KOLEJ PROFESIONAL MARA BERANANG
KOLEJ PROFESIONAL MARA INDERA MAHKOTA

DIPLOMA IN COMPUTER SCIENCE

COURSE NAME	:	**OBJECT ORIENTED PROGRAMMING**
COURSE CODE	:	**CSC2923**
ACADEMIC SESSION	:	**SESSION 2 2025/2026**
TYPE OF ASSESSMENT	:	**FINAL ASSIGNMENT**
DURATION	:	**15/10/2025 - 5/11/2025**

CLO 3: Employ third party data in object oriented application development using graphical user interface (GUI) application framework

INSTRUCTION TO CANDIDATES:

1. Late submissions after the given due date will not be accepted.
2. Report should be written using:
   Font type: Arial
   Size: 12 pts
   Line Spacing: 1.5
3. Coding format:
   Font type: Consolas
   Size: 10 pts
   Line Spacing: Single

Personal Details	
Name	MUHAMMAD AIRIEL IKMAL BIN AZHAR
I/D Number	ICS24-02-040
Class	DCS 4C
Lecturer	MADAM ZULIARTY

Section / Question No.	**Marks**
1	
2	
3	
4	
5	
6	
Total	**/ 50**

I hereby declare that no form of plagiarism will be tolerated in this assessment. Failure to comply will result in a failing grade for the assessment.

Signature: _____ Date: 10/11/2025

1.0 Introduction and Overview

The Quran Journal is a comprehensive desktop application developed using Electron.js that serves as a digital companion for Muslims seeking to enhance their Quranic journey. This application addresses the growing need for organized spiritual tracking in the digital age by providing a structured platform where users can systematically engage with the Quran, record their reflections, and monitor their reading progress over time.

The application integrates with the Al-Quran Cloud API to fetch real-time data about all 114 surahs, ensuring users have access to accurate information including surah names in Arabic and English, verse counts, revelation types, and translations. This reliable data foundation allows users to focus entirely on their spiritual growth.

CRUD Implementation:

- Create: Users can add new journal entries by selecting surahs, tracking progress, and writing reflections
- Read: The application displays organized lists of surahs and journal entries with search functionality
- Update: Existing journal entries can be modified, including progress updates and reflection edits
- Delete: Users can remove journal entries with confirmation dialogs to prevent accidental deletion

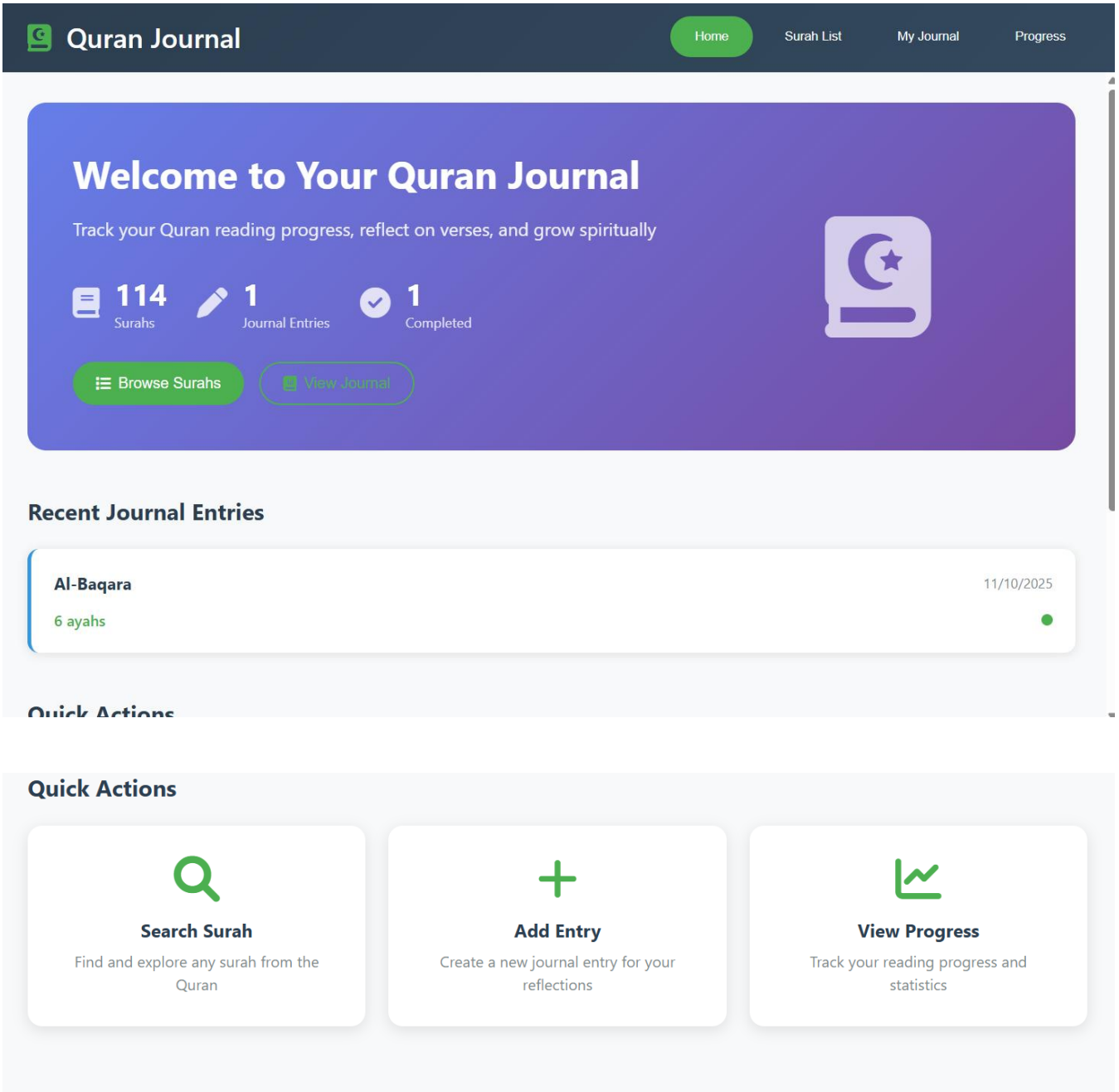
This comprehensive data management system transforms the application from a simple tracker into a personalized digital journal that grows with the user's spiritual journey.

