Ramadhan Companion App

Technical Report

Web Services Course

Presented By:

Balkis Djebbi
Senior IT/Fin Student

Academic Year: 2024/2025

University: Tunis Business School

Abstract

The Ramadhan Companion is a web-based application designed to assist Muslims during the holy month of Ramadhan by providing dynamic prayer times, fasting schedules, and Quranic reading plans. The application integrates astronomical calculations for accurate prayer times, Hijri-Gregorian date conversions, and location-based customization. Built using NestJS, MongoDB, and Docker, the system ensures scalability, real-time accuracy, and portability. This report details the architecture, functionalities, logic flow, and deployment process of the system, along with security considerations and API interactions.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The **Ramadhan Companion App** is an *intelligent web-based platform* designed to assist Muslims in *tracking prayer times, managing Quranic recitations, and following Ramadan fasting schedules.* The application dynamically computes *daily prayer schedules* and *updates users in real-time* based on their **location and time zone**.

Key functionalities include:

- A **Ramadhan calendar** with daily prayer times for the entire month.
- **Today's prayer times box** that highlights the **ongoing prayer** and shows both **Gregorian and Hijri dates**.
- A **real-time countdown timer** for the **next event (Imsak or Iftar)**.
- A **Quranic reading plan** to help users complete the Quran in 30 days.
- A **user profile management system** for updating location and password settings.

1.2 Motivation

During the month of **Ramadhan**, Muslims require accurate *prayer schedules, fasting times, and Quranic tracking tools.* Many existing apps rely on *static data*, which may not reflect the user's *exact location or updated Hijri calendar dates.* The **Ramadhan Companion App** aims to address these issues by:

- **Providing a real-time, location-aware Islamic timekeeping system**.
- **Using astronomical calculations** instead of static time schedules.
- **Automatically adjusting for time zones** and **Hijri date variations**.

Chapter 2

System Architecture

2.1 Backend Architecture

The back-end is built using *NestJS*, a TypeScript framework structured into modular services:

- **AuthService** – Manages user authentication and profiles.
- **QuranService** – Fetches Quranic data and tracks the progress of the recitation.
- **PrayersService** – Computes daily prayer times based on astronomy.
- **AstronomyService** – Calculates solar angles for prayer times.
- **RamadanService** – Generates Ramadhan fasting schedules.
- **DateParserService** – Converts and formats Hijri-Gregorian dates.

2.2 Frontend Architecture

The frontend uses:

- *Handlebars (HBS)* – A templating engine to render pages dynamically.
- *Tailwind CSS* – A responsive CSS framework for styling.
- *JavaScript (ES6)* – For client-side logic and real-time updates.

2.3 UML Diagram

The following UML diagram illustrates the back-end service interactions:

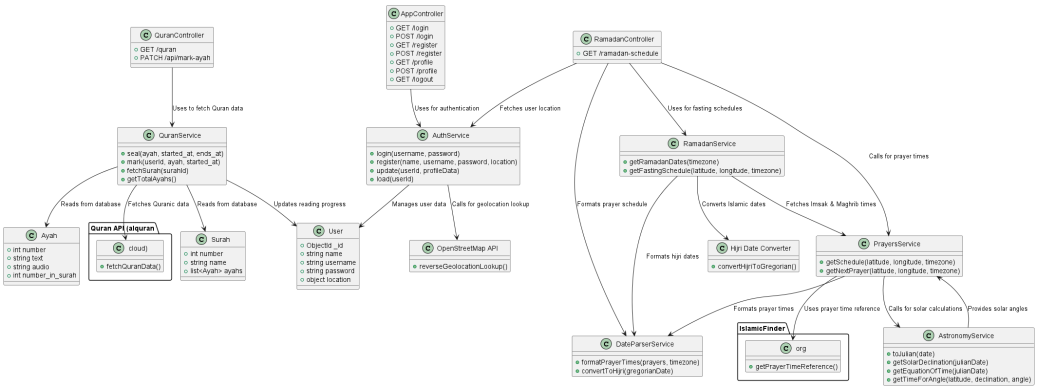

Figure 2.1: System Architecture UML Diagram

Chapter 3

Functionalities and Logic Flow

3.1 Ramadhan Schedule Page

- Displays a **calendar table** of prayer times for the full month.
- **Today's Prayer Box** highlights the **current prayer**.
- Shows **Imsak and Iftar times**.
- A **real-time countdown timer** dynamically tracks the time left for the next event.

