ARCHITECTURE AND DESIGN

PulseCheck

Trinity Dhillon, Daniel Sirias, Michael Campos, Brandon Budhan, Peter Georgaklis, & Zaira Garcia

2025-03-25
v1.0.0

Table of Contents

1. Introduction

1.1 Purpose

This document provides a detail-oriented architecture and design specification for **PulseCheck** outlining **how** the system will be implemented to meet the defined requirements in the software specification document and respect the project plan schedule. This document includes system architecture diagrams similar to the C4 model [2], design patterns, and implementation strategies.

1.2 Scope

The system is a web application that facilitates real-time polling sessions, where results are visible in real time. The application is fully web-based, and is able to be accessed on modern web browsers on both desktop and mobile devices. The system is built on a serverless architecture using Firebase, which provides deployment of the system for system-end users to access, database management, and cloud storage. Key capabilities that are in-scope for the system include:

- **User Authentication** (Email/Password, Google Sign-In via Firebase Authentication)
- **Poll Creation & Management** (Polls consisting of multiple-choice questions are saved and can be accessed at any time)
- **Real-Time Voting and Results** (Leveraging Firestore listeners for instant updates)
- **Responsive UI** (Designed with Material UI components for consistency and optional experiences on any device)
- **Results and Analytics Display** (Poll statistics available in real time, such as vote percentages, response counts, and graphs)

In regards to constraints, the following areas are out of scope for the system's intended deliverables:

- **Open-Ended/Subjective Questions** (Only multiple-choice, multi-select, and ranking-poll questions are supported)
- **Third-Party Integration** (External API integration is not included)
- **Offline Functionality** (Internet connection is required for real-time updates)

2. References

[1] "IEEE Standard 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems." Institute of Electrical and Electronics Engineers, 2000, github.com/davideuler/architecture.of.internet-product/blob/master/F.10%20papers%20for%20architect/ieee%20std%201471-2000.pdf.

[2] Brown, Simon. "C4 Model for Software Architecture." C4 Model, 2024, c4model.com.

[3] "Best Practices for Cloud Firestore." Google Firebase Documentation, Google, 2024, firebase.google.com/docs/firestore/best-practices.

[4] "GitHub Actions Documentation." GitHub Docs, GitHub, 2024, docs.github.com/en/actions.

3. System Overview

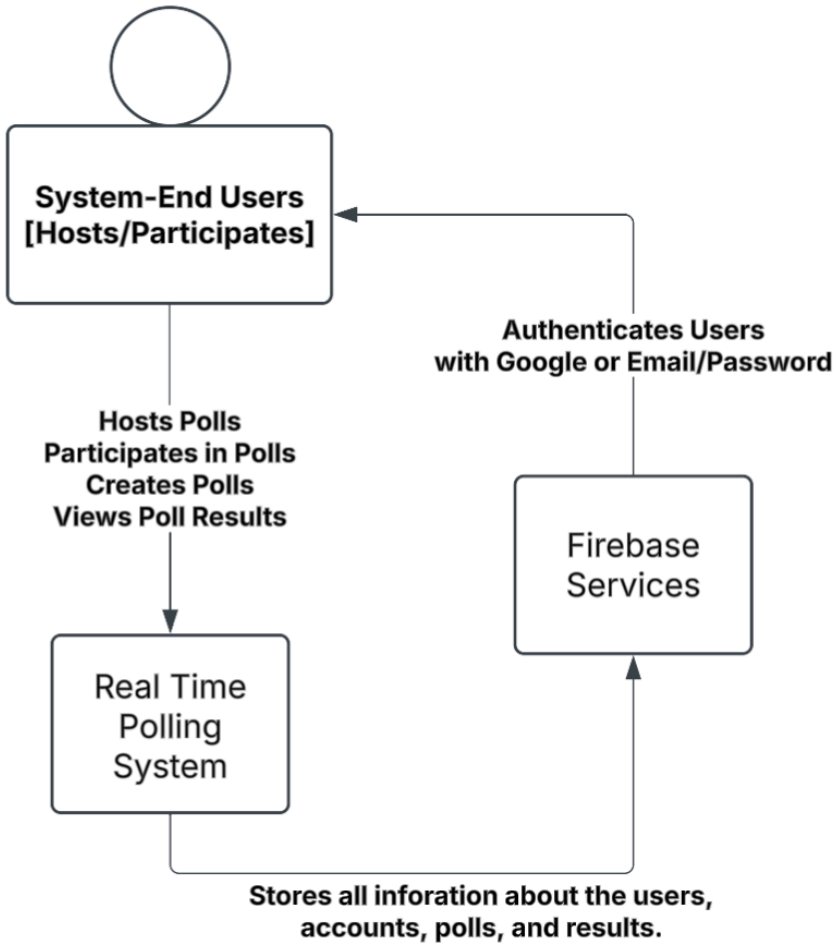

Figure C.1: System Context [2]

In Figure C.1, the real-time polling system is an interactive quiz and polling web application designed for large lecture halls. It enables system-end users to create polls, manage responses, and analyze results in real time. The application is designed with a responsive frontend built with React and Material-UI, as well as a serverless backend utilizing Firebase's services, which contains services such as Firestore, Authentication, and Cloud Storage. Its key features include the poll creator, which allows system-end users to create a poll that can be hosted at any time, and the real-time polls, which allows for a high level of interaction.

