Design Document:

AE Project Management Tool

Group 2: JSON Bourne

Date: April, 7th, 2020
Version: 1.1.0

Document Information

Revision History

Version	Date	Status	Prepared By	Comments
1.0.0	02/06/2020	Completed	JSON Bourne	N/A
1.0.1	02/11/2020	Completed	Jacqueline	Updated the ER diagram
1.0.2	02/24/2020	Completed	Katrina	Updated Client side architecture diagram
1.0.3	02/24/2020	Completed	Jacqueline	Updated Server-side architecture diagram
1.1.0	04/06/2020	Completed	Jacqueline	Updated: - ER Diagram - Server-side architecture diagram - Forecasting APIs

Document Control - Team Participants

Role	Name	Email
Project Manager/Developer	Jacqueline Yin	yin.sangxu@gmail.com
Backend Developer	Jenessa Tan	jenessatan@alumni.ubc.ca
Backend Developer	Bang Chi Duong	bcduong@students.cs.ubc.ca
UX-UI Designer/ Front-end Developer	Katrina Tan	katrinatan91@yahoo.com
Front-end Developer	Lisa O'Brien	lisa.obrien@alumni.ubc.ca
Front-end Developer	Stacy Leson	stacy.leson@gmail.com
Full Stack Developer	Nicole Kitner	nicolekitner@gmail.com

Stakeholders / Signoff list

Role	Name	Signature	Sign-Off Date
Supervisor	Dave Pagurek		
Sponsor	Nash Naidoo		
Sponsor	Steve Robinson		
Sponsor	Shawn Goulet		
Sponsor	Eric Ly		

Table of Contents

Introduction

Overview

As one of the largest engineering consultants in Canada, Associated Engineering faces a challenge in which their current system of seeking and allocating resources requires manual effort, making for an inefficient and time-consuming process.

To minimize the extensive time needed to ensure that appropriately skilled individuals are assigned to suitable projects and/or initiatives, AE is requesting for an implementation of a resource utilization optimization system through which their staff members will be able to insert, retrieve, and organize relevant information on their engineers/staff and projects.

Project Objectives and Goals

Our goal is to design and implement a resource utilization optimization system that allows for the easy tracking of an individual's skills and utilization by a Resource or Project Manager.

Assumptions

- Azure has a backup of our database
- Azure will host our application and be available 24 hours per day
- Maximum capacity is 1000 users
- Program must be able to run on IE11 and Chrome v79

Environments

Programming Environment

- **OS:** Microsoft Windows 10
- **Server side:**
    - C# 8.0
    - .Net Core 3.1
    - IDE: Visual Studio Community v16.0.29519.181
- **Client side:**
    - Node.js v10.16.3
    - npm v6.9.0
    - React v16.12.0
    - IDE: Visual Studio Code v1.41.1
    - Material-UI v4.9.1
- **Database:**
    - SQL Server Express v14.0.1000.169
    - IDE: SQL Server Management Studio v18.4
- **API:** Azure Active Directory[1]
- **Source Control:** GitLab Enterprise Edition v12.7.0
- **Testing:** Postman v7.17.0, xUnit v2.4.1, Enzyme v3.11.0
- **UML Tools:** draw.io v12.6.2

Production Environment

- **API:** Azure Active Directory
- **Web Server (Azure):** Windows Server with IIS 10.0
- **Database:**
    - Cloud SQL Server Express v14.0.1000.169
    - IDE: SQL Server Management Studio v18.4

Test Environment

The test environment will be the same as the production environment.

[1] We could not find the version for the Azure Active Directory

Technical Requirements

System Availability

System is expected to have necessary availability during regular work hours Monday to Friday across different time zones in Canada. Address single point of failures as follows:

Single Point of Failure	Type	Mitigation/Planned response
Server crash	Hardware failure	Use multiple servers for redundancy
Network failure	Hardware failure	Contact communications provider or access another network
Power failure	Hardware failure	Have access to multiple circuits so if one is overloaded and fails, others can still be used
Azure Active Directory Application Proxy failure	Software failure	Follow the troubleshooting guide or contact directly[1]
Database deleted or corrupted	Hardware failure	Keep copy of uncorrupted database

1 https://docs.microsoft.com/en-us/azure/active-directory/manage-apps/application-proxy-troubleshoot

Capacity

The average concurrent users of the system would be about 100 with a peak of 1000 users (approximate number of employees at AE). The database will be hosted on Microsoft SQL Server (Azure) and the capacity will depend on the storage size and memory availability provided by Azure.