2.0 Features and Functionalities

Homepage:

The homepage serves as the central dashboard, providing users with immediate insights into their Quranic journey through visually appealing statistics and quick access to key features:

- Welcome section with application overview
- Quick statistics display (total surahs, journal entries, completed surahs)
- Recent journal entries preview
- Quick action cards for easy navigation to main features

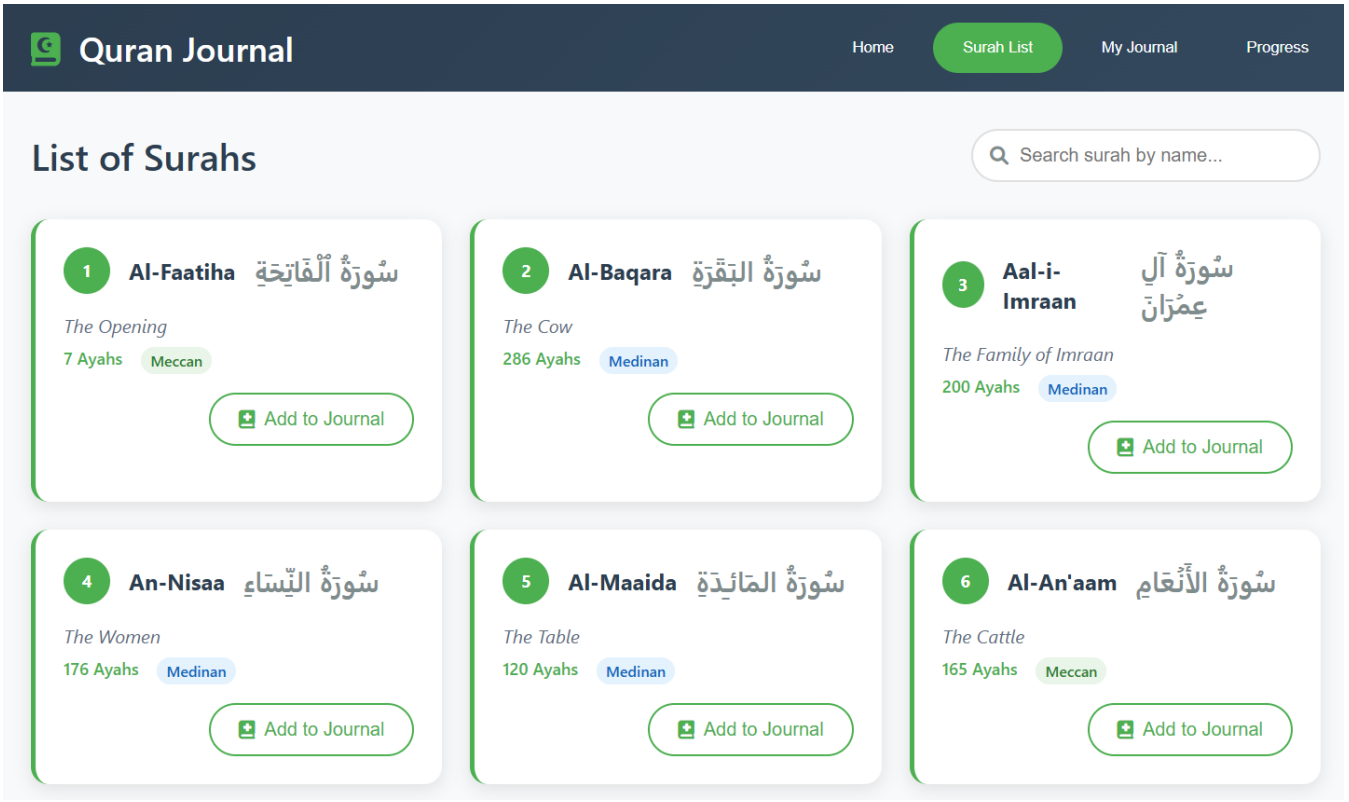


Surah List Page

