YourScope

System Design Document

Version 1.0

06/05/2023

Contents

List of Tables

1. General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

1.1 General Overview

The YourScope solution will be a web application that users will be able to access using any web browser on their own computers. The front-end user interface will be made using Angular and will connect with the back-end through exposed endpoints done by APIs written in **.NET 7**. Two services will be used in order to store user data; the first being Firebase Authentication and the second being a MySQL database hosted on AWS. Firebase Authentication will be used to securely store user account information such as email addresses and passwords and other user information such as names, grades, job applications, etc. will be stored on the MySQL database.

The Angular front-end will follow the MVC software design pattern in order to separate the internal logic for handling data from the logic that modifies what the user sees on their screen. The APIs will follow the REST guidelines in order to ensure a greater degree of flexibility, scalability, and simplicity in anticipation of the system growing larger as times goes by.

Design Goals

- **Simplicity:** We will aim to design a system that is simple and intuitive to use, allowing clients to quickly learn and make use of the system.
- **Modularity:** Our system will aim to be made up of self-contained entities that can be accessed and manipulated through the exposed endpoints using the API; this will reduce the chances of creating overly dependent and tightly coupled software.
- **Reliability:** We will design a system that is depending and will perform its functions consistently; as a step in this direction, we will follow the **open-closeed** principle of software design, ensuring that any existing software will not need to be modified in order to add new features or change features.

Types of Users

In order to meet product needs, we will have **three (3)** types of users with respect to our solution. The tree types of users are: **Students**, **Employers**, and **Admins**; the following is a description of each type of user.

1. **Students**

   These users represent high school students and they will have access to the event information of their affiliated school, courses for their school, the course selection feature, a resume builder and cover letter builder, and a list of relevant job postings.

   Students will be able to submit applications to job postings through YourScope.

2. **Employers**

   These users represent recruiters from companies who wish to use YourScope in order to seek new hires.

   Employers will be able to create the job postings that students will be able to see and apply to as well as view these applications through YourScope.

3. **Admins**

These users represent the administrative staff of a high school and their accounts will be created manually by the YourScope team should a school wish to partner with YourScope.

Admins will be able to manage the list of courses offered at their school as well as add and remove upcoming events to the system so that students will be able to see them.

1.2 Constraints

1.2.1 Constraints

Due to the nature of the solution as a web application, users of the software must have access to a stable internet connection at all times during its usage. The database will be hosted on AWS and Firebase Auth will be made use of; both of these services will use the free tier, and so at this point, there may be a limited number of users that we may have concurrently using the application.

2. System Architecture and Architecture Design

This section outlines the system and hardware architecture design of the system.

2.1 Hardware Architecture

Our solution will make use of the Firebase Auth service provided by Firebase on its free tier in order to perform user authentication and a MySQL database hosted on AWS using its free tier. Furthermore, the back-end APIs will be deployed using AWS as well.

2.2 Software Architecture

2.2.1 Security Software Architecture

In terms of security, initially the back-end API will expose two endpoints that do not require authentication in order to be used, and these two endpoints will be responsible for user logins and registrations. Once a user has registered an account, the client should call the login endpoint next in order to authenticate the user with our system.

Once the login endpoint has been called, a response containing a JWT generated by Firebase during the authentication process will be sent back to the client. Further calls to the API will make use of an “Authentication” header that will contain this token in order to access further internal resources (viewing events, job postings, etc).

Both of the login and registration endpoints will require POST calls in order to access their resources and the user information for both of these endpoints will be passed in the request body.

