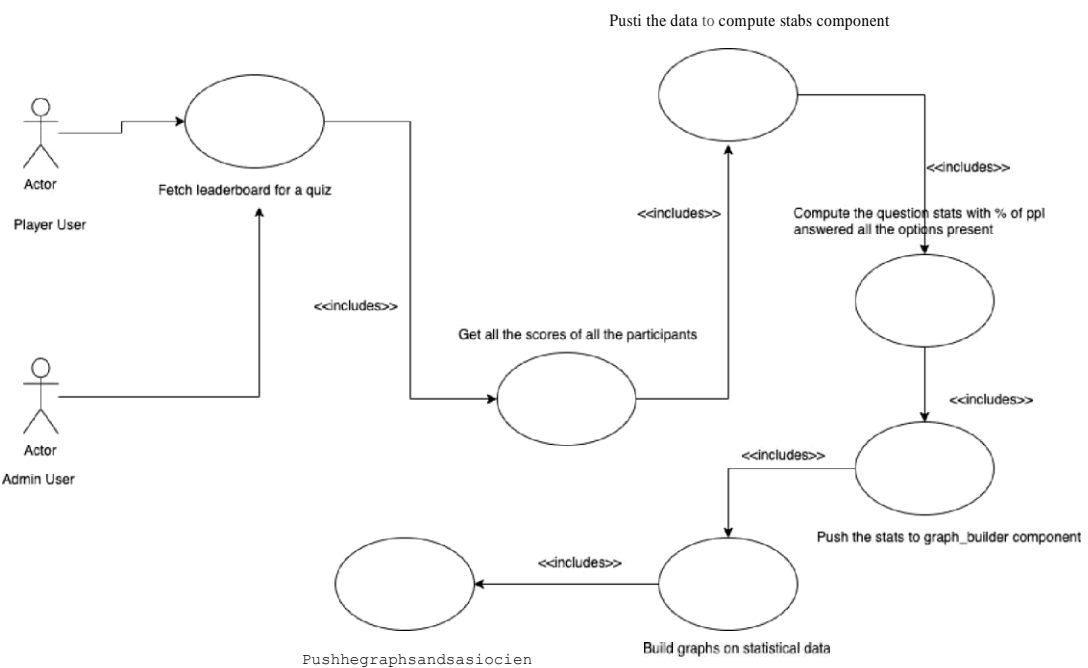
Admin User Login Scenario

Make amendments to the questions

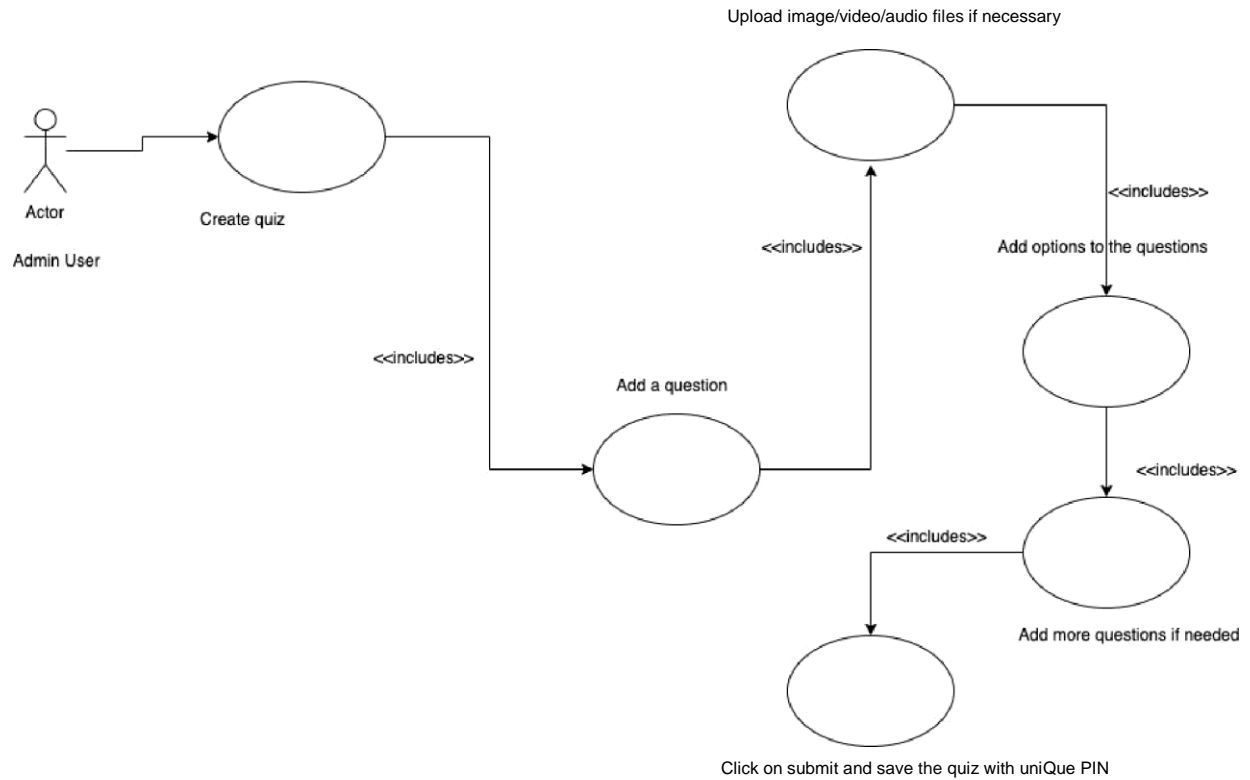


Click on submit and save the quiz

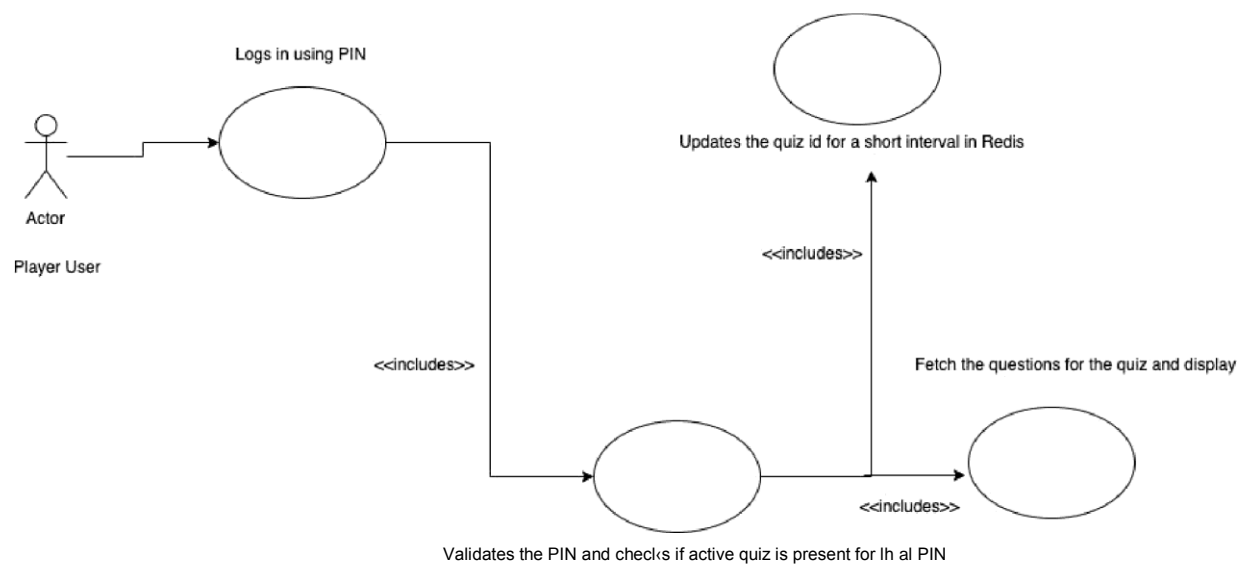
Quiz Edit Scenario



Quiz Leaderboard Scenario



Quiz Create Scenario



Player User Login Scenario

7) Class Diagram and Pattern discussion:

Difference between Project 4 and Project 6 Diagram:

As per our requirement we have added the necessary changes to the class diagram. For managing user services and game services we have made use of Chain of Responsibility pattern. We are using Firebase observer pattern. We are using Model View Controller Pattern for different views, model and services within our kale game. We have made use of Facade pattern instead of Factory pattern for question creation and answer creation.