This page displays the complete collection of Quranic surahs in an organized grid layout, featuring robust search functionality and detailed surah cards that enable users to quickly find and select surahs for journaling.Display of all 114 surahs in organized grid layout

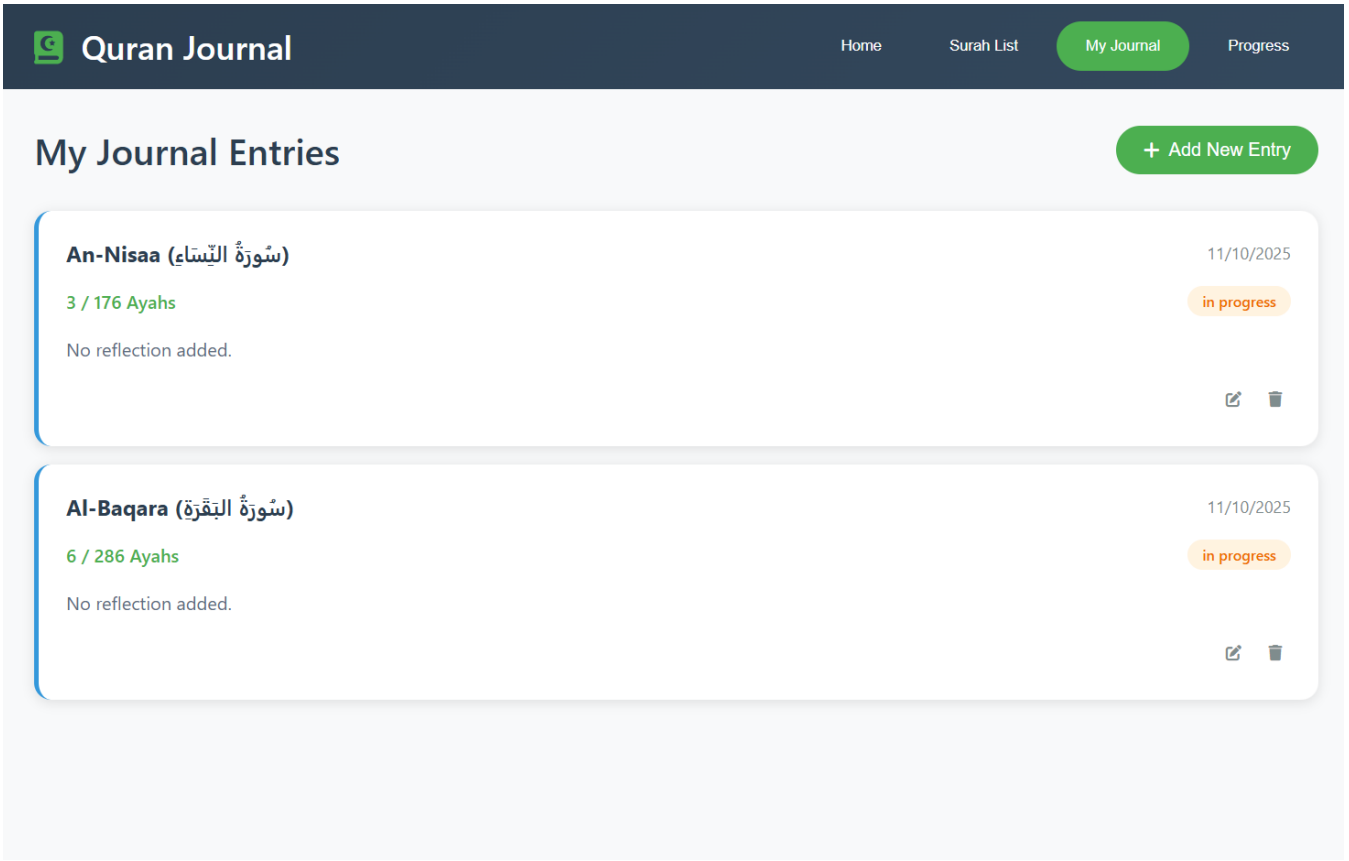
Its contain:

- Real-time search by surah names (Arabic/English)
- Detailed surah cards showing number, names, ayah count, and revelation type
- Direct "Add to Journal" buttons for quick entry creation