3.2 Quran Page

- Displays a **daily Quranic reading plan**.
- Tracks the **last read Ayah**.
- Helps users complete the Quran within 30 days.

3.3 Profile Page

- Allows users to **update their password**.
- Provides an interactive map for setting a custom prayer location.

Chapter 4

API Endpoints and Logic

The Quran Web Services API is structured around three core controllers:

- **AppController** – Handles authentication and user profile management.
- **QuranController** – Manages Quran reading progress and retrieval.
- **RamadanController** – Computes Ramadan schedules and prayer times.

Each controller interacts with multiple services, which provide business logic, database access, and external API integrations. This section details how each endpoint processes requests, which services it depends on, and how data flows through the system.

4.1 Authentication API

Endpoints

- `/login` (GET, POST)
- `/register` (GET, POST)
- `/profile` (GET, POST)
- `/logout` (GET)

Logic Flow

The `AppController` manages authentication using the `AuthService`.

- The `/register` endpoint:

  - Accepts user data (`RegisterDto`).
  - Calls `AuthService.register()`, which:
    * Hashes the password using **bcrypt**.
    * Stores user credentials in **MongoDB**.
    * Uses the **OpenStreetMap API** to convert latitude/longitude into a city name.
  - Automatically logs in the user by calling `AuthService.login()` after successful registration.

- The `/login` endpoint:
  - Calls `AuthService.login()`, which:
    * Fetches user credentials from **MongoDB**.
    * Compares passwords using **bcrypt**.
    * Generates a JWT access token.
    * Sets the JWT token as a cookie for session management.

- The `/profile` endpoint:
  - Loads user details (name, location, preferences) from **MongoDB**.
  - Calls `AuthService.update()` when updating profile information.

- The `/logout` endpoint:
  - Clears authentication cookies and redirects the user to `/login`.

Key Variables and Services Used

- **AuthService**: Handles database queries, password hashing, and token management.
- **MongoDB (User Collection)**: Stores user credentials and location.
- **External API**: Uses **OpenStreetMap API** to resolve coordinates into a city name.

4.2 Quran API

Endpoints

- `/quran` (GET)
- `/api/mark-ayah` (PATCH)

Logic Flow

The `QuranController` interacts with `QuranService` to manage the progress of Quranic reading.

- The `/quran` endpoint:
  - Retrieves the user's last read Ayah (`AuthService.load()`).
  - Calls `QuranService.seal()`, which:
    - Computes a daily reading plan to complete the Quran in a given timeframe.
    - Uses `QuranService.total()` to determine the total number of Ayahs.
    - Fetches the next set of Ayahs for reading (`QuranService.today()`).
  - Returns a structured reading schedule.
- The `/api/mark-ayah` endpoint:
  - Retrieves the user's ID and reading progress (`AuthService.load()`).
  - Calls `QuranService.mark()`, which:
    - Updates the last read Ayah and start date in the **User Collection**.

Key Variables and Services Used

- **AuthService**: Loads user information and Quran reading progress.
- **QuranService**: Manages Quranic content and tracking.
- **MongoDB (Surah Collection)**: Stores Surah and Ayah data.

External API Used

- **Quran Data API**: `https://api.alquran.cloud/v1/quran/ar.alafasy` Used during system initialization to fetch Quranic text and audio. Stored locally in MongoDB to reduce API calls.

4.3 Prayer API

Endpoints

- Indirectly used in `/ramadan-schedule` (GET)

Logic Flow

The `PrayersService` computes daily prayer times dynamically instead of using a static prayer timetable.

- Converts the current date into Julian format (`AstronomyService.toJulian()`).
- Computes solar declination and the equation of time (`AstronomyService.getSolarDeclina`
- Uses trigonometric functions to determine:
  - **Fajr** (18° sun angle).
  - **Isha** (17° sun angle, based on **IslamicFinder.org**).
- Formats times based on the user's timezone (`DateParserService.getTime()`).

Validation

Prayer times were tested against multiple external APIs and confirmed to be accurate.