MVC Pattern:

Module: This comprises of module config having configuration variables in ways of maintaining the config map having the environment variable in the settings and details maintaining external components like hostname, password, username and credentials to set up the environment variables. We setup the flask environment. Config object of flask environment is saved in config file and read by the flask application.

View: The view exposes the route in the file where the app instances are created. GET, POST, DELETE, PUT for all the entities are organized here.

Controller: The business logic is maintained in the controller i.e. maintains a large collection of objects for instance SQLAlchemy. SQLAlchemy is the ORM Layer which maps the class object to the database to enhance the object oriented design of development. Services: These contain the core business logic connecting other components. This comprises of the quiz, quiz question components, user authentication are captured for the entities.

DB Migration: This is used to track and maintain the version control for database tables. We create files and each file comprises the actions performed on tabled. Each file has a unique GUID maintained in a table which helps in rolling back and navigation. Upgrade and downgrade method follows the schema up gradation and maintenance of database before production and services. For update we need an existing object which we make changes to and accordingly update the object.

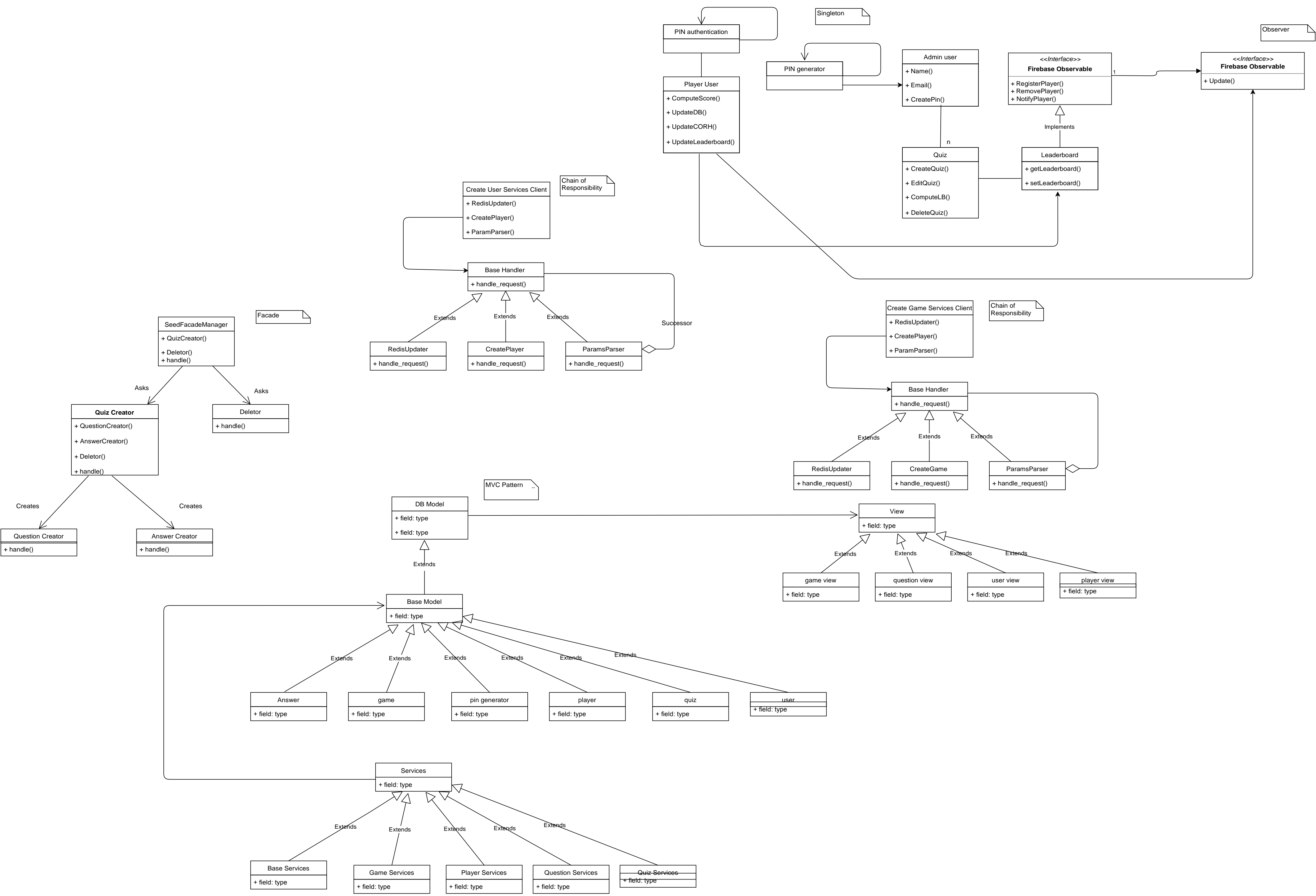
SQLAlchemy is layer of mapping over the SQL queries. Basically everything which can be done using SQL can be done using Object Relational Mapper using Object Oriented Paradigm. The relation name required for mapping comes from migration file.