The system’s notable key features are:

- **Real-Time Polling**: Participants can select answer options, and results are updated instantly.
- **User Authentication**: Users can sign-in securely using Firebase Authentication.
- **Custom Polls:** Users are able to create multiple-choice polls that can be reused, edited, and hosted at any time.
- **Live Data Visualization**: Poll results and statistics are displayed dynamically with accompanying graphs.
- **Responsive Design**: Ensures consistent experience across all devices.

4. System Architecture

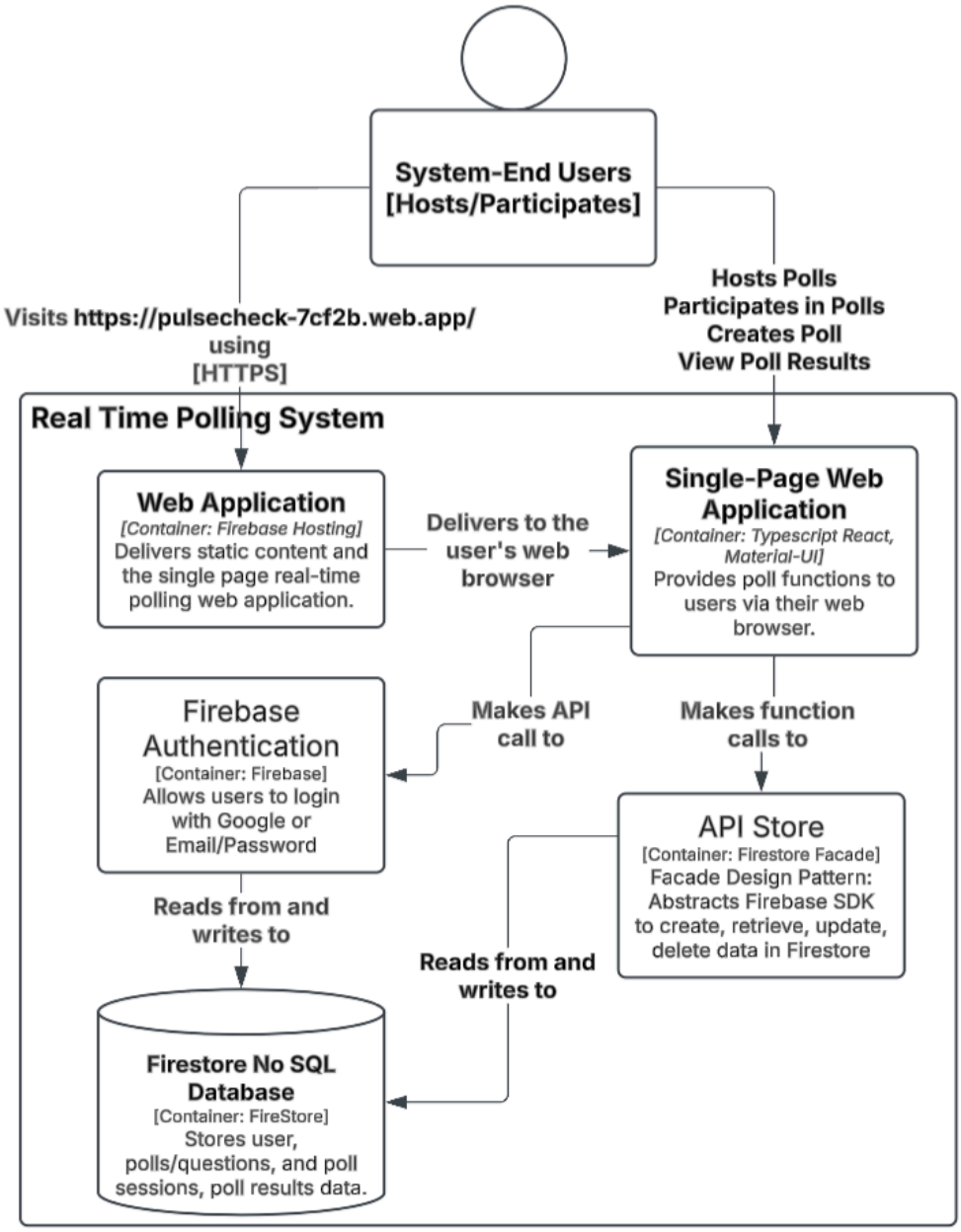

Figure C.2: Component Diagram [2]

The system follows a **client-server architecture**, where the **Single-Page Application (SPA)** serves as the frontend, and **Firebase's** functions as the backend. The key components are illustrated in Figure C.2.