Performance

Response Time

In supporting 1000 concurrent users, performance should not fall below the following level:

95% of all visible pages respond in 8 seconds or less, including requests to the Database.

Measurement Points:

- The client-side response times will be measured using Google's Pagespeed Insights. This will measure the time from the request for a page to when the last bit required to render the page is returned.

- The backend response times will be measured with Locust.io (or similar tool).
- The response time required to retrieve information from Azure Active Directory is excluded from measurement, as Azure is a 3rd party product.
- The test workload will be based on a normal business day (and as AE is a company with offices across Canada, the hours will range from 5am to 6pm Pacific Time, to accommodate the Eastern offices).

Scalability

Azure allows for increased capacity if needed, in response to needing the ability to support more concurrent users. Another way to scale horizontally would be adding multiple servers, which is also a possibility. In addition, based on the architecture of the system, it is easy to add more modules to the application in order to extend the scope of the capabilities.

Security

The application will be using the Azure Active Directory (AD) API for authentication and authorization of users. We will be implementing role-based access control (RBAC) to ensure that authenticated users have the appropriate level of access to resources. This will be done by confirming user authorization before providing or displaying any of the resources requested. This includes securing routes on the backend and components on the front-end from unauthorized access. This will provide defense-in-depth. We are assuming that role assignment and account creation and deletion will be done on AD. Below is the planned RBAC matrix:

	Application Resources					
	Users		**Projects**		**Settings**	
Roles	**Own Profile**	**Others' Profile**	**Own Projects**	**Others' Projects**	**Locations**	**Discipline & Skills**
Resource (Regular User)	RU	R	R	R	-	-
Project Manager	RU	RU	CRUD	R	-	-
Administrator	RU	RU	CRUD	CRUD	CRUD	CRUD

Legend:	C: Create	R: Read	U:Update	D: Delete

In addition, we will be following best practices for security such as sanitizing user inputs to ensure that SQL injection attacks to be prevented. An encrypted communication stream between the client and server is beyond the scope of this project but can be achieved by enabling HTTPS and acquiring an HTTPS certificate. In addition, NodeJS and .NET have additional cryptographic modules that enable encryption and decryption using standard cryptographic ciphers such as AES-256, RSA or SHA-256. These modules also make it possible for session key establishment using a Diffie-Hellman key exchange. The crypto modules also make it possible to consider the salting and hashing of sensitive data before storing it in the database. However, at the very least enabling HTTPS and acquiring an HTTPS certificate should be implemented before mass deployment of the application.