2.3 Information Architecture

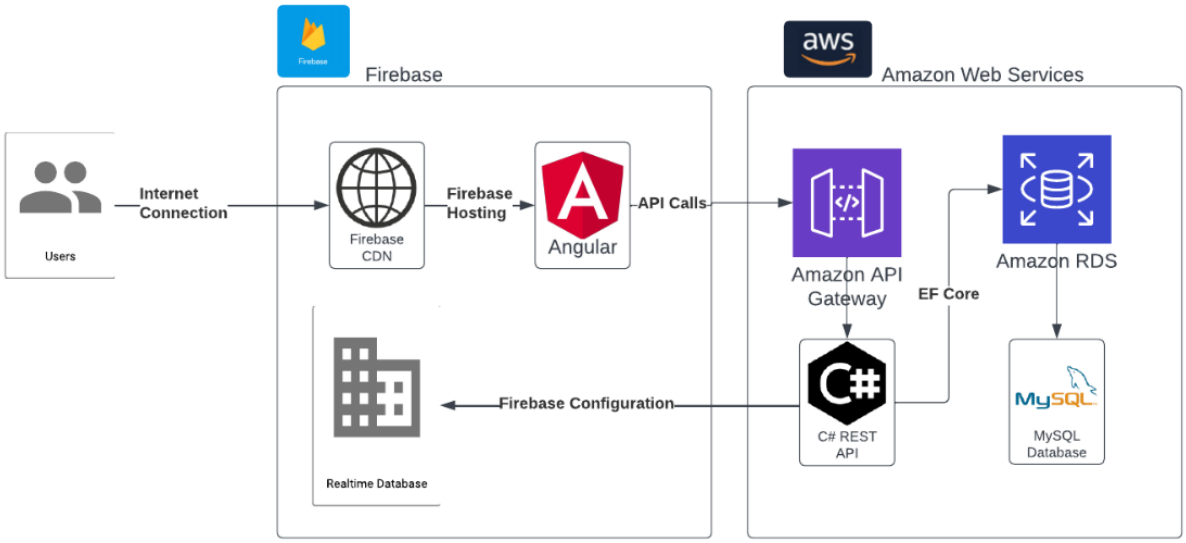
Firebase Auth will be handling user authentication and so a user’s password will be stored securely on their database. With respect to users specifically, email addresses, full names, grade levels, and names of affiliated schools will be stored in the MySQL database.

When an employer creates an account, we will ask that they first register their company with our database prior to the creation of their account (if the company has not already been registered). Information about this company will be stored in the MySQL database as well and includes: the name of the company, the address of the company, and primary contact information.

When an employer decides to create a job posting information about this posting will be stored on the MySQL database and this information will include: the name of the position, the description of the job, qualifications, the closing date, and the name of the company.

We will store a large collection of courses offered to high schools in Canada, which Admins will be able to pick and choose from in order to add to the list of offered courses at their schools. These course information sets will include: the course name, the course code, the description of the couse, and any prerequisuites.

2.4 System Architecture Diagram



3. System Design

3.1 Business Requirements

Functional Requirements

- A user (student and employer) should be able to **sign up** and **log in** into the system in order to access the dashboard containing their respective features.
- A **student** should be able to view upcoming events at their school and they should be able to view details about such events.
- A **student** should be able to use the resume builder and cover letter builder on the YourScope application in order to create job application documents.
- A **student** should be able to apply to job posthings through the YourScope application.
- A **student** should be able to select their courses and keep track of them through the YourScope application.
- An **employer** should be able to create new job postings and view applications to these postings.
- An **admin** should be able to add and remove offered courses at their respective schools.
- An **admin** should be able to add and remove upcoming events at their respective schools.

Nonfunctional Requirements

- The software should be scalable, reliable, and durable.
- The system should not crash as a result of small errors relating to issues such as improper user input.

3.2 Database Design

We will use Firebase's realtime database and other services to handle authentication. We use a MySQL database (hosted with Amazon RDS) to store all other data such as schools, companies, events, job postings, courses and all other data not relating to authentication. In the event that the API is unable to retrieve data (if the database is down, or firebase authentication is not working) we return an error status code to let the angular front-end know. This data will then be passed on to the user letting them know there is an issue and to try again later. All API endpoints have error handling in these exceptional cases. We are also given alerts if either firebase or AWS unexpectedly fails.

With respect to interacting with the MySQL database, we will apply a code-first approach and make use of Entity Framework Core in order to interact with the database.