Singleton Pattern:

The singleton pattern was implemented for Pin Authenticator. We need to have only one instance of our class for every single quiz and a single DB connection shared by multiple objects to store the unique pin generated for each quiz. There can be a validator or error manager in an application that handles generation and validation of unique pin

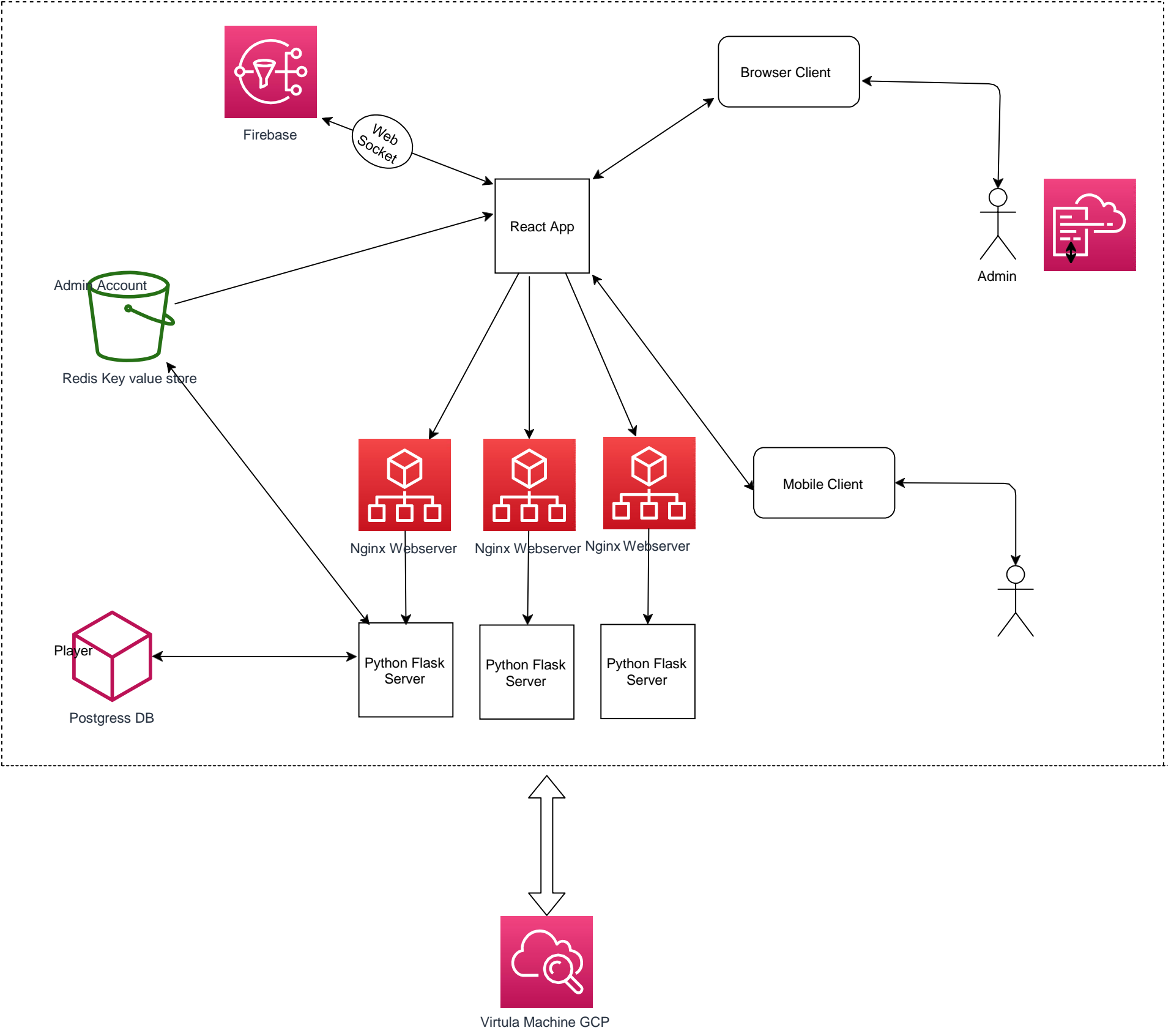
Decorator pattern:

This pattern creates a decorator class which wraps the original class and providing and adding CORS headers to the api responses



8) Architecture and System Design:

There is no substantial change in the architecture of our application. However we have changed a few technology usages. The admin user request originates from your browser. All the web request goes through GCLB (Google Cloud Load Balancer). This proxy the request to Nginx web servers that scale from 1 to n (n web servers). We are making sure Nginx serves the default port with our web builds. We also make sure it correctly handles Flask requests. Here the requests are forwarded on `ip/app/` to `localhost:3000/` where our flask app would be up and running. React apps can be hosted by extracting the build and putting the path in location `/`. Size of Python AppServer will be scaled based on requirement and load. AppServer will pull out statistics for the leaderboard where the players are listed with their nickname and the cumulative points in descending order. The generated statistics are pushed to Google Cloud Storage and hosted in virtual machines. webserver also persists data in persistent database store like Redis. The database store here is Postgress database. Firebase server keeps listening to the channels on Redis key value store. The Firebase server is configured such that when there is any new change in Redis then it handles the change and push the real time updates to the clients browser window.



9) Key Design process elements:

The following design principles were explored during the development of our OO semester project ;


- 1) Coding to interface
 - 2) Encapsulate what varies
 - 3) Open closed principle
 - 4) Principle of least knowledge
 - 5) Favour delegation over inheritance
- Adding interfaces in the model layer to support database CRUD operations helped in code maintenance and segregation of development code.
 - While adding cors header to responses the principle of open closed was explored in the way of implementing Decorator pattern which eased the request and response management.
 - The process of question creation was made easy by encapsulating the question type creation and providing answer options by grouping the classes using the Facade pattern and exposing a thin layer to the client.

10) Challenges:

To reflect on anything that was particularly difficult or anything you would have done differently.

1. Resolving Circular import dependency on python packages when app instance was required in instantiating sqlalchemy classes and serialization packages.
2. Building relationships among orm classes built on top of the tables
3. Parsing the nested child attributes using serialization packages
4. Handling CORS requests in Flask server
5. Maintaining database versioning.
6. Adding design patterns particularly at the model layer when the models are closely coupled with the database schema.

11) Changes in Mockups:




Member Login

Email

Password

LOGIN

No Account? [SignUp](#)



SignUp

Enter Email

Password

Password Confirmation

SIGN UP

Already Have An Account? [Login](#)

Enter Quiz ID

Quiz Pin

JOIN GAME

Login To Create Quiz

Choose a game to play



General Knowledge

Difficulty
easy

Question #
10



General Knowledge

Difficulty
medium

Question #
10



General Knowledge

Difficulty
hard

Question #
10

Waiting For Players To Join

QuizCode: 75856

Start Game

12) Code Reference

Postgres tutorial - <http://www.postgresqltutorial.com/>

Sqlalchemy tutorial - <https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/>

Flask tutorial - <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

Flask app tutorial - <https://realpython.com/tutorials/flask/>

Flask redis tutorial -

<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xxii-background-jobs>

Git tutorial - <https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>

React app tutorials - <https://reactjs.org/tutorial/tutorial.html>

React app template - https://github.com/jamaspy/kahootz_client

Firebase tutorial - <https://www.raywenderlich.com/3-firebase-tutorial-getting-started>

React firebase tutorial -

<https://hackernoon.com/the-easiest-way-by-far-to-build-a-real-react-firebase-web-app-5dc6fa6f1b61>

React firebase sample -

<https://blog.bitsrc.io/building-a-todo-app-in-react-with-firebase-and-redux-ba3ab53a671b>

Original code - Backend server developed using Postgres, Flask, Redis and Python. Frontend server with HTML, CSS, and firebase JS connection.