My Journal Page
The journal management interface provides a comprehensive view of all user entries, organized chronologically with clear progress indicators and easy access to editing tools for maintaining reflective records and contains:

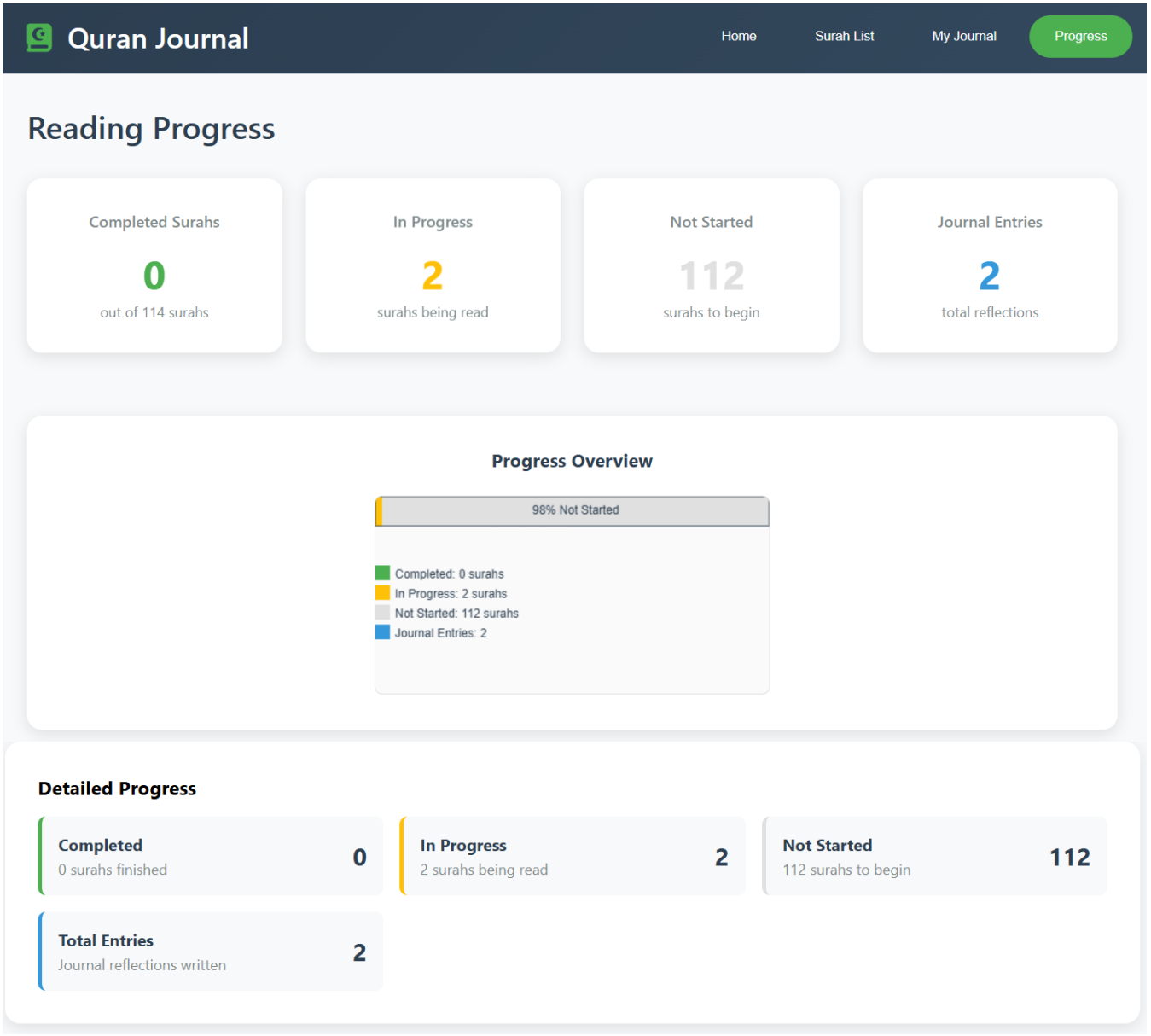
- Chronological display of all journal entries
- Add new entries via modal form
- View entry details including surah info, progress, and reflections
- Edit and delete functionality for each entry
- Empty state guidance for new users



Progress Tracking Page
This analytical dashboard transforms user data into visual insights through statistical cards and progress charts, helping users understand their reading patterns and set meaningful goals for their Quranic study.Statistical cards showing completion metrics