1.1.1 Data Objects and Resultant Data Structures

User Object

Class: User	
Parent class: None Sub-classes: UserLoginDto, UserRegistrationDto	
Responsibilities: • Represents a single YourScope user and contains information regarding their profile (email, affiliated school, etc) • Does NOT contain their password	Collaborators:

Data representing a single user (regardless of type) will be stored as a C# class in the backend API with each field corresponding to a single column of the Users table in the MySQL database. Fields will include: first name, middle name, last name, birthdate, grade (optional), and affiliation (the name of the school or company the user goes to or works at).

UserLoginDto and UserRegistrationDto

Class: UserLoginDto	
Parent class: None Sub-classes: None	
Responsibilities: • Contains all the data needed for the backend API to login a user account (email and password, etc)	Collaborators: • Auth

Class: UserRegistrationDto	
Parent class: User Sub-classes: None	
Responsibilities: • Contains all the data needed for the backend API to register a new user account (email and password, etc)	Collaborators: • Auth

When passing input data to the registration and login endpoints from the Angular front end, we will make use of two data transfer object classes, one of which extends the base User class. The UserLoginDto will contain email and password fields in order to provide login information, and the UserRegistrationDto extends the User class (because all user information is required for registration) and adds an extra password field.

Courses and Schedules

Class: Course	
Parent class: None Sub-classes: None	
Responsibilities: • Represents a single high school course and contains information about it (course code, title, description, etc) • Will contain a list of tags that the system will be able to use to suggest these features to users. • Will contain a collection of prerequisite courses so that users will know which courses they need to take before taking this one.	Collaborators: • CourseSchedule

Class: CourseSchedule	
Parent class: None Sub-classes: None	
Responsibilities: • Represents a student's course schedule for a single semester of school and contains information about the semester (Fall 2023, etc) • Will contain a list of Course objects that will indicate which courses a student has in their schedule	Collaborators: • Course

A high school course will be represented as a C# in the backend API with each field corresponding to a single column of the Courses table in the MySQL database. Fields include: course code, course title, course description, tags, and prerequisites.

A user will also be able to keep track of the courses they are taking during a semester, which will be stored in the **CourseSchedule** object class. This class will contain an identifier that indicates the semester of study that the courses stored within were taken (e.g. Fall 2023).

EventObject

Class: EventObject	
Parent class: None Sub-classes: None	
Responsibilities: • Represents a school event and stores the data of a single event (location, name, description, etc)	Collaborators:

Data representing a single school event will be stored as a C# class in the backend API with each field corresponding to a single column of the Events table in the MySQL database. Fields will include: the name of the event, its location, its description, and its time.

Each event will also include the unique identifier of the school that it is associated with in order to create a relationship between the two.

HighSchool Class

Class: HighSchool	
Parent class: None Sub-classes: None	
Responsibilities: • Represents a high school and contains information about such a school (name, address, contact, etc) • Will also contain a collection of courses that is offered at the school • Will store the events that are being hosted at the school • Able to provide the above information to users as needed.	Collaborators: • EventObject • Course

A high school will be represented as a C# class in the backend Api with some fields corresponding to a column of the Schools table in the MySQL database. Fields will include: the name of the school, its address, a primary contact number, a collection of offered courses, as well as a list of upcoming events. Events and courses will be stored in their own tables. Each high school will also contiain a unique identifier that will be used in other data types that have relations with the high school.

Jobs Postings and Applications

Class:JobPosting	
Parent class: None Sub-classes: None	
Responsibilities: • Contains fields that describe information about a single job posting (position title, company name, etc) • Will be able to keep track of its close date and have methods that will set it to closed	Collaborators: • JobApplication

Class:JobPosting	
Parent class: None Sub-classes: None	
Responsibilities: • Contains fields that describe information about a single job posting (position title, company name, etc) • Will be able to keep track of its close date and have methods that will set it to closed	Collaborators: • JobApplication

Employers will be able to create job postings, and these postings will be stored using a C# class in the backend API, with each field corresponding to a column in the JobPostings table in the MySQL database. Fields will include: the name of the position, the description of the job, qualitifications, company name, and posting close date. Each job posting will have a unique identifier that will be used in order to form a relationship with applications belonging to it.

