Time - Tracking System Report

Shagil Chaudhary

Index

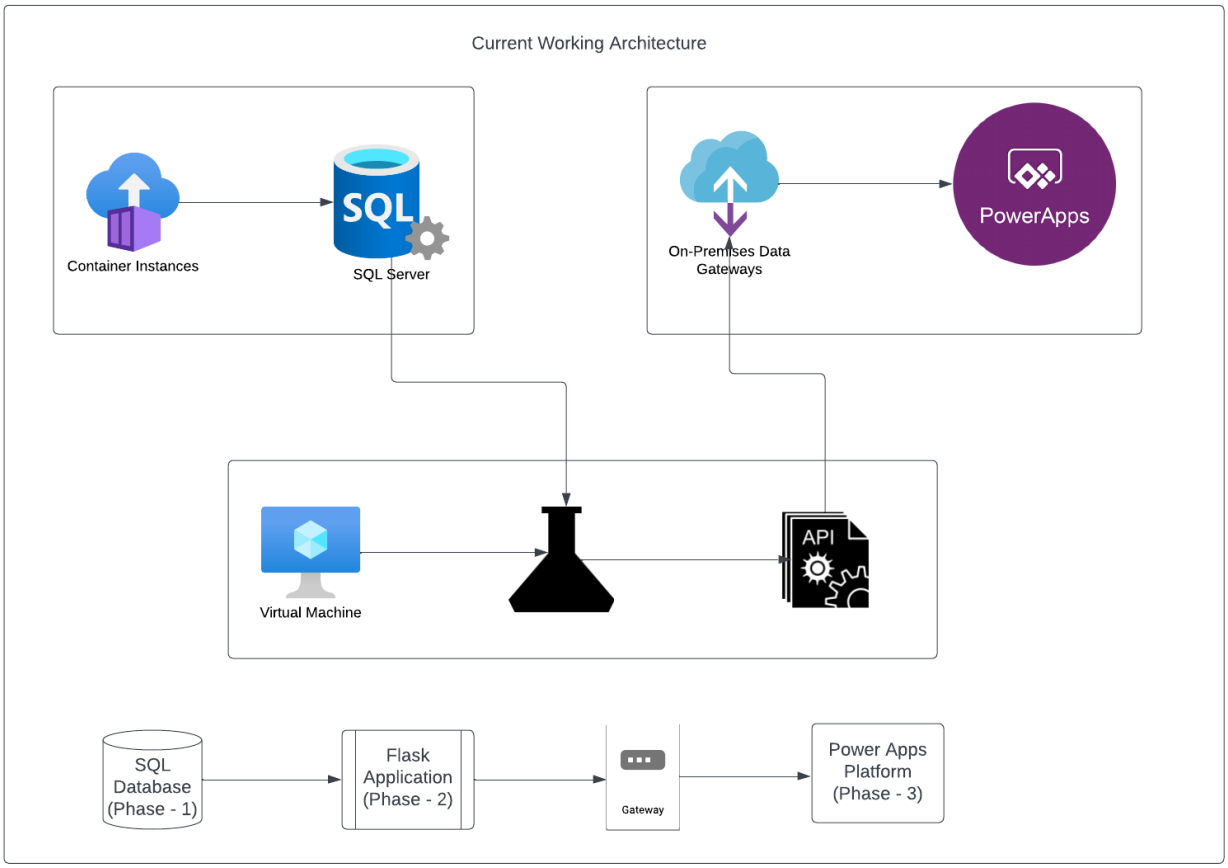
Executive Summary

The aim of the sprints were to setup the architecture and get it running so that all the three sections of the application are communicating to each other in a synchronous harmony. The database where all the information is being stored is designed and maintained in an SQL Server. There will be a section in the report with the database schema described in great details. The next part of the application is designed and maintained in a Flask Application which is the driving part of the application. This Flask Application contains the logic behind the functionality in the terms of API routes. These API routes will then be used to connect the Power Apps platform which constitute the third part of the application where the UI is being designed and the entire application comes together. Currently the graduate was focussing on up-skilling and then moved towards the development of the database schema and the APIs that will be used by the application. The API documentation will be provided in details in the API section.

Tech Stack

The current tech stack consists of the Flask Application running locally on port 5000. Also, the SQL Database is running on SQL Server instance that is deployed on Docker as an instance. This SQL Server instance is being accessed through Azure Data Studio for querying. The User Interface is being designed on the Power Apps Platform that is using the On-Premise Gateway for the communication. (This will be made redundant after the migration to cloud)

Architectural Structure



The current architecture of the application consists of three distinct parts that are responsible for specific functions of the application. The first part is the database which is a SQL Database that is running in SQL Server. This is being achieved currently by installing a SQL Server instance in Docker locally this holds the SQL Database which is being queried by the Flask Application that is working in the next part of the application. The second part consists of the Flask Application which is currently running locally but will be migrated to an Azure Cloud based Virtual Machine that will be responsible for running the Flask Application which would be connected to the SQL Server. Furthermore, the third part of the application is the User Interface that is being developed and deployed on the Power Apps Platform. Currently as the Flask Application is running locally on port 5000 the architecture requires an “On-Premise Gateway” which is responsible for the communication between the APIs and the Power Apps platform. This gateway will be redundant once the first two parts of the application is deployed on the cloud.