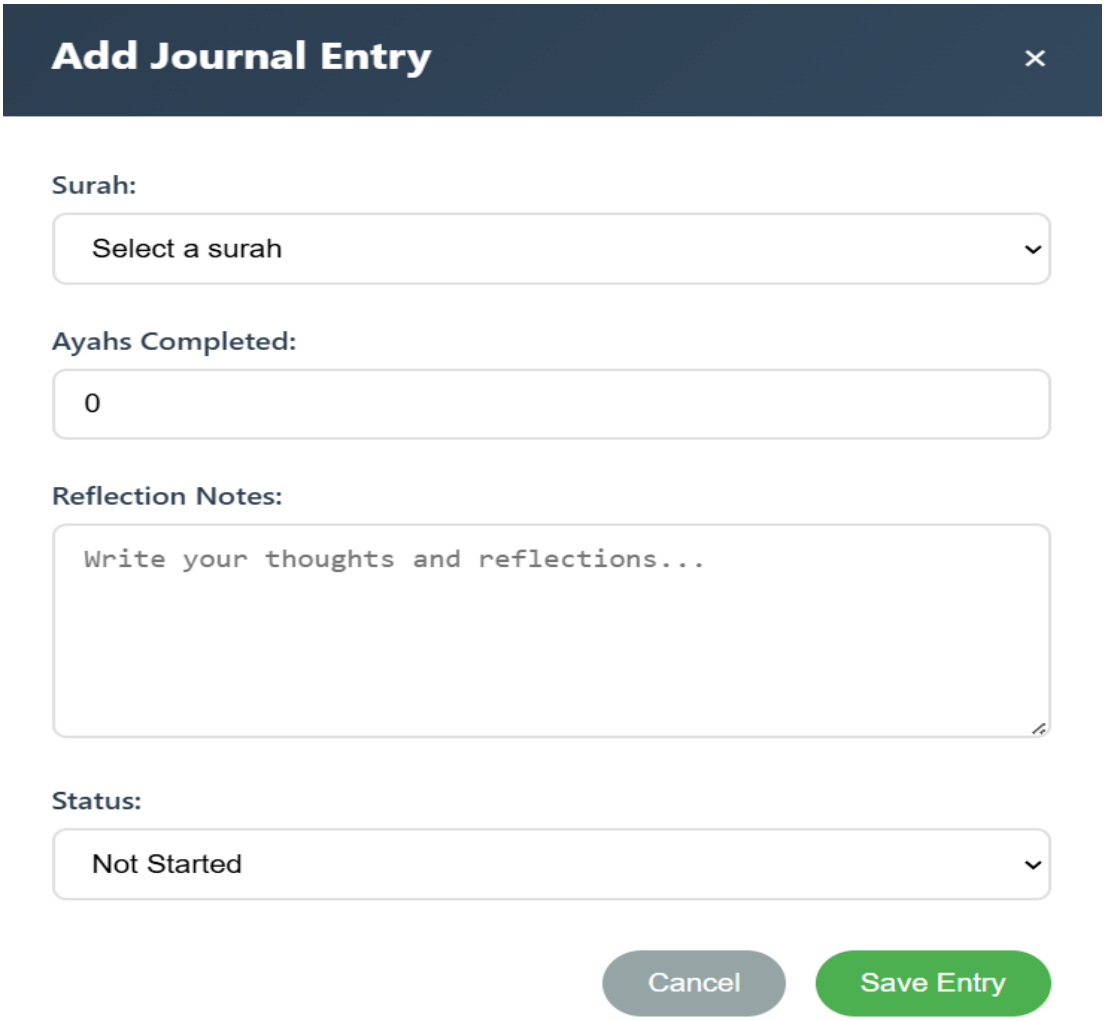
Its also contain:

- Color-coded progress bar chart
- Detailed breakdown of reading status
- Total journal entries count
- Visual representation of Quran reading journey



Journal Entry Modal
The modal interface provides a focused environment for creating and editing journal entries, with intelligent form fields that adapt based on surah selection and input validation to ensure data accuracy and contains:

- Surah selection dropdown with all 114 options
- Ayahs completed input with automatic validation
- Reflection notes textarea for detailed thoughts
- Status selection with auto-update based on progress
- Form validation and user feedback



3.0 Code Highlights

1.API Integration

```
class QuranAPI {
    constructor() {
        this.baseURL = 'http://api.alquran.cloud/v1';
    }

    async fetchSurahs() {
        try {
            const response = await fetch(`${this.baseURL}/surah`);
            const data = await response.json();

            if (data.code === 200 && data.data) {
                return data.data.map(surah => ({
                    number: surah.number,
                    name: surah.name,
                    englishName: surah.englishName,
                    englishNameTranslation: surah.englishNameTranslation,
                    numberOfAyahs: surah.numberOfAyahs,
                    revelationType: surah.revelationType
                }));
            }
            throw new Error('Failed to fetch surahs');
        } catch (error) {
            console.error('API Error:', error);
            // Return sample data if API fails
            return this.getSampleSurahs();
        }
    }
```

Explanation:

The API integration class serves as the bridge between the application and the external Quran data source. It implements a robust error handling system that ensures the application remains functional even when network connectivity is compromised. The class makes asynchronous requests to the Al-Quran Cloud API and processes the response data into a standardized format that the application can consistently work with. When API calls fail, it gracefully falls back to locally stored sample data, maintaining application functionality while providing appropriate error feedback to users.