Frontend

The frontend is built with React (TypeScript) and styled using Material-UI. The frontend is deployed via **Firebase Hosting** and is accessible at https://pulsecheck-7cf2b.web.app/.

To manage interactions with Firebase efficiently, the frontend includes a custom-defined **API-Store** that follows the facade design pattern. This store abstracts complex Firebase SDK operations, providing a simplified interface for the development team to handle poll sessions, created polls, and poll results.

Backend

The backend consists of Firebase's Firestore, a NoSQL database that stores and manages poll data, including poll sessions, questions, responses, and results.

Authentication

User authentication is managed through **Firebase Authentication**, allowing users to:

- Log in securely
- Create and manage polls
- Participate in real-time poll sessions.
- View poll results.

System Workflow

System-end users access the real-time polling system by navigating to https://pulsecheck-7cf2b.web.app/ using their web browser. The SPA is served through **Firebase Hosting** and interacts with Firebase through the **API Store**, which ensures a smooth user experience for creating ,managing, and participating in poll sessions.

5. Key Modules and Components

5.1 Components

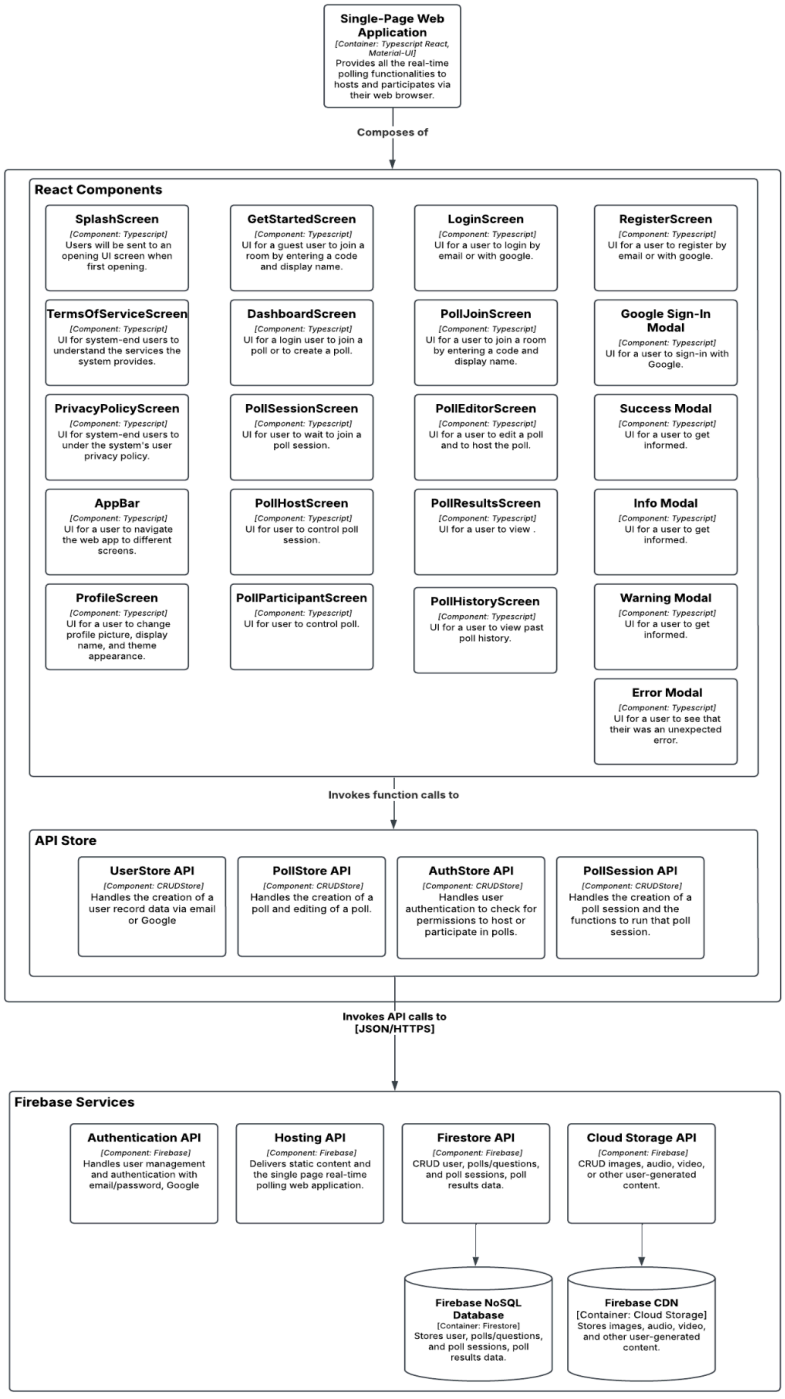

Figure C.3: Components Diagram [2]

