**🕌 Quran Application: System Design Document**

This document provides a comprehensive overview of the Quran application, detailing its architectural components, data structures, and the responsibilities of each module.

**1. Overall Application Architecture (HLDL)**

The Quran application follows a **Modular JavaScript** architecture, integrating with **Firebase** for backend services (Authentication and Firestore Database). The user interface is built with standard HTML and styled using Tailwind CSS.

* **Frontend (Browser-based)**:
  + **quran.html**: The main user interface, defining the layout, navigation tabs, display areas, and controls.
  + **JavaScript Modules (.js files)**: These are the brain of the application, handling all dynamic behavior, API calls, state management, and interaction with Firebase. They are organized into distinct files for better maintainability.
  + **Tailwind CSS**: Provides a utility-first CSS framework for responsive and modern styling.
* **Backend (Firebase)**:
  + **Firebase Authentication**: Manages user sessions, primarily using anonymous authentication for simplicity.
  + **Firestore Database**: A NoSQL cloud database used to persist user-specific data such as reading progress, listening history, and rewards.
  + **Firebase Hosting**: (Implied) Serves the static HTML, CSS, and JavaScript files.

**2. HTML Structure (quran.html)**

The quran.html file defines the visual layout and provides the DOM elements that the JavaScript interacts with. It's structured into logical sections:

* **Header**: Contains the application title and user authentication status (#auth-section, #quran-user-name, #quran-logout-btn, #quran-login-btn).
* **User Display**: A simple area to show the user's status (#user-display-status).
* **User Dashboard (#user-dashboard)**:
  + Displays various progress metrics:
    - **Ayahs Read**: Today, This Week, This Month, Overall (#total-read-today, #total-read-week, #total-read-month, #total-read-overall).
    - **Ayahs Heard**: Today, This Week, This Month, Overall (#total-listen-today, #total-listen-week, #total-listen-month, #total-listen-overall).
    - **Rewards**: Earned, Redeemed, Available (#rewards-accumulated, #rewards-redeemed, #rewards-available).
* **Controls Container (.controls-container)**:
  + **Tab Buttons**: For switching between "Listen" and "Read" modes (#tab-listen, #tab-read).
  + **Listen Tab Content (#listen-tab-content)**:
    - Dropdowns for Surah, Reciter, Speed, and Script (#surah-select, #reciter-select, #speed-select, #script-select).
    - Audio playback controls (Play/Pause, Stop) (#play-pause-btn, #stop-btn).
    - An HTML <audio> element (#quran-audio) for playback.
    - Display area for Quran text in Listen mode (#quran-display).
  + **Read Tab Content (#read-tab-content)**:
    - Dropdowns for Surah, Ayah, and Script (#surah-select-read, #ayah-select, #script-select-read).
    - Navigation buttons (Previous Ayah, Start/Done Reading, Next Ayah) (#prev-ayah-btn, #start-done-reading-btn, #next-ayah-btn).
    - Display area for Quran text in Read mode (#quran-display-read).
* **Notifications**: Areas for loading spinner, error messages, and confirmation messages (#loading-spinner, #error-message, #save-confirmation-message, #reading-summary).
* **Script Imports**: References to all JavaScript modules at the end of the <body> for efficient loading.

**3. JavaScript Module Structure and Responsibilities (LLDL)**

The application's logic is cleanly separated into several JavaScript modules, promoting maintainability and reusability.

**a. firebase-config.js**

* **Purpose**: Initializes the Firebase application and exports the necessary Firebase service instances (Auth and Firestore). This keeps your Firebase configuration centralized and separate from the application logic.
* **Key Exports**: auth (Firebase Authentication instance), db (Firestore Database instance).

**b. quran-api.js**

* **Purpose**: Handles all external API calls to fetch Quran data (Surah lists, reciters, scripts, Ayah text, and audio URLs). It acts as the data layer for the application.
* **Key Exports**:
  + fetchSurahs(): Retrieves a list of all Surahs.
  + fetchReciters(): Fetches available Quran reciters.
  + fetchScripts(): Fetches available translation/transliteration scripts.
  + fetchSurahData(surahNumber, scriptIdentifier): Gets Arabic text and translation for a specific Surah and script.
  + fetchSurahAudio(surahNumber, reciterIdentifier): Gets audio URLs for each Ayah of a specific Surah by a given reciter.

**c. quran-auth-display.js**

* **Purpose**: Manages the display of user authentication status in the header, independent of the main Quran app logic. It listens to Firebase Auth state changes and updates the UI (e.g., showing "Guest" or email, and toggling login/logout buttons).