Database Schema Documentation

The application is using a SQL Database is called "timesheet2" and have several tables with their respective columns and constraints.

Here's a technical description of the database schema:

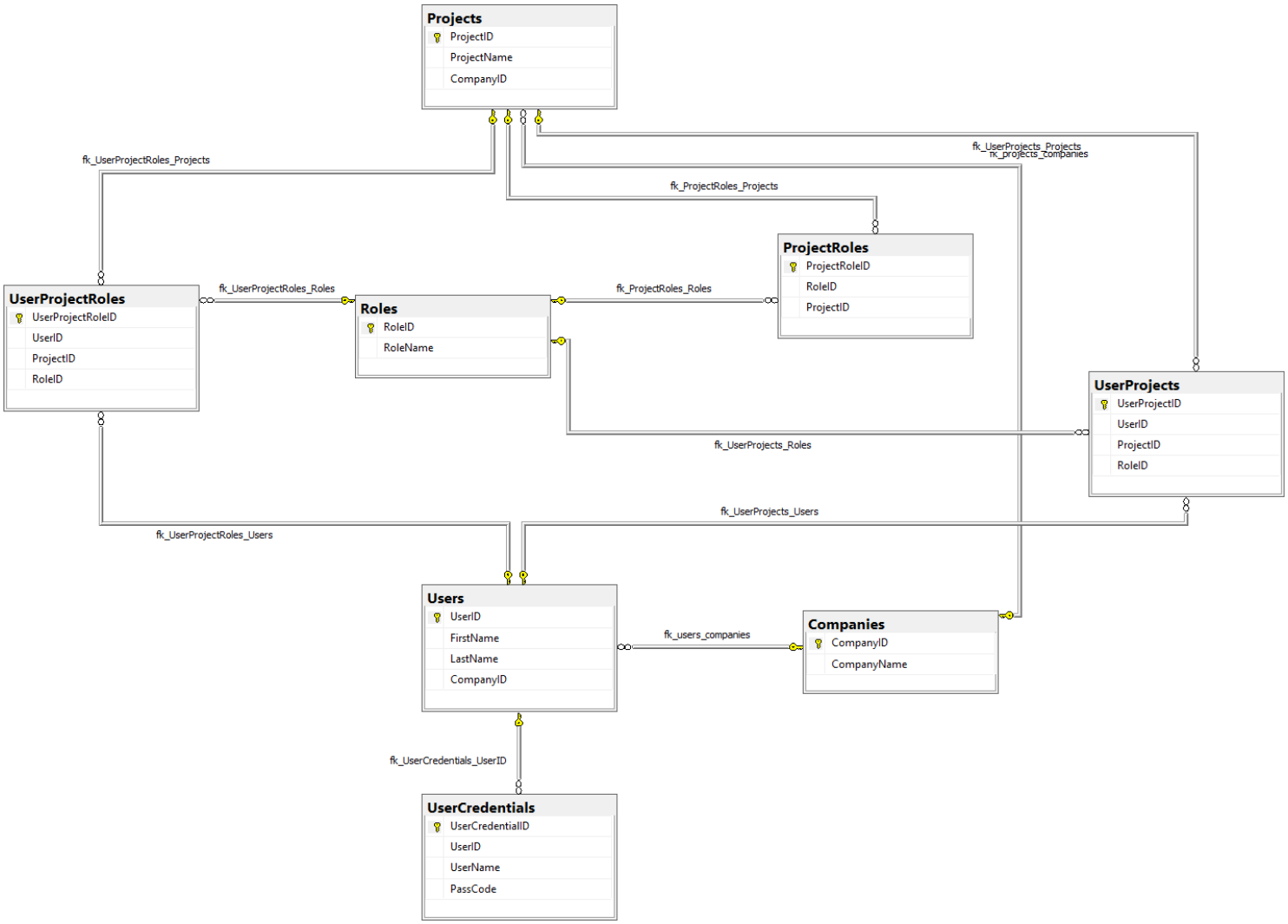
The database contains various relationships between the tables. The Companies, Roles, and Projects tables are referenced by other tables using foreign keys to establish relationships. The Users table references the Companies table, and both UserProjects and UserProjectRoles tables reference Users, Projects, and Roles tables. The ProjectRoles table also references the Roles and Projects tables.