5.1.1 Frontend Components

- **SplashScreen [Component: Typescript]** - Users are first sent to this screen when first entering the website. Allows the users to navigate to other parts of the website, such as the GetStarted screen, Terms of Service screen, Privacy Policy screen, Login screen, and Register screen. It also displays information about the system.
- **GetStartedScreen [Component: Typescript]** - User can join a poll session by typing in a valid room code and display name.
- **LoginScreen [Component: Typescript]** - Displays a login form that allows the user to sign in by email or to continue with Google.
- **RegisterScreen [Component: Typescript]** - Displays a registration form that allows the user to register by email or to continue with Google.
- **TermsOfServiceScreen [Component: Typescript]** - Displays the terms of service of the system to the user.
- **DashboardScreen [Component: Typescript]** - Displays the option to join a poll or to create a poll. And displays user’s recently created polls if any.
- **PollJoinScreen [Component: Typescript]** - Displays the form for authenticated users to join a poll session.
- **Google Sign-In Modal [Component: Typescript]** - Displays a new window to sign in with the user's Google account.
- **PrivacyPolicyScreen [Component: Typescript]** - Displays the privacy policy of the system to the user
- **PollSessionScreen[Component: Typescript]** - Displays progress for waiting users to join a poll session.

- **PollEditorScreen[Component: Typescript]** - Displays the UI for users to create a poll and edit. Users can change the title of the poll, create and delete questions, set poll settings and question settings, and host the poll.
- **Success Modal [Component: Typescript]** - Displays pop up window of a success message to the user such as successful completion of the poll session.
- **AppBar[Component: Typescript]** - Displays a constant app bar allowing users to navigate the web app. Guest users are also given the ability to register and login. Login users are able to navigate to the dashboard, poll editor and join poll sessions through the appbar. During a poll session users are unable to leave the poll with the appbar.
- **PollHostScreen[Component: Typescript]** - Displays host screen and shows UI controls of the poll session such as moving to the next question for participants to answer.
- **PollResultsScreen [Component: Typescript]** - Displays user's poll results such as responses for questions in the poll, indicating whether it is correct or wrong, along with statistics of overall user scores such as median score, highest/lowest score, and quartiles.
- **Info Modal [Component: Typescript]** - Displays pop up window of information to the user.
- **ProfileScreen [Component: Typescript]** - Displays profile information and allows the user to change profile picture, display name, and system theme appearance.
- **PollParticipantScreen [Component: Typescript]** - Displays a poll session and allows the user to submit poll questions answers.
- **PollHistoryScreen [Component: Typescript]** - Displays user's past polls user participated in along with their scores.

- **Warning Modal [Component: Typescript]** - Displays pop up window of a warning message to users such as leaving in the middle of a poll session.
- **Error Modal [Component: Typescript]** - Displays pop up window of an error message to the user such as invalid user input.

5.1.2 API Store

- **UserStore API [Component: CRUDStore]** - Handles the creation of a user record data via email or Google
- **PollStore API [Component: CRUDStore]** - Handles the creation of a poll and editing of a poll.
- **AuthStore API [Component: CRUDStore]** - Handles user authentication to check for permissions to host or participate in polls.
- **PollSession API [Component: CRUDStore]** - Handles the creation of a poll session and the functions to run that poll session.

5.1.3 Firebase Services

- **Authentication API [Component: Firebase]** - Handles user management and authentication with email/password, Google.
- **Hosting API [Component: Firebase]** - Delivers static content and the single page real-time polling web application.
- **Firestore API [Component: Firebase]** - Handles CRUD user, polls/questions, and poll session, poll results data.
- **Cloud Storage API [Component: Firebase]** - Handles CRUD images, audio, video, or other user-generated content.
- **Firebase NoSQL Database**

    - Stores user data such as polls/questions, poll sessions, and poll results.

- **Firebase CDN**
    - Stores uploaded images, audio, video, and other user-generated content.