2.CRUD Operations

Create Operation

```
async createEntry(entryData) {
    console.log('Creating new entry:', entryData);

    const newEntry = {
        id: Date.now().toString(),
        ...entryData,
        createdAt: new Date().toISOString(),
        updatedAt: new Date().toISOString()
    };

    this.entries.push(newEntry);
    console.log('Entries after push:', this.entries);

    const success = await this.saveData();

    if (success) {
        console.log('Entry created successfully');
        return { success: true, entry: newEntry };
    }
    console.error('Failed to save entry');
    return { success: false, error: 'Failed to save entry' };
}
```

The create operation is in charge of adding new entries to the Journal. When a user fills out the journal form and submits it, this operation generates a unique id from the current timestamp and binds it with user input. The operation also adds records for a creation and update timestamp to track when the entry was created and when it was last updated. The operation will add the newly created entry to the existing entries array and persist the data to storage. Feedback confirmation of submission is provided to the user when an entry is successfully created and describes a failure when there are error

Read Operation

```
// READ
getEntries() {
    return this.entries.sort((a, b) => new Date(b.updatedAt) - new Date(a.updatedAt));
}

getEntryById(id) {
    return this.entries.find(entry => entry.id === id);
}

getEntriesBySurah(surahNumber) {
    return this.entries.filter(entry => entry.surahNumber === surahNumber);
}
```

The read functionality is responsible for retrieving and displaying journal entries from storage. The stored journal entries are retrieved and sorted in descending order with respect to the last time they were modified (updated). So readers will always see their most recent updates first. The read function provides filtered views when specific subsets of the data are in demand, such as when searching for journal entries from a specific surah. The read loads in place error handling so that the application remains stable if journal entries cannot be loaded from storage.

Update Operation

```
// UPDATE
async updateEntry(id, updatedData) {
    console.log('Updating entry:', id, updatedData);

    const index = this.entries.findIndex(entry => entry.id === id);
    if (index !== -1) {
        this.entries[index] = {
            ...this.entries[index],
            ...updatedData,
            updatedAt: new Date().toISOString()
        };

        const success = await this.saveData();
        if (success) {
            console.log('Entry updated successfully');
            return { success: true, entry: this.entries[index] };
        }
        return { success: false, error: 'Failed to update entry' };
    }
    return { success: false, error: 'Entry not found' };
}
```

The update operation enables the user to modify the existing journal entries whenever their progress changes or they want to add new reflections. It locates the target entry by its unique identifier and replaces it with updated data, preserving the original creation timestamp. Automatically, the modification timestamp will be updated as a result of this operation, because changes have taken place. Therefore, it ensures the integrity of the data with respect to the creation and last modification time of the entries. This operation gives feedback to the user in terms of success or failure of the update attempt.

Delete Operation

```
// DELETE
async deleteEntry(id) {
    console.log('Deleting entry:', id);

    const index = this.entries.findIndex(entry => entry.id === id);
    if (index !== -1) {
        this.entries.splice(index, 1);
        const success = await this.saveData();

        if (success) {
            console.log('Entry deleted successfully');
            return { success: true };
        }
        return { success: false, error: 'Failed to delete entry' };
    }
    return { success: false, error: 'Entry not found' };
}
```

The delete action allows users to eliminate journal entries that they no longer wish to continue keeping. This action alerts users with a confirmation mechanism to help prevent users from accidentally deleting important data. The action finds the correct entry using its unique identifier and removes that journal entry from the entries array. After a successful delete, all subsequent data structures are updated, and the progress statistics for the user are also updated to maintain accurate data. The user is notified in the UI that the entry has been successfully removed. Any problems that arise during the action are clearly reported, so users can understand what went wrong.

3. Main Process (main.js)

```
// IPC handlers for file operations
ipcMain.handle('read-journal-data', async () => {
  try {
    const dataPath = path.join(__dirname, 'data', 'journal.json');
    if (fs.existsSync(dataPath)) {
      const data = fs.readFileSync(dataPath, 'utf8');
      return JSON.parse(data);
    }
    return { entries: [] };
  } catch (error) {
    console.error('Error reading journal data:', error);
    return { entries: [] };
  }
});
```

```
ipcMain.handle('save-journal-data', async (event, data) => {
  try {
    const dataPath = path.join(__dirname, 'data', 'journal.json');
    const dir = path.dirname(dataPath);
    if (!fs.existsSync(dir)) {
      fs.mkdirSync(dir, { recursive: true });
    }
    fs.writeFileSync(dataPath, JSON.stringify(data, null, 2));
    return { success: true };
  } catch (error) {
    console.error('Error saving journal data:', error);
    return { success: false, error: error.message };
  }
});
```

The Electron.js main process manages the application's lifecycle and system-level operations. IPC handlers facilitate safe data interchange between the main and the renderer processes while enforcing secure data exchange. Structure for file operations with robust error handling, directory, and file creation for durable data will ensure reliability. This architecture enforces safe security practices by isolating file system operations from the renderer, while providing an easy-to-use API for data management operations.