Table Name	Columns	Constraints
Companies	CompanyID (INT, primary key, auto-incrementing), CompanyName (VARCHAR(50), not null)	unique_company (ensures unique company names)
Roles	RoleID (INT, primary key, auto-incrementing), RoleName (VARCHAR(50), not null)	unique_role (ensures unique role names)
Projects	ProjectID (INT, primary key, auto-incrementing), ProjectName (VARCHAR(50), not null),	fk_projects_companies (foreign key referencing Companies table)
	CompanyID (INT, not null)	
Users	UserID (INT, primary key, auto-incrementing), FirstName (VARCHAR(50), not null),	fk_users_companies (foreign key referencing Companies table)
	LastName (VARCHAR(50), not null), CompanyID (INT, not null)	
UserProjects	UserProjectID (INT, primary key, auto-incrementing), UserID (INT, not null),	fk_UserProjects_Users (foreign key referencing Users table),
	ProjectID (INT, not null), RoleID (INT, not null)	fk_UserProjects_Projects (foreign key referencing Projects table),
		fk_UserProjects_Roles (foreign key referencing Roles table),
		unique_UserProject (ensures unique combinations of UserID, ProjectID, RoleID)
UserProject Roles	UserProjectRoleID (INT, primary key, auto-incrementing), UserID (INT, not null),	fk_UserProjectRoles_Users (foreign key referencing Users table),
	ProjectID (INT, not null), RoleID (INT, not null)	fk_UserProjectRoles_Projects (foreign key referencing Projects table),

Table Name	Columns	Constraints
		fk_UserProjectRoles_Roles (foreign key referencing Roles table),
		unique_UserProjectRole (ensures unique combinations of UserID, ProjectID, RoleID)
ProjectRoles	ProjectRoleID (INT, primary key, auto-incrementing), RoleID (INT, not null),	fk_ProjectRoles_Roles (foreign key referencing Roles table),
	ProjectID (INT, not null)	fk_ProjectRoles_Projects (foreign key referencing Projects table),
		unique_ProjectRole (ensures unique combinations of RoleID, ProjectID)
UserCredentials	UserCredentialID (INT, primary key, auto-incrementing), UserID (INT),	fk_UserCredentials_UserID (foreign key referencing Users table),
	UserName (VARCHAR(255), not null), PassCode (VARCHAR(255), not null)	

Additionally, the script inserts sample data into the tables using INSERT statements. It populates the Companies, Roles, Projects, and Users tables with some example records. It also inserts records into the UserProjects and UserProjectRoles tables to establish relationships between users, projects, and roles. Finally, it inserts user credentials into the UserCredentials table, generating a unique passcode using the NEWID() function.

Database Schema Diagram



The above database schema diagram shows the relationship between the different tables consisting the database. The database implements good database structuring practices to make the database as linear as possible and also easily accessible and efficient.

The database schema includes several relationships between the tables. Details of the relationships that are implemented in the database schema is provided below.

Here's a detailed description of the relationships present in the schema:

1. **One-to-Many Relationship: Companies to Projects**

   - The "Companies" table has a primary key column called "CompanyID."
   - The "Projects" table has a foreign key column called "CompanyID" referencing the "CompanyID" column in the "Companies" table.
   - This relationship allows multiple projects to be associated with a single company.

2. **One-to-Many Relationship: Companies to Users**

   - The "Companies" table has a primary key column called "CompanyID."
   - The "Users" table has a foreign key column called "CompanyID" referencing the "CompanyID" column in the "Companies" table.
   - This relationship allows multiple users to be associated with a single company.

3. **Many-to-Many Relationship: Users to Projects with Roles**

   - The "Users" table has a primary key column called "UserID."
   - The "Projects" table has a primary key column called "ProjectID."
   - The "Roles" table has a primary key column called "RoleID."
   - The "UserProjects" table serves as a junction table and has foreign key columns "UserID," "ProjectID," and "RoleID" referencing the corresponding primary key columns in the "Users," "Projects," and "Roles" tables, respectively.
   - This relationship allows multiple users to be associated with multiple projects, each with a specific role. It represents the assignment of users to projects with their respective roles.

4. **Many-to-Many Relationship: Users to Projects with UserProjectRoles**

   - The "Users" table has a primary key column called "UserID."
   - The "Projects" table has a primary key column called "ProjectID."
   - The "Roles" table has a primary key column called "RoleID."
   - The "UserProjectRoles" table serves as another junction table and has foreign key columns "UserID," "ProjectID," and "RoleID" referencing the corresponding primary key columns in the "Users," "Projects," and "Roles" tables, respectively.
   - This relationship represents an additional way of assigning users to projects with their respective roles. It allows for multiple roles to be assigned to the same user on a particular project.