5.2 Backend Structure (Firestore Data Model)

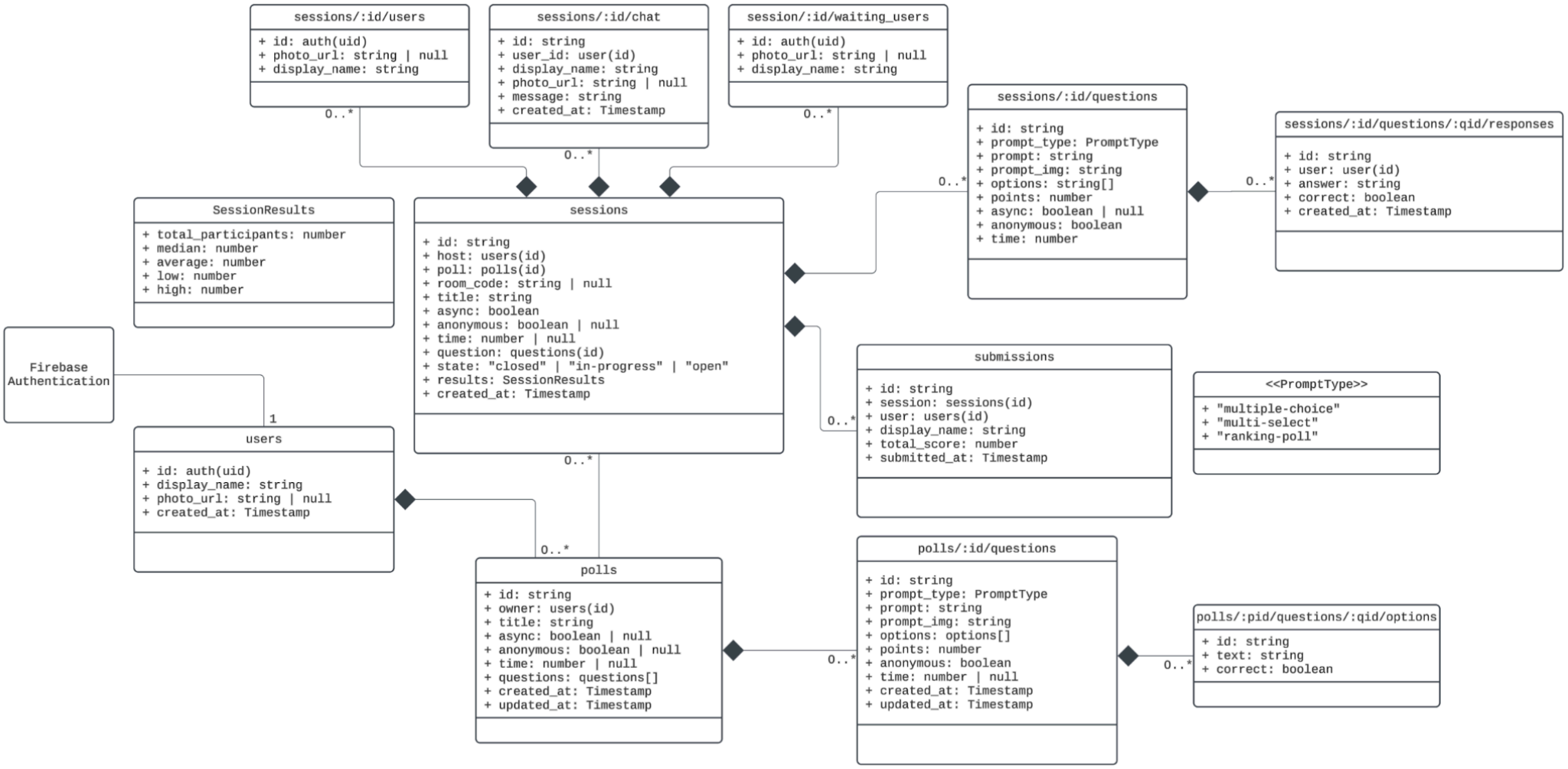

Figure D.M

Figure D.M visualizes a relational database model that represents the structure of Firebase's Firestore service, which stores collections of documents that are to be referenced in the application. The database model adopts UML conventions, indicating relationships and X possible number of instances of any given data model. The key components are as follows:

Users Collection (users)**:** Stores user details for individual user profiles.

- **display_name: string** is the user's chosen preferred name. This is the name which auto-fills as the "display name" field when the users want to join a poll in the /poll/join page.
- **photo_url: string | null** is the URL stored in Firebase's Cloud Storage that holds the user's profile picture.
- **created_at: Timestamp** is the date and time the user's account was created for this system.

Polls Collection (polls)**:** Stores details about user-created polls that can be hosted at any time.

- **title: string** is the name of the poll assigned by the user (e.g. CSC 190 Quiz).
- **async: boolean** indicates whether each question in the poll is answered simultaneously or is taken at each participants' pace.
- **anonymous: boolean** indicates whether questions within the poll are anonymous or not.
    - Docs in the questions subcollection have their own anonymous field value. The anonymous field value in **polls** overrides this for every question if true.
- **time: number | null** field indicates the amount of time allocated for all questions.
    - Overrides all docs in subcollection questions.time field value, similarly to the anonymous field.

- **questions: DocumentReference<Question>[]** is an array which contains references to questions in the questions subcollection to maintain their ordering.
- **created_at: Timestamp** and **updated_at: Timestamp** indicates the timestamp at which the poll was created and last updated.

Poll Questions Collection (polls/:id/questions)**:** Stores individual questions within polls.