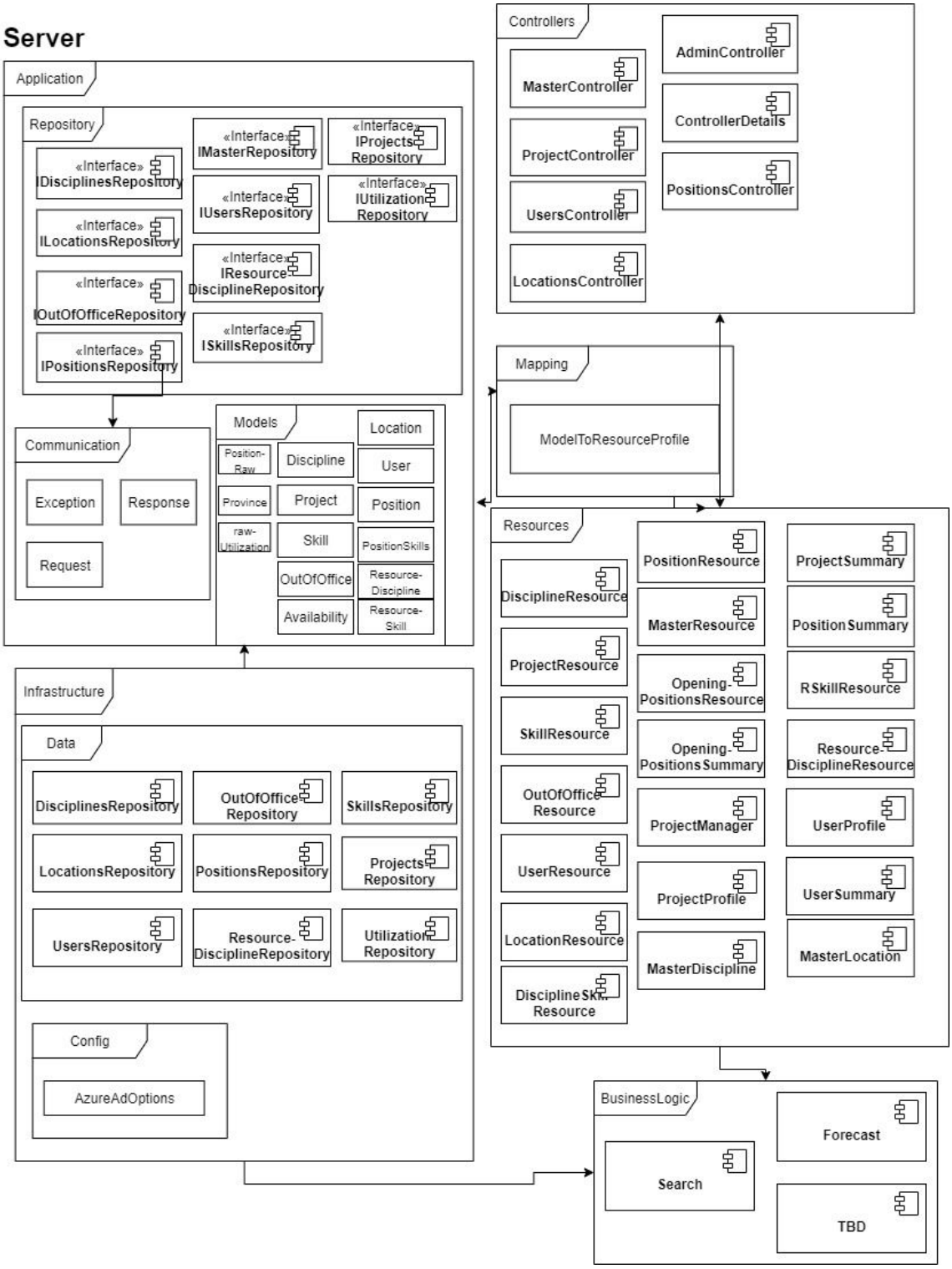
Maintenance

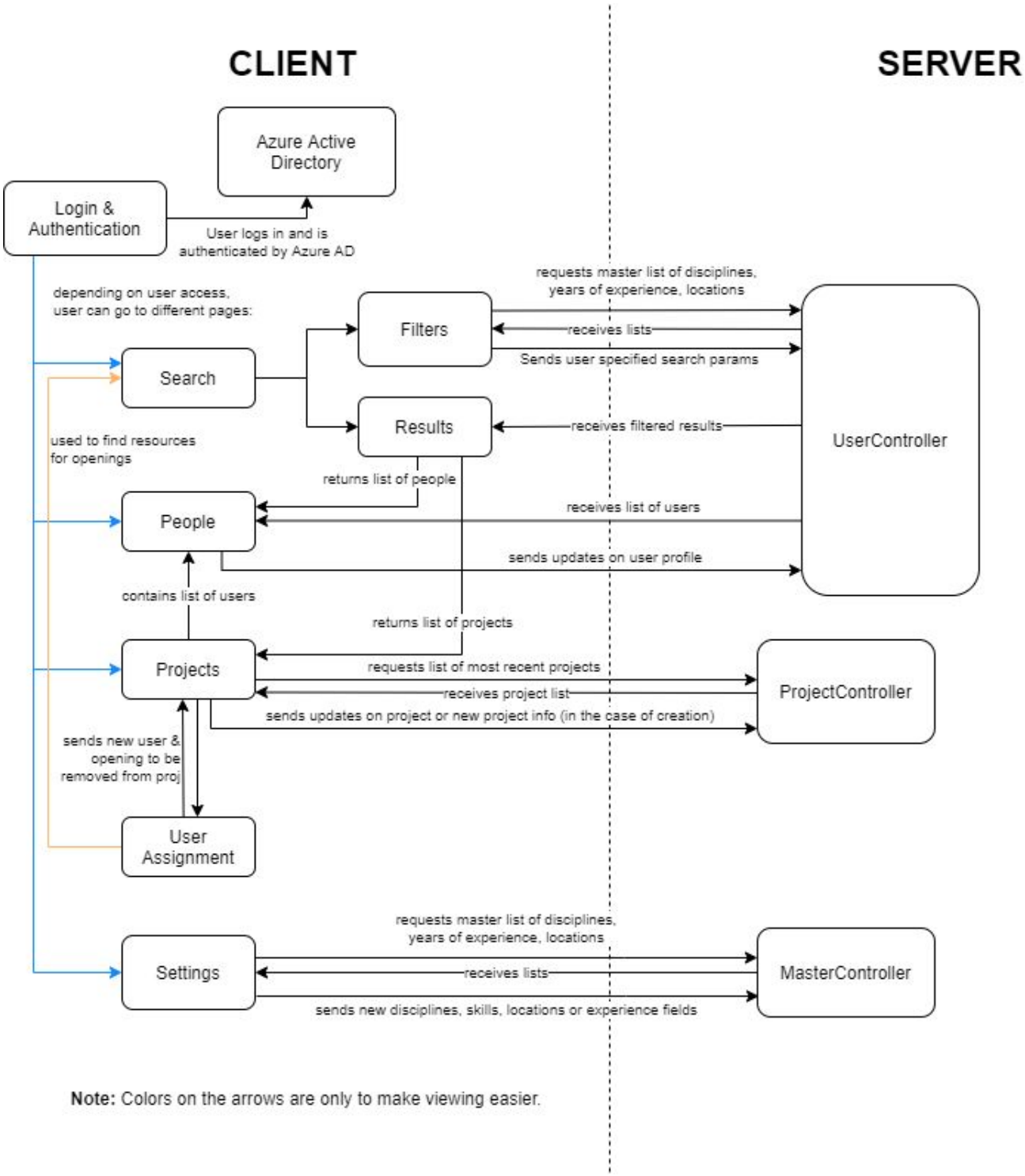
The maintenance concerns mainly with the changing of Server APIs version as well as the Frontend version, together with the Azure Deployment and Azure DevOp Pipeline for Continuous Integration and Continuous Deployment. When deploying the system, we expect to have at least 2 instances of Database running so that we have a backup instance if one goes down. Git Version Control will be used with deployment on the master branch. There will be a “dev” branch that will make pull requests into the “master” branch, such that everything on “dev” is working and tested properly. There will be a main “server” branch and a main “client” branch, and these should make pull requests to the “dev” branch.

UI Design and Mockups

UI Design and Prototype pages are included in the Appendix of the Business Requirements Document