5. **One-to-Many Relationship: Roles to UserProjectRoles and ProjectRoles**

   - The "Roles" table has a primary key column called "RoleID."
   - The "UserProjectRoles" table has a foreign key column called "RoleID" referencing the "RoleID" column in the "Roles" table.
   - The "ProjectRoles" table has a foreign key column called "RoleID" referencing the "RoleID" column in the "Roles" table.
   - These relationships allow a role to be associated with multiple user-project combinations in the "UserProjectRoles" table and multiple project-role combinations in the "ProjectRoles" table.

6. **One-to-Many Relationship: Projects to ProjectRoles**

   - The "Projects" table has a primary key column called "ProjectID."
   - The "ProjectRoles" table has a foreign key column called "ProjectID" referencing the "ProjectID" column in the "Projects" table.
   - This relationship allows multiple roles to be associated with a single project in the "ProjectRoles" table.

7. **One-to-One Relationship: Users to UserCredentials**

- The "Users" table has a primary key column called "UserID."
- The "UserCredentials" table has a foreign key column called "UserID" referencing the "UserID" column in the "Users" table.
- This relationship associates user credentials (username and password)

Flask Restful Plus API Documentation

The application is Flask-based web application that serves as an API for managing users, companies, roles, projects, and user-project relationships. It interacts with a SQL Server database using the pyodbc library for database connectivity.

Here is a breakdown of the technical details and components of the application:

1. **Flask:** Flask is a lightweight web framework for Python that allows building web applications. It is used as the foundation for this application, providing routing, request handling, and response generation.

2. **Routes:** The application defines several routes (API endpoints) using the `@app.route` decorator. Each route corresponds to a specific URL path and HTTP method (GET or POST) and is associated with a handler function that executes the necessary logic.

3. **Database Connection:** The application establishes a connection to a SQL Server database using pyodbc. The connection string specifies the driver, server name, database name, and authentication method (in this case, trusted connection).

4. **Error Handling:** The application defines error handlers for specific types of exceptions. The `@app.errorhandler` decorator is used to associate these handlers with the corresponding error types. The error handlers log the error using Flask's built-in logger and return a JSON response with an appropriate error message and status code.

5. **Logging:** The application utilises the Python `logging` module to log errors and other relevant information. The `basicConfig` function is called to configure the logging settings, specifying a log file name ('app.log') and logging level (DEBUG).

6. **API Endpoints:**

- **`/users`:** This endpoint handles GET requests and retrieves a list of users from the database. The query result is transformed into JSON format and returned as a response. It also includes error handling for database query errors.

- **`/users/all_info`:** This endpoint retrieves detailed information about users, including their roles, companies, and projects. The SQL query joins multiple tables to retrieve the required data. The result is converted into JSON and returned as a response.

- **`/companies`, `/roles`, `/projects`, `/userprojects`:** These endpoints handle GET requests and retrieve lists of companies, roles, projects, and user-project relationships, respectively. The query results are converted into JSON format and returned as responses.

- **`/users/<project_name>`:** This endpoint retrieves users associated with a specific project. The project name is provided as a parameter in the URL. The SQL query joins multiple tables to fetch the relevant data, which is then returned as JSON.

- **`/users`:** This endpoint handles POST requests and allows adding a new user to the database. The request body should include the required information such as first name, last name, role name, project name, and company name. The endpoint performs validation, checks for existing users, and inserts the new user into the database. Appropriate responses are returned based on the success or failure of the operation.