- **prompt_type: PromptType** that determines the category of an asked question.
    - **PromptType** = "multiple-choice" | "multi-select" | "ranking-poll"
        - **multiple choice**: one correct answer
        - **multi-select:** multiple correct answers
        - **ranking-poll:** no correct answer, strictly for recording user perception
- **prompt: string** is the question being asked.
- **prompt_img: string** is the URL to an image to be portrayed along with the question if available. Similarly to the *user's photo_url,* this link is where the image is stored in Firebase's Cloud Storage
- **options: DocumentReference<PromptOption>[]** is an array of references to docs in the options subcollection to maintain order of the question options.
- **points: number** indicates how many points the question is worth.
- **anonymous: boolean** indicates if user responses to the question are anonymous or not.
- **time: number | null** indicates whether the question is timed and how long a participant has to answer the question in milliseconds
- **created_at: Date** and **updated_at: Date** indicate when the date the poll was created and last updated.

Poll Question Options Collection (polls/:id/questions/:qid/options)**:** Stores multiple choice options within a question.

- **text: string** indicates the text describing the answer option
- **correct: boolean** indicates whether the option is correct.

Poll Sessions (sessions): Stores poll sessions with snapshots of poll data, ensuring immutability of running poll sessions and differentiates between distinct sessions of a given poll.

- **host: DocumentReference<User>** indicates the user who is the host of the poll.
- **poll: DocumentReference<Poll>** indicates which poll this session is based on.
- **room_code: string** indicates the 6 character key necessary to join a given poll session.
- **title: string** is the title of the poll session which is copied from the user's poll title.
- **async: boolean** indicates whether each question in the poll by participants simultaneously or is taken at each participants' pace.
- **anonymous: boolean** indicates if questions within the poll are anonymous or not.
- **time: number | null** field indicates the amount of time allocated for all questions.
- **question: questions(id)** is the current question displayed to users.
- **state: "closed": "in-progress" | "open"** is the current status for the poll session. If the session is closed, then no users are able to join the session. If the session is in-progress, then users that wish to join the poll session will require approval from the host. Poll sessions that are open do not require approval to join. All status's require the appropriate room code.
- **results: SessionResults:** data involving the results of the poll
    - **total_participants: number** is the # of users who took part of the poll session
    - **median: number** median score of the poll if applicable

    - **average: number** average score of the poll if applicable
    - **low: number** lowest score of the poll if applicable
    - **high: number** highest score of the poll if applicable
- **created_at: Timestamp** is the timestamp at which this poll session was created.

Waiting Users in Session (sessions:/id/waiting_users): Stores users waiting in the lobby of a running poll session prior to starting.

- **photo_url: string | null** is the URL stored in Firebase's Cloud Storage that holds the user's profile picture, should they have one.
- **display_name: string | null** is the display name the user has chosen.

Session Participants (sessions/:id/users): Stores active participants within a running poll session.

- **photo_url: string | null** is the URL stored in Firebase's Cloud Storage that holds the user's profile picture, should they have one.
- **display_name | null** is the display name the user has chosen upon entering the poll

Session Questions (sessions/:id/questions): Stores the questions to be asked in a given live poll session. The semantics for these fields are identical to the **Questions Collection** (polls/:id/questions).

- **prompt_type: PromptType**
- **prompt: string**
- **prompt_img: string**
- **options: string[]**
- **points: number**

- **async: boolean | null**
- **anonymous: boolean**
- **time: number**

Session Question Responses (sessions/:id/questions/responses): Stores the responses of all participants within a live poll session.

- **user: DocumentReference<User>** reference to the user who made the response.
- **answer: string** the answer option the user chose.
- **correct: boolean** whether the user's chosen answer is correct
- **created_at: Timestamp** the time at which the user responded to the question.

Submissions Collection: Stores the submissions of all participants after a given live poll session.

- **session: DocumentReference<Session>** reference to the poll user participated in.
- **user: DocumentReference<User>** reference to the user who took the poll.
- **display_name: string** the display name the user used for the poll.
- **total_score: number** the score the user obtained after the poll.
- **submitted_at: Timestamp** the time the user completed the poll (if the poll was async, then the expected times would be similar across all users who participated in the poll).