Students will see these postings and they will be able to apply to them through YourScope; the information contained within these applciations will be stored in the **JobApplication** C# class in the backend API as well as the JobApplications table in the MySQL database. Fields include: name of the applicant, resume and cover letter information, contact information, as well as a the identifier of the posting it belongs to.

Document Builders

One of the features of YourScope is the ability to dynamically create resumes and cover letters for students to use in their job search.

Interface: IDocumentBuilder	
Parent class: None Sub-classes: ResumeBuilder, CoverLetterBuilder	
Responsibilities: - Acts as an interface for the construction of useful documents (cover letters, resumes, etc) that will help a student with their schooling career.	Collaborators: • ResumeBuilder • CoverLetterBuilder

The above entity will act as an interface for both the resume and cover letter builders that will be used in our solution.

Class: ResumeBuilder	
Parent class: IDocumentBuilder Sub-classes: None	
Responsibilities: • Using some AI, takes input and helps the user build their resume.	Collaborators: • Some AI (ChatGPT)

Class: CoverLetterBuilder	
Parent class: IDocumentBuilder Sub-classes: None	
Responsibilities: • Using some AI, takes input and helps the user build their cover letter.	Collaborators: • Some AI (ChatGPT)

Our implementation of the resume builder and cover letter builder will include calls to the external ChatGPT API in order to assist in the content generation process for the respective documents. The user will provide input in the form of answers to questions about their past experiences, skills, etc.

Controllers and Services

We will be making use of **ASP.NET Core**'s **ApiControllers** in order to manage the different types of endpoints and routes that we will use in our API. For context, **.NET 7** provides a **ControllerBase** base class that we may inherit for our purposes. Each isolated feature of YourScope will have its own controller within the backend API.

Furthermore, in order to reduce coupling of components, increase code readability, and explicitly break down modules into their respective responsibilities, we will make use of **dependency injection** within the design and implementation of the RESTful API. Each controller will receive a dependency injection of an interface representing the service layer of the controller, which will hold within it the majority of the business logic.

For example, the AccountsController will be responsible for managing the endpoints related to user accounts, and in order to perform its functions (such as changing a password for a user account), it will make use of methods in the injected AccountsService interface.

Note: We choose to inject interfaces instead of concrete implementations in order to help comply with the **open-closed** principle, and in the case we need to change the implementation of a Service class for one controller but not another, we will simply implement a new class.

AccountsController

Interface: AccountsController	
Parent class: None Sub-classes: ControllerBase	
Responsibilities: • Provides routing and manages the entryways for each endpoint within our backend API • Controls and manages any exceptions thrown within the service layer of the API.	Collaborators: • AccountsService • MySQL database • Firebase • Auth

For all endpoints relating to the management of user accounts (creation, login, modification, etc), the endpoints will be managed by the AccountsController class, which extends the ASP.NET ControllerBase base class.

IAccountsService and AccountsService

Interface: IAccountsService	
Parent class: None Sub-classes: AccountsService	

Responsibilities:
• Provides an interface for all the services needed in terms of user account management, which include:
  • Student Registration
  • Employer Registration
  • User login
  • User password changing
	Collaborators:
	• Firebase
	• Auth

Class: AccountsService	
Parent class: IAccountsService Sub-classes: None	

Responsibilities:
• Provides all the services needed in terms of user account management, which include:
  • Student Registration
  • Employer Registration
  • User login
  • User password changing
	Collaborators:
	• Firebase
	• Auth

The **IAccountsService** interface and its implementation **AccountsService** will handle all of the business logic regarding operations involving user accounts. This includes calls to the external Firebase Auth service, as well as modifications to the MySQL database.