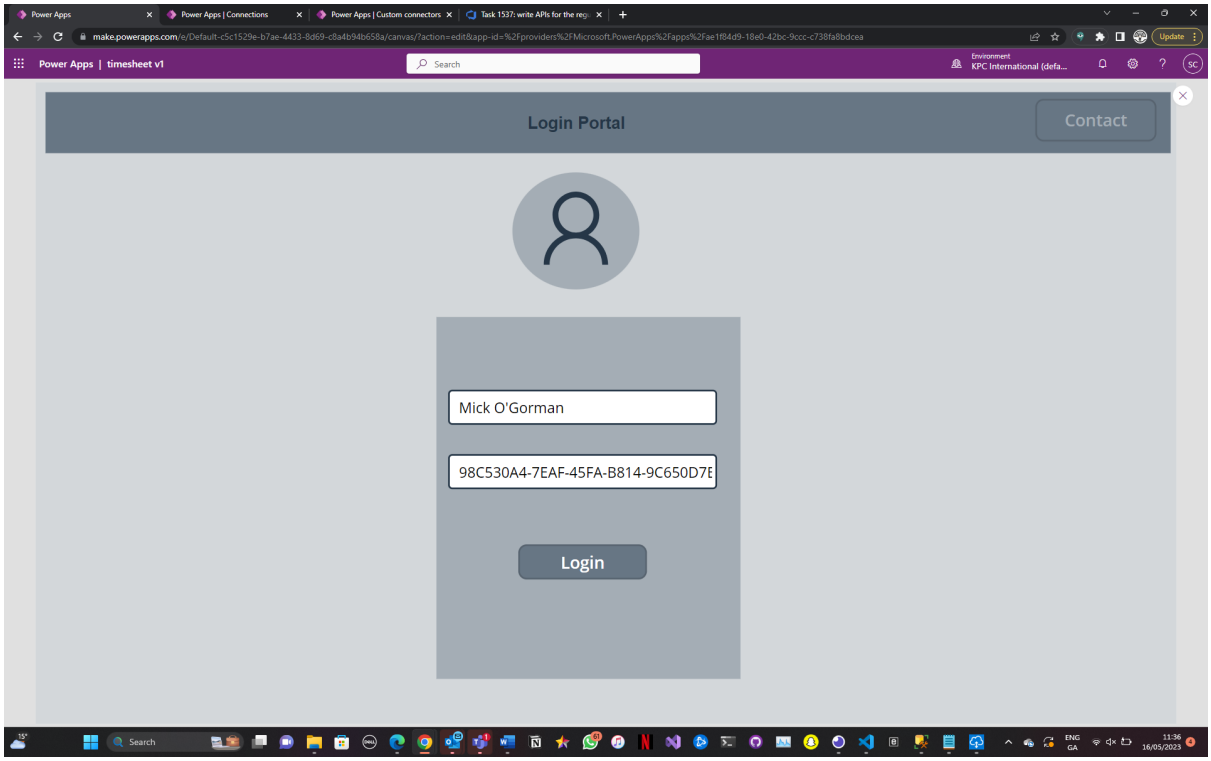
7. **JSON Responses:** The application uses Flask's `jsonify` function to convert Python dictionaries and lists into JSON responses. This ensures that the API responses adhere to the expected JSON format.

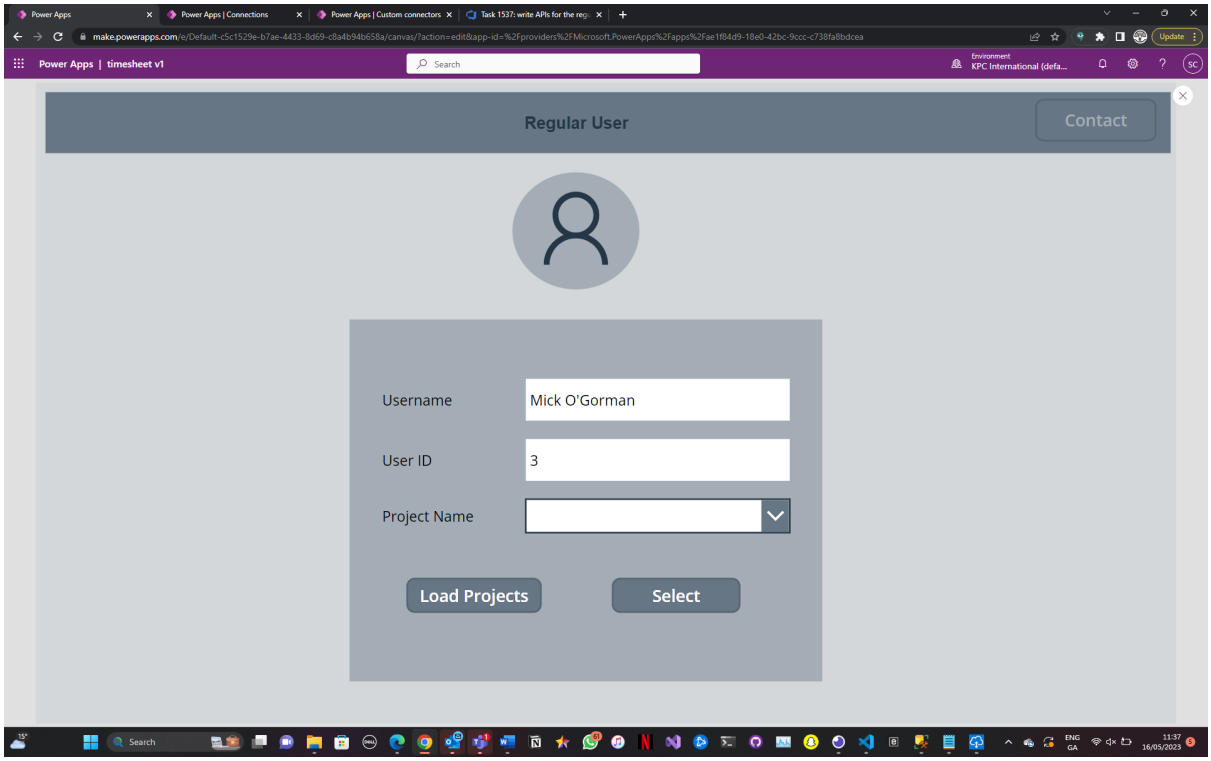
8. **Exception Handling:** The application utilises exception handling to catch and handle specific types of exceptions that may occur during the execution of code blocks. It differentiates between `pyodbc.Error` (database-related errors) and general `Exception` types. The error messages are logged, and appropriate JSON responses with error messages and status codes are returned.