6. Design Patterns

6.1 Model View Controller (MVC)

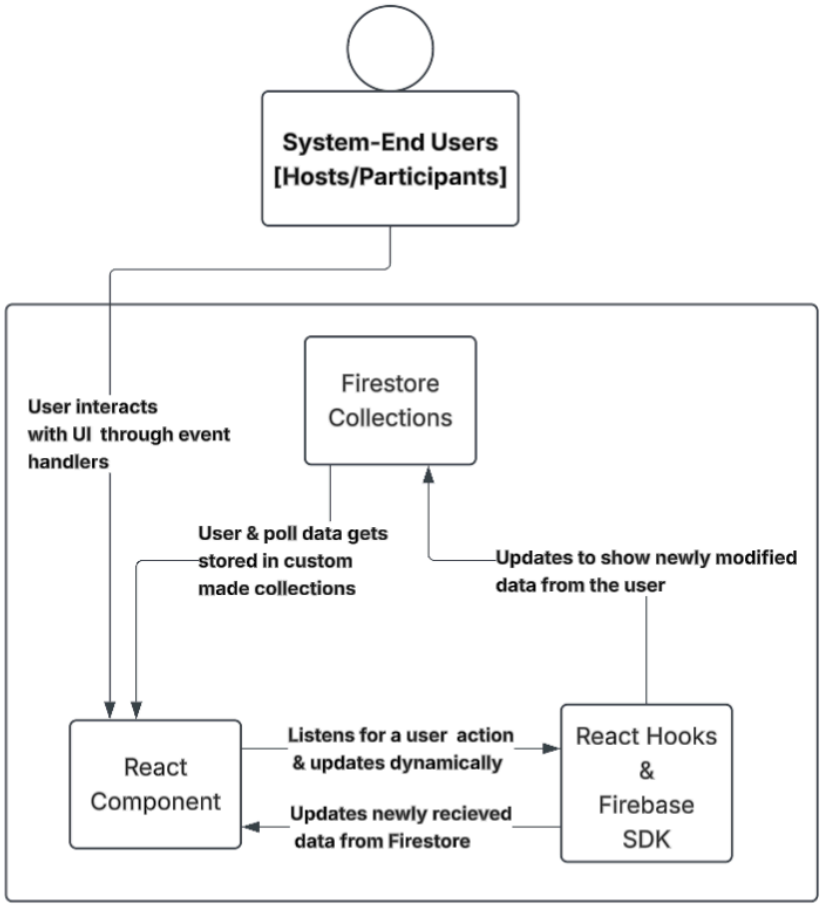

Figure 6.1

The system follows the Model-View-Controller (MVC) architectural pattern to ensure scalable, maintainable, and modular system. Decoupling the data (Model), presentation (View), and logic (Controller) allows the system for independent development, testing, and updates across different layers of the system without affecting the entire system; the Controller acts as a mediator, or proxy, to allow such changes to be recorded in other layers of the system.

- Model: Firestore Collections
    - Firestore can store all user and poll data in a document based structure. Collections represent different data entities, such as users, polls, and responses. In addition, firestore provides a real-time mechanism that allows for the update of data across all clients that are connected to the website.
- View: React UI components (Material-UI)
    - The view layer consists of React components using Material-UI . These components define the user interface and enable reusable maintainability. Reusable components such as buttons, input fields, and dropdowns are some examples of this. In addition, with dynamic rendering, components can update periodically based on the data received from the controller.
    - **Maybe provide an code example here (on click example, handler)**
- Controller: React Hooks, Event Handler, with Firebase SDK handlers.
    - React hooks act as the controller to manage state and logic by interacting with Firestore using the Firebase Software Development Kit (SDK). Such hooks give representation to data fetching, receiving polls and responses, real-time updating, using Firestore snapshot, and managing login/logout using Firebase Auth. In addition to this, hooks allow for state management in UI updating.

6.2 Observer Pattern

Real-time updates in poll sessions are handled using Firebase SDK's real-time listeners known as snapshots which are encapsulated in Firebase's react hooks. When it comes to implementation, the Firebase SDK implies a snapshot function to ensure changes in poll data is auto-pushed to all clients, which then ensures all data is synced across all users on the subscribed poll data. This implementation goes as follows:

- A user creates a poll and such data gets stored in Firestore.
- The Firebase SDK triggers a snapshot listener, and notifies all connected clients participating in the poll.l
- React hooks then communicate with the Firestore database to update the UI dynamically.

6.3 Factory Pattern

The Factory pattern is used for dynamically generating poll components, using functional React components, based on question types. Benefits in using such a pattern allows for code reusability and maintainability, where new question types can be added without a modification to pre-existing components and eliminates an attempt on having to use redundant code when referencing similar logic.

7. Security Considerations

In order to preserve the integrity of data affiliated with authenticated and guest users, PulseCheck implements and enforces custom Firestore Security Rules for both roles. Additionally, the system's security rules create a distinction between host and participant in addition to their authentication status. These user subcategories limit or allow access to specific read and write tasks, depending on their status. Access is distributed as follows:

- Access to authenticated user dashboard
    - Access to the authenticated user dashboard is only granted to users once request.auth.uid is assigned a non-null value, which tells the system that the user is authenticated.
    - Once access is granted, the user will be able to read and write on their specific dashboard. Possible actions include:

        - Draft, publish and update polls hosted by themselves
        - View their own results for past quizzes
        - View results of participants that have taken quizzes published by the user (unless set to anonymous)
        - Participate in published polls
- Guest User Acess
    - If a user chooses not to join through a PulseCheck or Google account, their request.auth.uid variable is assigned a null value, indicating to the system that they are guest users. This status limits their access to the following actions:
        - participate in published polls
        - View results right after quiz termination
- Poll Read
    - If the user's ID matches the ID of the writer of the given poll, they will be able to view the poll at both draft (in progress) and published states. This user is the host.
    - If a user's id does not match the id of the writer of the poll, they will only be able to read the poll until the resource.data.state has been set to "published" by the poll writer. This user is either an authenticated or guest participant.
    - To prevent unwanted participants from joining a poll, randomly generated room codes are used to join each respective poll session.
    - Once a poll is completed, participant users can view their respective scores for the poll, but cannot view another participant's responses, regardless of their authentication status.