**d. quran-app.js (Main Application Orchestrator)**

* **Purpose**: This is the central hub that glues all other modules together. It manages the overall application state, initializes Firebase, handles user authentication callbacks, orchestrates data loading, and sets up all event listeners for UI interactions.
* **Key Responsibilities**:
  + **Global State**: Manages userId, isReadingActive, isPlaying, currentTab, surahs, reciters, scripts, currentSurahData, currentSurahAudioData, listeningAyahIndex, currentAyahIndex, currentPlaybackSpeed.
  + **Initialization**: initializeAppAndListeners() runs on window.onload, signs in users (anonymously if not already signed in), ensures user Firestore documents exist, fetches initial API data, and loads user progress.
  + **UI Utilities**: Provides helper functions like showLoading(), hideLoading(), showError(), clearError(), showConfirmation(), setDisplayUsername(), and populateSelect().
  + **Dashboard Display (displayUserDashboard)**: Fetches data from dailyReadLogs, dailyListenLogs, and rewardsSummary to populate the user dashboard.
  + **Tab Switching (switchTab)**: Manages the visibility of "Listen" and "Read" tab content and loads initial data for the active tab.
  + **Event Handling**: Attaches event listeners to all major UI elements (buttons, selects, audio player) and delegates specific logic to quran-read.js and quran-listen.js.

**e. quran-read.js**

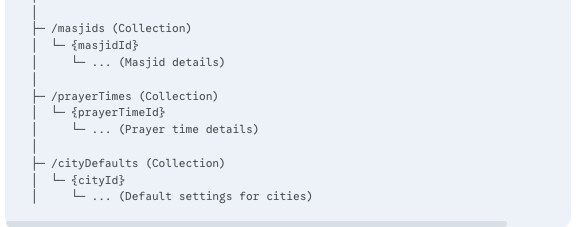
* **Purpose**: Contains the core logic and UI manipulations specific to the "Read" tab.
* **Key Exports**:
  + displaySingleAyah(index, surahData, displayEl, ayahSelectEl): Renders a single Ayah with its translation.
  + populateAyahDropdown(surahData, ayahSelectEl, currentAyahIndex): Fills the Ayah dropdown with numbers for the current Surah.
  + updateReadTabUIState(btnEl, isReadingActive): Updates the text and style of the "Start Reading" / "Done Reading" button.
  + updateDailyReadCount(userId, db, displayDashboardCb, errorCb): Increments the daily read count in Firestore and updates rewards.
  + loadReadingProgress(userId, db, surahSelectEl, confirmCb, errorCb): Fetches the user's last read position from Firestore.
  + saveReadingProgress(userId, db, surahSelectEl, currentAyahIndex, confirmCb, errorCb): Saves the user's current reading position to Firestore.
  + navigatePreviousAyah(params): Logic for moving to the previous Ayah or Surah in read mode.
  + navigateNextAyah(params): Logic for moving to the next Ayah or Surah in read mode.
  + updateRewardsEarned(userId, db, amount, errorCb): Centralized function to increment earned rewards.

**f. quran-listen.js**

* **Purpose**: Contains the core logic and UI manipulations specific to the "Listen" tab.
* **Key Exports**:
  + displayFullSurah(surahData, displayEl, audioEl, highlightCb): Renders the entire Surah for listening, setting up audio.
  + stopPlayback(audioEl, playPauseBtn, displayEl, listeningAyahIndex): Pauses audio, resets playback, and updates UI.
  + startRecitation(audioData, index, audioEl, playPauseBtn, highlightCb, errorCb, stopCb, speed, displayEl): Starts playing a specific Ayah's audio.
  + playNextAyah(index, audioData, audioEl, updateListenCountCb, stopCb, confirmCb, highlightCb, firestoreParams): Handles automatic playback of the next Ayah and increments listen count.
  + highlightAyah(index, displayEl): Visually highlights the currently playing Ayah in the displayed text.
  + updateDailyListenCount(userId, db, displayDashboardCb, errorCb): Increments the daily listen count in Firestore and updates rewards.

**4. Firebase Firestore Database Structure**

The database structure is designed to be user-centric and to efficiently store progress and reward data. All user data is nested under a document identified by their userId within the users top-level collection.