Key Variables and Services Used

- **AstronomyService**: Provides solar declination and time offsets.
- **DateParserService**: Formats computed prayer times.
- **MongoDB (User Collection)**: Stores user latitude, longitude, and timezone.

External API Used for Validation

- **IslamicFinder.org**: Reference for **Isha** and **Fajr** angles (17° and 18°).

4.4 Ramadan API

Endpoints

- `/ramadan-schedule` (GET)

Logic Flow

- Retrieves user location and timezone (`AuthService.load()`).
- Calls `RamadanService.dates()` to:
  - Convert Hijri to Gregorian (`hijri-converter` package).
  - Determine the start and end dates of Ramadan.

Key Variables and Services Used

- **AuthService**: Fetches user location and timezone.
- **PrayersService**: Provides daily prayer times for Ramadan.
- **RamadanService**: Computes Ramadan start/end dates.
- **DateParserService**: Formats prayer times.

External API Used

- **Hijri Date Converter**: `https://github.com/xsoh/hijri-converter`

Chapter 5

Deployment and Testing

5.1 Docker Deployment

- Containerized using Docker.
- Deployed using ‘docker-compose up –build‘.

5.2 API Testing

- Endpoints tested via Postman.
- The accuracy of prayer time was validated against external APIs.

Chapter 6

Conclusion

6.1 Summary of Findings

This report has presented a detailed analysis of the **Ramadhan Companion App**, covering its architecture, API endpoints, logic flow, deployment process, and security considerations. The system successfully integrates *prayer time calculations, Quranic reading tracking, and fasting schedules* through *modular services* and *external API integrations*. The application's scalability and accuracy were validated through *containerized deployment in Docker* and *API testing via Postman*.

6.2 Current Limitations

While the system provides a comprehensive solution for Ramadhan-related services, several *limitations* have been identified:

- **Limited Frontend Interactivity**: The current frontend relies on static templates and lacks advanced interactivity such as *real-time prayer countdowns*.
- **Timezone Synchronization Issues**: Some users may experience slight discrepancies in prayer times due to *timezone-dependent calculations*.
- **Limited API Rate Handling**: Since some data is fetched from **external API**s, rate limits may impact real-time updates.
- **No Mobile Application Support**: The system is currently *web-based only*, limiting accessibility for mobile-first users.

6.3 Future Enhancements

Several improvements can be made to enhance the application's functionality and usability:

- **Progressive Web App (PWA)**: Enhancing the frontend to function as a **PWA** would improve **offline capabilities** and allow **mobile app-like usability**.
- **Push Notifications**: Implementing ***real-time notifications*** for prayer times and Quranic recitations.
- **User Dashboard with Analytics**: Providing users with *data visualization* on their prayer and Quran reading habits.
- **Multi-Language Support**: Expanding the application to support multiple languages for better *global usability.*
- **Decentralized Data Storage**: Storing user preferences and prayer schedules on *local storage* to reduce API dependencies.

The proposed enhancements will make the *Ramadhan Companion App more interactive, efficient, and widely accessible* to the global Muslim community.

Chapter 7

Summary

7.1 Key Contributions

This report has covered the **entire lifecycle** of the **Ramadhan Companion App**, detailing its:

- **System Architecture** – Backend (NestJS, MongoDB) and Frontend (Handlebars, Tailwind CSS).
- **API Endpoints and Logic** – Authentication, Quranic tracking, Prayer schedules, and Fasting schedules.
- **Security and Deployment** – JWT authentication, API security, and Docker containerization.
- **Testing and Validation** – Postman API testing and comparison with external prayer time APIs.

7.2 Final Remarks

The **Ramadhan Companion App** is a *fully functional system* that assists users in *tracking their religious practices with automation and accuracy.* With additional *feature enhancements and optimizations*, the platform can become a *widely adopted tool for Muslims worldwide.*

This document serves as a **technical foundation** for further improvements and research in Islamic digital services.

7.3 References

- **Quran API**: https://alquran.cloud/api
- **OpenStreetMap API**: https://nominatim.org/release-docs/develop/api/Reverse/
- **IslamicFinder Prayer Times**: https://www.islamicfinder.org/
- **Hijri Date Converter**: https://github.com/xsoh/hijri-converter