    - If the poll does not have an “anonymous” setting, the host user is able to view individual participant performance at all times after poll termination, as well as real time responses throughout the duration of the session.

- Poll Write
    - Once a user initiates a poll, they are set as the admin of the poll. Meaning they can update the resource.data.state of the poll from “in progress” to “published”, and back to “in progress” if needed. This user is an authenticated host.
    - Users are only able to write questions in polls they are the administrator of. Meaning that participants are denied access to edit poll questions written by a distinct host user.
    - In sessions, participant users are able to document their own responses to the questions, but cannot interfere with another user’s responses. Host users are also unable to write responses for another participant user.
    - Once a poll session is initiated and question traversal begins, users are not able to go back and change their responses.

- Anonymous Poll Participant View
    - For polls with an “Anonymous” setting, only participant users will be able to view their own scores with answers. This can be done with the result display right after poll termination for all participants, and on their respective dashboard if the user is authenticated. Host users will only be able to view general performance, but won’t be able to read specific participant responses.

8. Deployment and CI/CD

Github Actions is a continuous integration and delivery tool allowing for automatic building, and deployment of code within the github repository (refer to Figure G.R) when set workflow conditions are met, such as commits or pull requests. Github Actions automatically builds the system whenever pull requests are committed on the main branch, allowing for easier testing as well as automatic deployment using Firebase's hosting service, making it so the deployed site does not need to be updated every single time there is a change in the code.

Workflows are automated processes that can be defined within the repository letting tasks execute automatically when appropriate conditions are met, such as the commit and pull request examples mentioned earlier. The workflow procedure is written using YAML Scripts, which are configuration files that let you execute command lines, and control when those command lines are triggered. This allows for full control regarding when and how Github Actions are used.

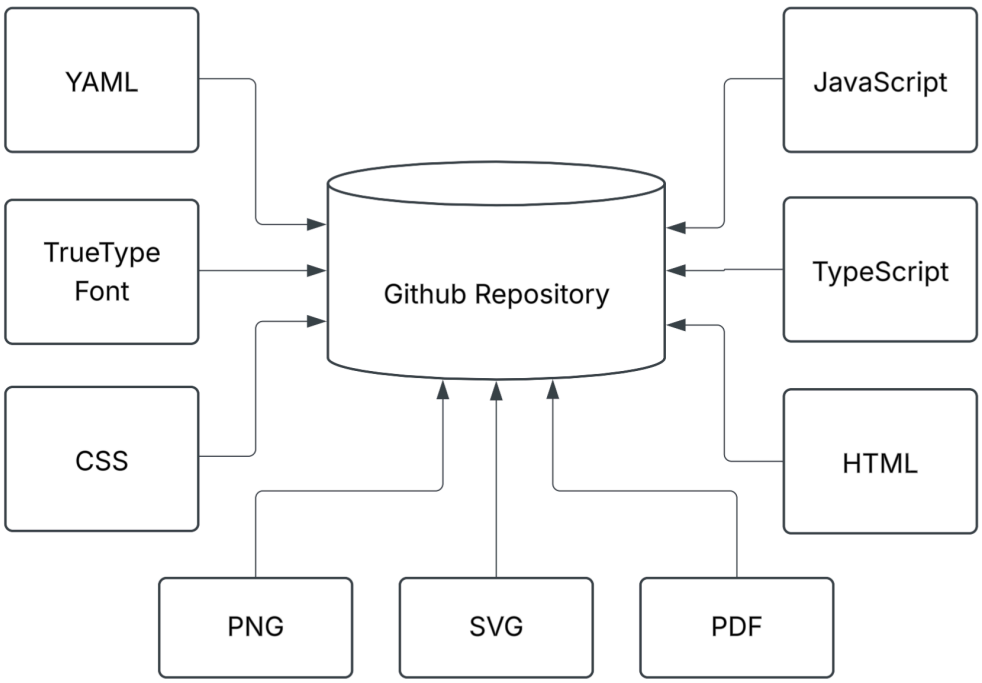

Figure G.R

In Figure G.R, the diagram represents the repository of the types of files the system will require in order to build the production build for deployment using Github Actions. This repository bucket is being stored on Github https://github.com/CSC190-289/PulseCheck/tree/main. Types of files include static content (e.g. PNG, SVG, PDF, and TrueType files). HTML for accessing the DOM (Document Object Model) to render the defined React components in the code base (src directory). CSS files for customized styling in addition to styling offered by Material-UI. JavaScript files for configuration of the development environment (e.g. Prettier, ESLint, Vite packages). TypeScript files for defining React components to build the system. And YAML files to define the scripts Github Actions executes to deploy the system using Firebase Hosting.