Software Architecture

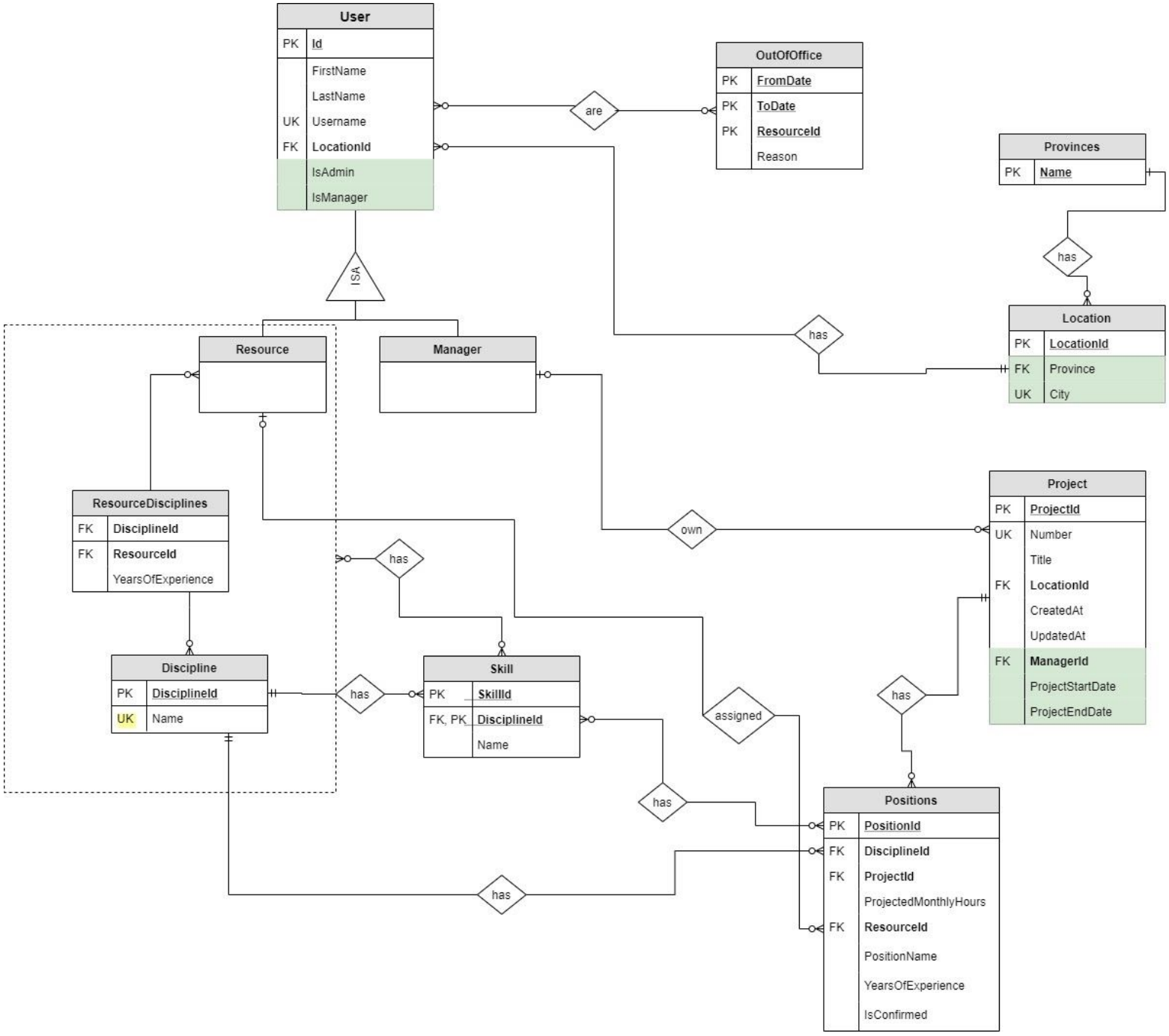




Note: For more details on Client-Side and Server-Side Architecture, please refer to Section 1.1 in the Appendix.

Data Design

Entity Relationship Diagram



Note: We have highlighted in green the attributes that we have added to the original tables given to us by AE in green

API Design

User Module	
Endpoint URL	api/users/{userID}
Method	GET
Description	Retrieves a single user's profile from the server.
URL Parameters	userID = integer
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	payload: UserProfile { UserSummary, Disciplines[ ], Projects[ ], Availability[ ] },
	message:
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Non-existent User:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Search Module	
Endpoint URL	api/users?key={filtertype}&*[{filtertype}={filterValue}]&order={order}&page={pageNumber}
Method	GET
Description	Returns a list of 50 users that match provided search parameters. The key is a string that determines which filter should determine the order. Order determines if it

	should be ascending or descending. PageNumber determines which set of users are provided. When no parameters are provided, it returns the first set of 50 users sorted according to utilization in descending order.
URL Parameters	Optional Parameters: filterType = [location | discipline | experience | utilization | searchParam | startDate | endDate] ; filterValue = String ; order [ascending | descending] ; pageNumber = integer
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	payload: {
	users: UserProfiles [UserProfile { UserSummary, Disciplines[ ], Projects[ ], Availability[ ] }],
	page: integer
	},
	message:
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Not Found:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	api/projects?key={keyValue}&page={pageNumber}
Method	GET
Description	Returns at most the top 50 projects that match the provided keyValue. When no key value is provided, it returns a list of the top 50 projects whose end dates are after the current date. The projects are sorted according to their start dates.
URL Parameters	Optional: keyValue = String ; pageNumber = integer
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	payload: {
	projects: ProjectProfiles [ProjectProfile { ProjectSummary, UserSummary[ ], Openings[ ] }],
	page: integer
	},
	message:
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Not Found:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Project Module	
Endpoint URL	api/projects/{projectID}
Method	GET
Description	Retrieves a single project profile from the server
URL Parameters	projectID = integer
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	payload: ProjectProfile { ProjectSummary, UserSummary[ ], Openings[ ] },
	message:
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Not Found:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	api/projects/{projectID}/assign/{openingID}
Method	PUT

Description	Assigns a user to an opening and modifies the project
URL Parameters	projectID = integer ; openingID = integer

Request Body	{
	positionID: integerID,
	userID: userID
	}

Successful Response	{
	code: 200,
	status: "OK"
	payload: [openingID, userID]
	message: "Successfully assigned user {userID} to opening {openingID}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	/api/projects/{projectId}

Note: Some API designs have been excluded from this section, as they are similar to the ones shown above. To see them, please refer to Section 2 of the Appendix.