EventsController

Class: EventsController	
Parent class: ControllerBase Sub-classes: None	
Responsibilities: • Provides routing and manages entryways for each endpoint relating to events within our backend API • Controls and manages any exceptions thrown within the service layer for events.	Collaborators: • SQL database • EventsService

For all endpoints relating to the management of school events (creation, deletion, getting paginated lists of events), the endpoints will be managed by the EventsController class, which extends the ASP.NET ControllerBase base class.

IEventsService and EventsService

Class: IEventsService	
Parent class: None Sub-classes: EventsService	
Responsibilities: • Acts as an interface for the service layer of the Events component of the backend API.	Collaborators: • EventsController

Class: EventsService	
Parent class: IEventsService Sub-classes: None	
Responsibilities: • Provides implementations for methods that will be used in calls to the Events portion of the API. • Functionality includes: creation and deletion of events, retrieving paginated list of all events coming to a school, etc.	Collaborators: • SQL database • EventsController

Functionality related to the endpoints regarding school events will be placed within the **EventsService** class, and will be injected into the controller through the **IEventsService** interface. Functionality includes reading and writing event data from the database.

JobPostingsController

Class: JobPostingsController	
Parent class: ControllerBase Sub-classes: None	
Responsibilities: • Provides routing and manages entryways for each endpoint relating to job postings within our backend API • Controls and manages any exceptions thrown within the service layer for job postings.	Collaborators: • SQL database • JobPostingsService

For all endpoints relating to the management of job postings (creation, deletion, getting paginated lists of job postings, creating job applications, removing job applications, getting list of

job applications under a given posting), the endpoints will be managed by the JobPostingsController class, which extends the ASP.NET ControllerBase base class.

IJobPostingsService and JobPostingsService

Class: IJobPostingsService	
Parent class: None Sub-classes: JobPostingsService	
Responsibilities: • Acts as an interface for the service layer of the job postings component of the backend API.	Collaborators: • JobPostingsController

Class: JobPostingsService	
Parent class: IJobPostingsService Sub-classes: None	
Responsibilities: • Provides implementations for methods that will be used in calls to the job postings portion of the API. • Functionality includes: creation and deletion of job postings, creation and deletion of job applications, fetching a paginated list of job applications, fetching a paginated list of job postings.	Collaborators: • SQL database • JobPostingsController

Functionality related to the endpoints regarding job postings will be placed within the **JobPostingsService** class, and will be injected into the controller through the **IJobPostingsService** interface. Functionality includes creating and deleting job postings, applying to and dropping out from job postings (creating and deleting job applications), as well as fetching paginated lists of job postings and job applications.

CoursesController

Class: CoursesController	
Parent class: ControllerBase Sub-classes: None	
Responsibilities: • Provides routing and manages entryways for each endpoint relating to high school courses within our backend API • Controls and manages any exceptions thrown within the service layer for courses.	Collaborators: • SQL database • CoursesService

For all endpoints relating to the management of high school courses (getting course information based on course code, adding and removing courses to schedule, etc), the endpoints will be managed by the CoursesController class, which extends the ASP.NET ControllerBase base class.

ICoursesService and CoursesService

Class: ICoursesService	
Parent class: None Sub-classes: CoursesService	
Responsibilities: • Acts as an interface for the service layer of the courses component of the backend API.	Collaborators: • CoursesService • Courses Controller

Class: CoursesService	
Parent class: ICoursesService Sub-classes: None	
Responsibilities: • Provides implementations for methods that will be used in calls to the courses portion of the API. • Functionality includes: getting the information of a course given a course code, adding and removing courses from a schedule, etc.	Collaborators: • SQL database • CoursesController

Functionality related to the endpoints regarding courses will be placed within the **CoursessService** class, and will be injected into the controller through the **ICoursesService** interface. Functionality includes: getting the information of a course given a course code, as well as adding and removing courses from a schedule.

1.1.2 Data Conversion

The bodies of our requests and reponses will be returned in JSON format, and so before the API returns any objects, we will first serialize them into JSON format.

Appendix A: Record of Changes

Table 1 - Record of Changes

Version Number	Date	Author/Owner	Description of Change
1.0	*06/05/2023*	*Jason Su*	*Initial creation of document.*