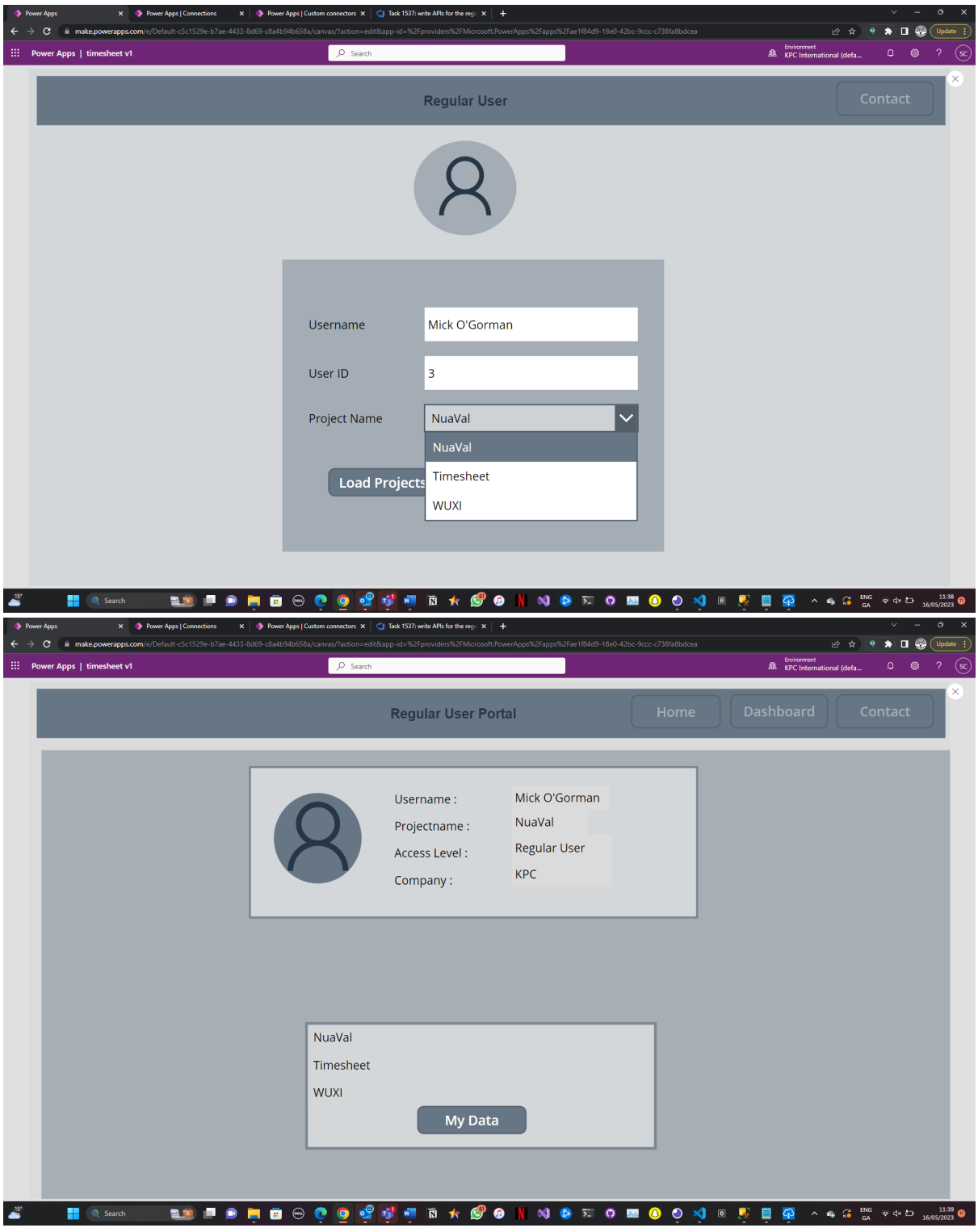
Summary of the Flask Application

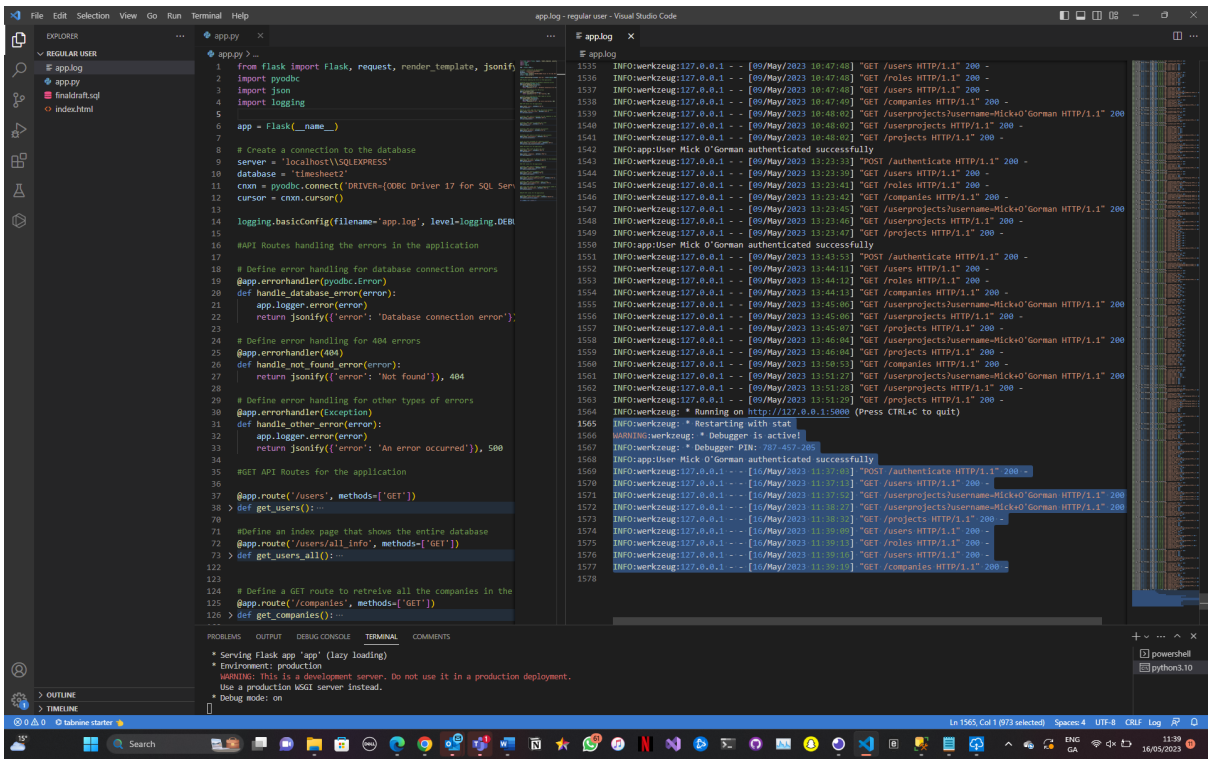
Overall, this application provides a RESTful API for managing users, companies, roles, projects, and user-project relationships, using Flask as the web framework and pyodbc for connecting to a SQL Server database. It incorporates error handling, logging, and JSON response generation to ensure smooth operation and effective communication with clients consuming the API.

Power Apps Status






Power Apps | timesheet v1
Regular User
Contact
Username
Mick O'Gorman
User ID
3
Project Name
NuaVal
NuaVal
Timesheet
WUXI
Load Projects
Regular User Portal
Home
Dashboard
Contact
Username :
Mick O'Gorman
Projectname :
NuaVal
Access Level :
Regular User
Company :
KPC
NuaVal
Timesheet
WUXI
My Data



The above show case the different screens that have been developed in power apps at the moment and are running perfectly. They are using the different APIs defined for different purposes and is using the SQL Database that is running on SQL Server.

The last screen shows the log file entry for every API being called for the different purposes or to populate the different sections of the screen.

Current Status of the Application

Currently the application is working harmoniously with all the three parts of the application. The API is able to query the SQL Server to access the SQL Database. The API is also capable of connecting to the Power Apps Platform using the On-Premise Gateway. There is a Login Page that has been created which is currently using an auto-incremented table in the SQL Database containing the “Username” and the “Passcode” this will later be replaced by the Active Directory on the Power Apps Platform. Furthermore, the Regular User page is running as well in the Power Apps Platform which is using all three parts of the application.
The modification of the database schema is also in place where the timesheet entry part of the application is implemented. Additionally, the API for the same purpose is being created as well which then will be used to create and setup the custom connector in the Power Apps Platform to provide the interface to add the timesheet entry which is the part of the Regular User Tools Portal.
The next part that is in progress is the development of the Project Manager User Relationship and also Manager and User Relationship which would be assigned by the administrator. This leads to the development of the administrator portal.
The last part of the development sprint is development of the approval process logic and implement it before migrating the application on the cloud in order to start with the development of the Project Manager Portal.
At the end of all the sprints the aim would be to rollout the first version of the application that does the basic timesheet processes and would be gathering the data to proceed towards the next phase of the application which would be the development of the timesheet entry that takes into consideration of the details of the task assigned to further add analytical parts in the application as well.