Notable Trade Offs

Cost-Free Development

To reduce costs, we are using the free trials of Azure which means that we can only host a SQL Database up to 250 GB, have a bandwidth of 15 GB, and can only be used for up to 12 months from the start of development. This means that we are only scalable upto a database of 250 GB unless a license is purchased.

Risk

There is a risk of losing database integrity and having database redundancy, we will be normalizing our databases in 3rd Normalized Form to mitigate against these risks.

Our communications between client and server will not be encrypted as this is beyond the scope of our project, but modules will be made available that will allow this in the future (see Security section for more information).

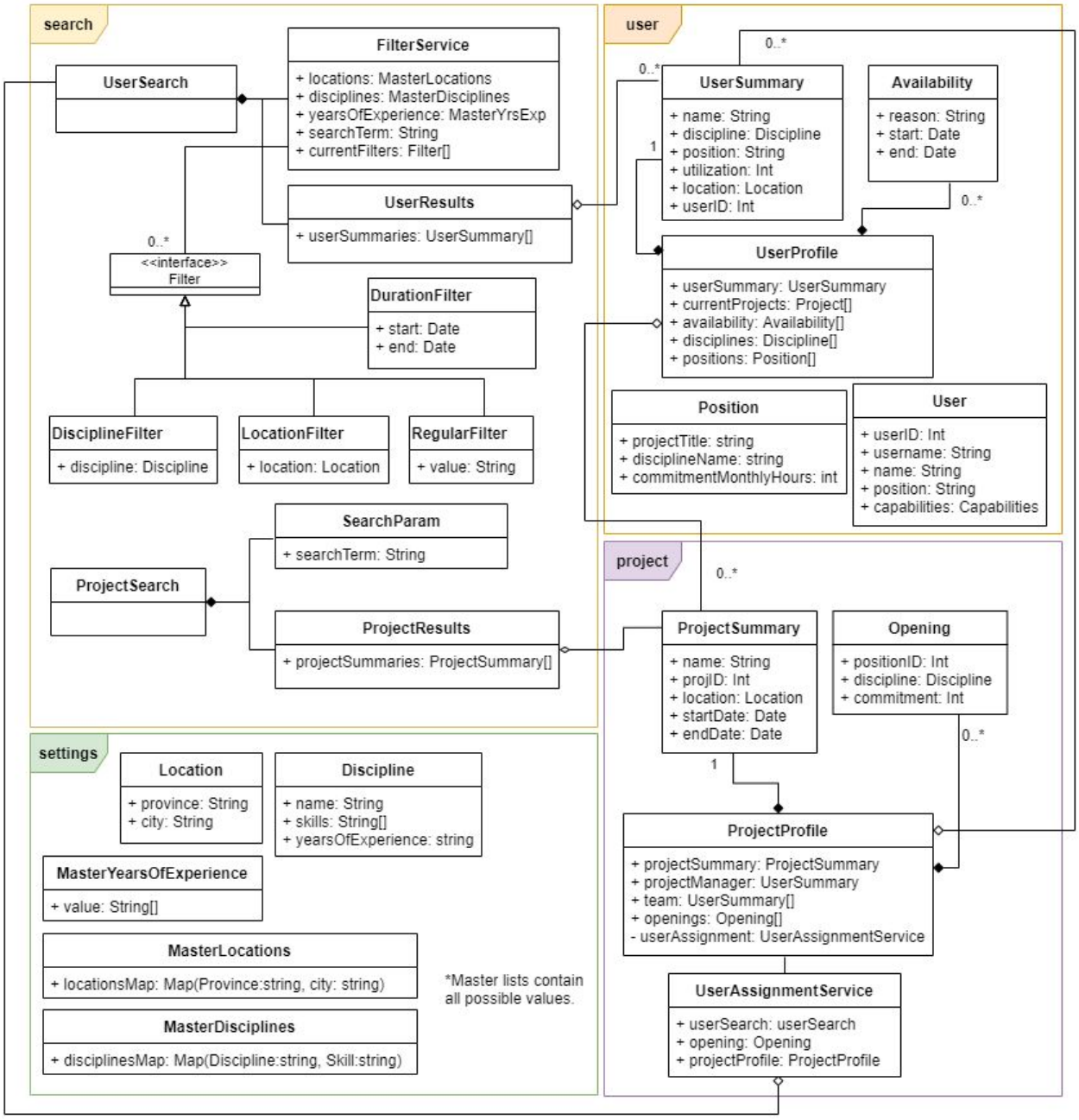
Risk ID	Risk Description	Assessment	Mitigation
1	Unrealistic project component completion timelines	medium	Incremental development, weekly timeline check-ins to stay on track and divert resources as needed
2	Discrepancy between created product and desired product	medium	Sponsors review wire-frames and prototypes. Identify and clarify assumptions that are made
3	Possible corruption of database during development	medium	Keep uncorrupted version of database to reset if needed
4	Private contractor information could be leaked through the system during development	low	Provide testing environment that only contains mock data
5	Database attacks -SQL injection attacks via product	low	Sanitizing user inputs, following best practices for security, include in testing plan