4. Progress Tracking Logic

```
getProgressStats() {
    const totalSurahs = 114;

    // Get all unique surahs that have entries
    const surahsWithEntries = new Set(this.entries.map(entry => entry.surahNumber));

    // For each surah with entries, determine its status based on the most recent entry
    const surahStatus = {};

    this.entries.forEach(entry => {
        const surahNumber = entry.surahNumber;
        // Only update if this entry is more recent or we don't have a status yet
        if (!surahStatus[surahNumber] ||
            new Date(entry.updatedAt) > new Date(surahStatus[surahNumber].updatedAt)) {
            surahStatus[surahNumber] = {
                status: entry.status,
                updatedAt: entry.updatedAt
            };
        }
    });
```

```
renderProgressChart(stats) {
    const canvas = document.getElementById('progress-chart');
    if (!canvas) {
        console.error('Progress chart canvas not found');
        return;
    }

    const ctx = canvas.getContext('2d');
    if (!ctx) {
        console.error('Could not get canvas context');
        return;
    }

    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    const total = stats.totalSurahs;
    const completedPercent = (stats.completed / total) * 100;
    const inProgressPercent = (stats.inProgress / total) * 100;
    const notStartedPercent = (stats.notStarted / total) * 100;

    console.log('Chart percentages:', { completedPercent, inProgressPercent, notStartedPercent });
```

The progress tracking feature utilizes an intelligent algorithm that evaluates (or runs an analysis on) journal entries to create useful statistics about the user's experience in reading the Quran. It analyzes all the entries to determine and track the latest state of each surah based on the most recently tracked entries. The algorithm takes into consideration the distinction of completed, in-progress, and not-started surahs, edge cases, and provides accurate calculations. This intelligent analysis offers the user useful insights into their reading behaviors and their pace to completion of the full Quran.

4.0 Project Folder Structure

```
quran-journal-app/
├── main.js                    # Electron main process - window management
├── preload.js                 # Security bridge between processes
├── package.json               # Dependencies and build scripts
├── src/
│   ├── index.html             # Main application interface
│   ├── styles/
│   │   └── main.css           # Complete application styling
│   └── js/
│       ├── api.js             # External API integration
│       ├── crud.js            # Data management operations
│       └── renderer.js        # User interface logic
├── data/
│   └── journal.json           # User data storage
└── assets/
    └── icons/                 # Application icons and images
```

5.0 Gifthub link

https://github.com/rielllll222/quran_journal.git

Assessment Rubric

ATTRIBUTES	CRITERIA	POOR (1 mark)	FAIR (2 marks)	GOOD (3 marks)	VERY GOOD (4 marks)	EXCELLENT (5 marks)	Weightage	Mark Obtained
Reproduce and Process Information	1. Develop a desktop application using Electron and integrate it with one external API	• Application does not run or is complete	• Application runs minimally but lacks structure or second screen	• Basic electron app with two screens but limited interaction	• Fully working app with two fully functional screens	• Well-structured and fully working app with more than two fully functional screens	1	
		• No API used or integration failed	• API used with limited or static display of data.	• API properly integrated with dynamic data shown.	• Excellent API use with dynamic display, error handling, and interaction.		1	
		• Screens are not related to the API or are repetitive.	• Screens are somewhat distinct but lack of clear purpose	• Two screens with clear but basic purposes.	• Screens have relevant and useful features tied to the API.	• All screens are thoughtfully designed, distinct, and add strong value to the user experience.	1	
	2. Implement full CRUD operations (Create, Read, Update, Delete) to allow users to manage relevant data in your app.	• Able to create only 2 of the CRUD processes. • No feedback for the CRUD process. • The design for data input is poor	• Able to create only 3 of the CRUD processes. • No feedback for the CRUD process. • The design for data input is good with some room for improvement	• Able to perform all the CRUD processes. • No feedback for the CRUD process. • Well-designed data input for CRUD process.	• Able to perform all the CRUD processes. • No feedback for the CRUD process. • Well-designed data input for CRUD process.	• Able to perform all the CRUD processes. • Appropriate feedback for the CRUD process. • Well-designed and user-friendly data input for CRUD process.	1	