Logic for Timesheet Entry

FirstName	LastName	Office	EmployeeID	Manager Name	WeekStartDate	WeekEnd Date	HoursWorked	Overtime

It was mentioned by Terry that he would require the data in this format to import directly to the payroll system. He does require a project specification section as well.

There was a doubt in the office section so was asked Terry to give a sample of the dataset so afterwards new values came into perspective.

See below as an example
:

First Name	Last Name	Employee ID	Payroll ID	Office	Department
Gerry	James	34		Ireland - Staff	Finance
Aindrias	Corcoran			Germany - Contractors	Engineering
Charles	Boyle			Germany - Contractors	Engineering
Dara	Fay			Ireland - Contractors	Engineering
Herbert	Boczian			Ireland - Contractors	Engineering
Kate	Kelly			Germany - Contractors	Engineering
Wiliam	Kelly			Germany - Contractors	Engineering
Niall	Keegan			Ireland - Contractors	Engineering
Alfie	Chalk			UK - Staff	Engineering
Morven	Gannon			Ireland - Contractors	Engineering
Leonat	Xhokli			Germany - Contractors	Engineering
Elva	O'Conaire			Ireland - Staff	Tech Transfer
Gerry	James	34		Ireland - Staff	Finance
Orla	Barber	45		Germany - Staff	Engineering
Terence	McCartan (MP)			Germany - Contractors	Engineering
Breda	Conophy	83		Ireland - Staff	Recruitment
Brian	O Connell	07		Ireland - Staff	Engineering
Daniel	Devlin	92		Ireland - Staff	Engineering
Danny	O Driscoll	41		Ireland - Staff	Recruitment
David	McCurtin (MP)	93		Ireland - Staff	Engineering
Declan	O Riordan	70		Germany - Staff	Engineering
Eoin	Reilly (MP)	48		Germany - Staff	Engineering
Fathin	Jafri	82		Ireland - Staff	Engineering
Karolina	Gorzewska	74		Ireland - Staff	Recruitment
Kevin	Foley	59		Germany - Staff	Engineering
Leah	Dillon (MP)	97		Germany - Staff	Engineering
Louis	Tate	98		Germany - Staff	Engineering
Nathan	Cairns	72		Germany - Staff	Engineering
Norma	Nagle	81		Germany - Staff	Engineering
Orla	Redmond			Germany - Staff	Engineering

Points to be noted:
1. The office has two broader categories: contractors and employees. The employees have an employee number whereas the contractors do not.

2. In the provided example dataset there is a section for payroll id but there is no value provided in the dataset.
3. The data source in reference to office and payroll id is not clear yet but it will be coming from another system which will be connected to the timesheet system.

Office Table

ID	Office ID	Office Name
1	1	UK Staff
2	2	UK Contractor
3	3	Ireland Staff
4	4	Ireland Contractor

Department Table

ID	Department ID	Department Name
1	1	Engineering
2	2	Recruitment
3	3	Finance
4	4	Tech Transfer

Manager Table

ID	Manager ID	UserProjectRoleID	Project ID	Manager Name
1	1	1	1	Mick O'Gorman
2	2	2	2	Daniel Devlin
3	3	3	3	John Devlin
4	4	4	4	Chris Fernandes

Week Table

ID	WeekYear ID	Year	Week Number	Week Start Date	Week End Date
1	1	2023	1		
2	2	2023	2		
3	3	2024	1		
4	4	2024	2		

Timesheet Table

first name	lastname	Office ID	Department ID	Manager ID	WeekYear ID	Hours Worked	Overtime Hours

Clarification required:

1. Is the project manager the manager for everyone in the team or there are separate managers for everyone and then those managers report to the project manager.
2. How can we get the “week start date” and “week end date” populate by itself.
3. Should sectioning be implemented for the dates.

Workplan with Timelines

97	**Time Sheet Application**
98	DB + API Tech Doc (Uni. Ulster)
99	Course
100	DB + API Tech Doc (Uni. Ulster)
101	Presentation for quarterly meeting
102	Sprint workshop planning
103	Azure cloud research
104	Source Control
105	Test Strategy Plan - Natalia
106	**Sprint 1 (Timesheet entry, man/user relationship, admin portal)**
107	Workshops / Planning
108	Document workshop
109	Send any relevant info / queries to UU
110	Implementation
111	Sprint review
112	**Sprint 2 (Approval process, PM portal, migrate to cloud)**
113	Workshops / Planning
114	Document workshop
115	Send any relevant info / queries to UU
116	Implementation
117	Sprint review
118	**Sprint 3 (Extend to Nuaval Data Collection)**
119	Test Run Sprint 1 & 2 - ??
120	Workshops / Planning
121	Document workshop
122	Send any relevant info / queries to UU