Appendix

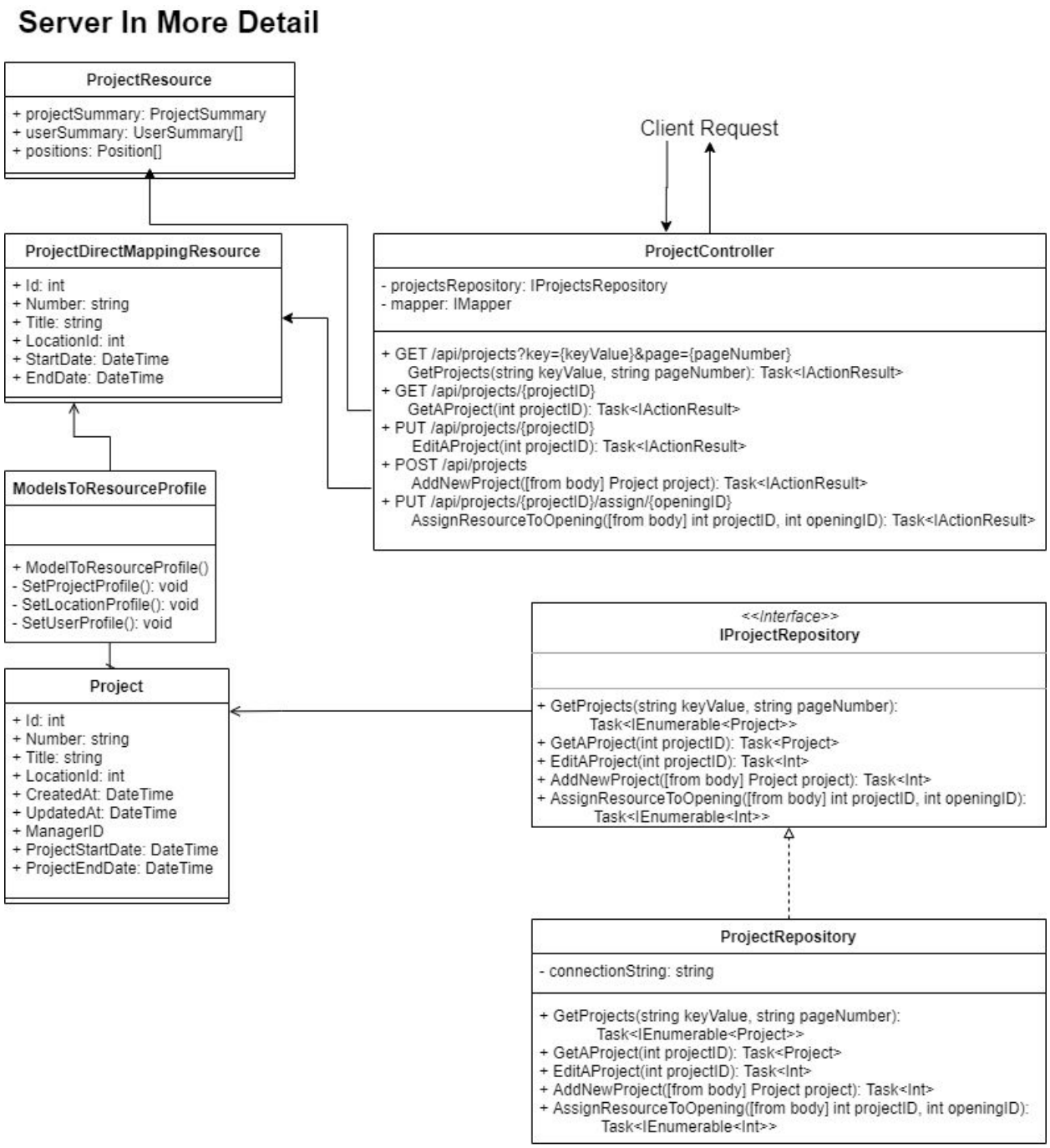
Section 1: Software Architectural Designs