	3. Design an attractive and interactive user interface using HTML and CSS.	• **Text -** All text used is too small to view or the font type is wrongly chosen. • **Graphics** - Graphics seem randomly chosen, and of are of low quality.	• **Text** – Some of the text used is too small to view or the font type is wrongly chosen. • **Graphics -** Graphics chosen are somewhat appropriate but are of low quality.	• **Text** - Most of the text used is of the right size and the font type is somewhat appropriate. • **Graphics** - Graphics chosen are related to the theme/ purpose of the application and are of excellent quality.	• **Text** - All text used is of the right size and the font type is appropriate • **Graphics** - Graphics chosen are related to the theme/purpos e of the application and are of excellent quality • Enhances reader's interest and understanding	• **Text** - All text used is of the right size, the font type is appropriate and able to describe the content well. • **Graphics** - Graphics chosen are related to the theme/purpose of the application, are of excellent quality,and thoughtfully cropped. • Enhances reader's interest and understanding	1	

Curate	4.GUI Elements: i. Apply essential GUI elements that assist users in using application.	• Not able to curate for required content. • **Layout** - The HTML elements in the application are cluttered looking or confusing. • **Navigation** Links do not take the reader to the sites/ pages described. User typically feels lost.	• Limited curation for required content. • **Layout** - The HTML elements in the application is messy, may appear busy or boring. • **Navigation** Links seem to be missing and don't allow the user to easily navigate.	• Satisfactory curation for required content. • **Layout** -The HTML elements are suitable. • **Navigation** Links allow the reader to move from page to page, but some links seem to be missing.	• Good curation for required content. • **Layout** - The HTML elements are suitable and usable. • **Navigation** Links are labelled and allow the user to easily move from page to page.	• Excellent curation for required content. • **Layout** - The HTML elements are well structured, attractive, and usable layout. • **Navigation** Links are clearly labelled, consistently placed, and allow the user to easily move from page to page.	1	
	ii. The application's 'look and feel' is attractive and informative.	• The application lacks visual coherence and polish; layout appears inconsistent or mismatched to the target audience. • Color choice and combinations are unsuitable and distract usability. • Minimal evidence of personal design input; interface appears generic	• The application shows some attention to visual design principles but still feels incomplete or unrefined • Colors and layout partially align with the application's concept but lack balance or harmony. • Limited customization is evident; design elements resemble pre-	• The application mostly follows good visual design principles (alignment, contrast, readability, consistency). • Color choices generally support the application's theme and target audience. • Some personalization or manual refinement is visible, though	• The application demonstrates good visual design principles and clear attention to layout and composition. • Color palette effectively supports the intended atmosphere and audience. • Design decisions show thoughtful customization and adaptation	• The application displays outstanding design coherence and originality, perfectly suited to the target audience. • Colors and layout are harmoniously integrated to enhance the concept and user experience. • Strong evidence of personal creativity and	1	

		or directly adapted from pre-made resources without meaningful customization	designed or auto-generated layouts with only minor adjustments	parts may still rely on pre-built patterns or styles	beyond ready-made styles or automated layouts	manual refinement throughout; interface is distinct and clearly hand-curated, not dependent on pre-built design structures or automated styling		
Convey	5. Produce a project report that includes following: i. Introduction and overview	• The description is unclear or missing. The API and CRUD components are not explained or are incorrect.	• The application's concept and purpose are mentioned but lack detail. API and CRUD explanations are minimal or partly incorrect.	• The concept is clear and relevant. API usage and CRUD operations are explained with reasonable accuracy and clarity.	• The application's concept and purpose are well-articulated. API usage is clearly and accurately explained, and CRUD implementation is outlined with meaningful and relevant examples.		1	
	ii. Features and functionalities	• The description of the functionalities and features is poor and unorganized. • Incomplete print screen of the application.	• The description of the functionalities and features is very brief. • Incomplete print screen of the application.	• The description of the functionalities and features is complete. • Complete screenshot of the pages.	• The description of the functionalities and features is complete with extensive information. • Complete screenshot of the pages.	• The description of the functionalities and features is complete with extensive information. • Complete screenshot of the pages and the labelling of the features in the application.	1	

	iii. Code highlights	• Code snippets are mostly missing or irrelevant. API and CRUD are not shown or are incorrect. No or unclear explanations.	• Code shows minimal attempt to include API or CRUD logic. Explanations are limited or unclear. Formatting may be inconsistent.	• Code includes basic API and CRUD examples with some explanation. Some clarity and relevance but lacking depth.	• Code clearly demonstrates API integration and CRUD logic. Well-explained and formatted, with only minor gaps.	• Code is highly relevant, clearly presented, and thoroughly explained. API and CRUD logic are well-implemented and easy to follow.	1	
	6. Upload your project files to GitHUb and provide the repository URL in the report.	• Does not submit complete electron files in GitHub	• Completely submit all the electron files in GitHub and include the link in the report.				1	
Total Marks Earned								**/50**
Total Percentage (40%)								**/40%**