**5. Firebase Security Rules**

Robust security rules are critical to ensure that users can only access and modify their own data. Here's the most up-to-date set of rules designed for this structure:

rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

function isAuthenticated() {

return request.auth != null;

}

function getExistingUserRole() {

// Ensures the user's document exists and has a 'role' field

return get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role;

}

function getUserMasjidId() {

// Ensures the user's document exists and has a 'masjidId' field

return get(/databases/$(database)/documents/users/$(request.auth.uid)).data.masjidId;

}

// Rules for the top-level 'users' collection and its subcollections

match /users/{userId} {

// Allow read, update, delete only if the user is authenticated and accessing their own document

allow read, update, delete: if isAuthenticated() && request.auth.uid == userId;

// Allows new user document creation on first sign-in (e.g., anonymous auth)

allow create: if isAuthenticated();

// NEW: Rules for the 'lastAyahRead' subcollection

match /lastAyahRead/{docId} { // docId will be 'summary'

allow read, write: if isAuthenticated() && request.auth.uid == userId;

}

// NEW: Rules for 'dailyReadLogs' and 'dailyListenLogs' subcollections

match /dailyReadLogs/{docId} { // docId will be the date, e.g., '2025-08-11'

allow read, write: if isAuthenticated() && request.auth.uid == userId;

}

match /dailyListenLogs/{docId} { // docId will be the date, e.g., '2025-08-11'

allow read, write: if isAuthenticated() && request.auth.uid == userId;

}

// NEW: Rules for 'rewardsSummary' subcollection

match /rewardsSummary/{docId} { // docId will be 'summary'

allow read, write: if isAuthenticated() && request.auth.uid == userId;

}

// Existing 'rewards' rule (if still needed for individual transactions, otherwise remove)

match /rewards/{rewardId} {

allow read, write: if isAuthenticated() && request.auth.uid == userId;

}

}

// Rules for other collections (unchanged)

match /masjids/{masjidId} {

allow read: if true;

allow write: if isAuthenticated() && getExistingUserRole() == 'superAdmin';

allow update: if isAuthenticated() && getExistingUserRole() == 'masjidAdmin' && getUserMasjidId() == masjidId;

}

match /prayerTimes/{prayerTimeId} {

allow read: if true;

allow write: if isAuthenticated() && getExistingUserRole() == 'superAdmin';

allow update: if isAuthenticated() && getExistingUserRole() == 'masjidAdmin' && getUserMasjidId() == prayerTimeId;

}

match /cityDefaults/{cityId} {

allow read: if true;

allow write: if isAuthenticated() && getExistingUserRole() == 'superAdmin';

}

}

}

**6. Data Flow and Logic (LLDL Examples)**

**a. Initial App Load & User Authentication**

1. quran.html loads, including all .js modules.
2. window.onload triggers initializeAppAndListeners in quran-app.js.
3. onAuthStateChanged (from firebase-auth.js) detects the user's authentication state.
4. If no user is logged in, signInAnonymously attempts to create an anonymous user.
5. quran-app.js ensures a root user document exists in users/{userId}.
6. quran-app.js then calls fetchSurahs(), fetchReciters(), fetchScripts() from quran-api.js concurrently.
7. Once core data is loaded, displayUserDashboard(userId, db) is called from quran-app.js.
   * It queries users/{userId}/dailyReadLogs and users/{userId}/dailyListenLogs to sum up counts for today, week, month, and overall.
   * It also fetches the users/{userId}/rewardsSummary/summary document to display earned, redeemed, and available rewards.
8. loadReadingProgress(userId, db, ...) from quran-read.js retrieves the last read Ayah from users/{userId}/lastAyahRead/summary.
9. Finally, switchTab(currentTab) initializes the default "Listen" or "Read" tab content.

**b. Incrementing Read Ayah Count**

1. User navigates to the next Ayah in the "Read" tab via nextAyahBtn (handled by quran-app.js).
2. quran-app.js calls navigateNextAyah() in quran-read.js.
3. Inside navigateNextAyah(), updateDailyReadCount(userId, db, ...) is called.
4. updateDailyReadCount():
   * Constructs the path users/{userId}/dailyReadLogs/{YYYY-MM-DD}.
   * Fetches the current day's document. If it exists, increments the count field. If not, creates it with count: 1.
   * Calls updateRewardsEarned(userId, db, 1, ...) (also in quran-read.js) to add 1 to the earned rewards.
   * Triggers displayUserDashboard() again to refresh the dashboard display.

**c. Incrementing Listen Ayah Count**

1. Audio playback ends for an Ayah (handled by quran-app.js's quranAudio.addEventListener('ended', ...)).
2. quran-app.js calls playNextAyah() in quran-listen.js.
3. Inside playNextAyah(), after starting the next Ayah's audio, updateDailyListenCount(userId, db, ...) is called.
4. updateDailyListenCount():
   * Constructs the path users/{userId}/dailyListenLogs/{YYYY-MM-DD}.
   * Fetches the current day's document. If it exists, increments the count field. If not, creates it with count: 1.
   * Calls updateRewardsEarned(userId, db, 1, ...) (in quran-read.js) to add 1 to the earned rewards.
   * Triggers displayUserDashboard() again to refresh the dashboard display.