1. Client-Side Architecture

CLIENT



2. Detailed Server



Section 2: API Designs continued...

User Module	
Endpoint URL	api/users/{userID}
Method	PUT
Description	Send modifications of user profile to server
URL Parameters	userID = integer

Request Body	{
	user: UserProfile { UserSummary, Disciplines[ ], Projects[ ], Availability[ ] }
	}

Successful Response	{
	code: 200,
	status: "OK"
	message: "Successfully saved changes to user {userID}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Non-existent User:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Project Module	
Endpoint URL	api/projects/{projectID}
Method	PUT
Description	Edits a single project profile and saves it to the database

URL Parameters	projectID = integer

Request Body	{
	project: ProjectProfile { ProjectSummary, UserSummary[ ], Openings[ ] }
	}

Successful Response	{
	code: 200,
	status: "OK"
	message: "Successfully saved changes to project {projectID}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	api/projects
Method	POST
Description	Saves a newly created project
URL Parameters	

Request Body	{
	project: ProjectProfile { ProjectSummary, UserSummary[ ], Openings[ ] }
	}

Successful Response	{
	code: 200,
	status: "OK"
	message: "Successfully saved created project {projectID}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	/api/projects/{projectId}
Method	DEL
Description	Deletes a project
URL Parameters	projectID = integer
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	message: "Successfully deleted project {projectID}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Not Found:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Common Endpoints (used by multiple modules)	
Endpoint URL	api/masterlists
Method	GET
Description	Retrieve master list of locations, disciplines and skills, and years of experience to populate the form
URL Parameters	
Request Body	

Successful Response	{
	code: 200,
	status: "OK"
	payload: {
	disciplines: MasterDisciplines,
	locations: MasterLocations,
	yearsOfExp: MasterYearsOfExperience
	},
	message:
	extra:

	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Not Found:
	{
	code: 404
	status: "Not Found"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Forecasting Module	
Endpoint URL	api/positions/{openingId}/assign/{userId}
Method	PUT
Description	Assigns a user to an opening and modifies the positions
URL Parameters	openingId = int; userId = int
Request Body	N/A

Successful Response	{
	code: 200,
	status: "OK"
	payload: {openingId, userId, confirmedUtilization}
	message: "Successfully assigned user {userId} to opening {openingId}"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	api/positions/{openingId}/confirm
Method	PUT
Description	Confirms a resource's position on project - modifies position & returns user's updated utilization
URL Parameters	openingId = int
Request Body	N/A

Successful Response	{
	code: 200,
	status: "OK"
	payload: {openingId, userId, confirmedUtilization}
	message: "Successfully confirmed resource on project"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Invalid Parameters:
	{
	code: 400
	status: "Bad Request"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"
	message:
	}

Endpoint URL	api/positions/{openingId}/unassign
Method	PUT
Description	Unassigns a Resource from a position - modifies position & returns user's updated utilization
URL Parameters	openingId = int
Request Body	N/A

Successful Response	{
	code: 200,
	status: "OK"
	payload: {openingId, userId, confirmedUtilization, openingPostionsSummary}
	message: "Successfully confirmed resource on project"
	extra:
	}

Error Response	**Unauthorized Access:**
	{
	code: 401
	status: "Unauthorized Access"
	message:
	}
	
	Connection/Database Error:
	{
	code: 500
	status: "Internal Server Error"

	message: }