Alberta Science Park

Software Design Document

Version 1.1

Introduction

Document Purpose

The purpose of this document is to outline the technical aspects of the Alberta Science Park Web Application and the technologies used to develop and implement the application. The goal of this document is to give the reader a better understanding of how the application is being developed and implemented through examples of requirements, constraints, and system architecture.

Architectural Overview

Application overview

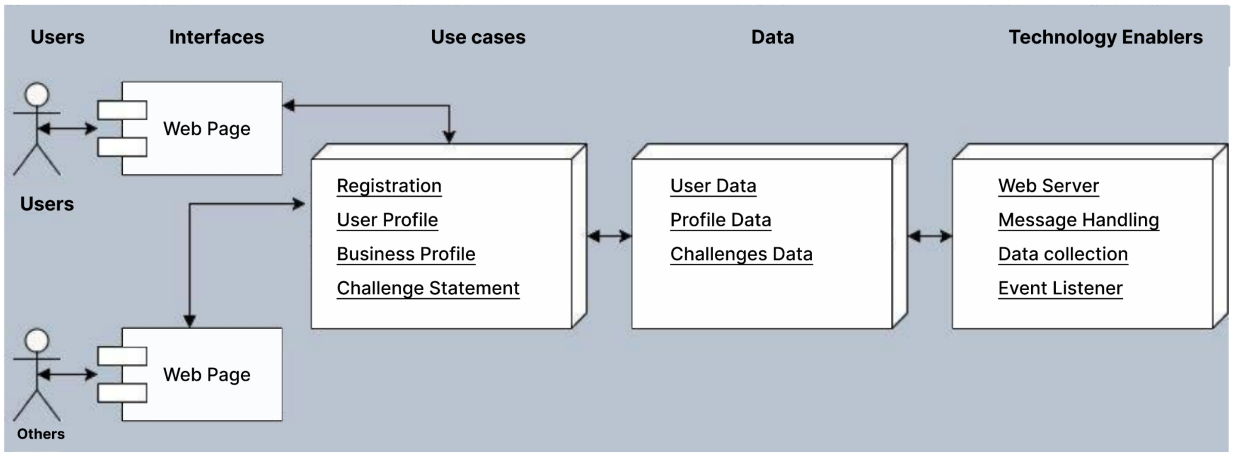

Figure 1: Application Overview and Diagram

Core application uses

Figure 1 highlights the use-cases for the Alberta Science Park application. These uses are the expectations of information to be presented to the user.

- Persona One - Geroge
    - His purpose for using the platform is to seek solutions to his particular engineering challenges which affect pipeline operations
- Persona Two - Graham
    - His purpose for using the platform is to ensure utilization of all the lab's resources, to help secure ongoing funding and attract a steady stream of research applicants to the lab.
- Persona Three - Nancy
    - Her purpose for using the platform is to be more efficient and effective in her sales process, by utilizing the matchmaking process between solution seeker and solution provider, by ensuring that the lab's CV is always up to date in the platform.

- Persona Four - Brenda
    - Her purpose for using the platform, as a platform owner, will be to ensure that the funds were well allocated and spent, and that the utility of the platform is not skewed toward one user group, but offers equally useful functionality to all users and user groups

Data

The core information and data constructs required to realize the core application uses are highlighted in Figure 1. The follow data is fundamental to the core use cases.

- Users
    - The current user data will be the current/potential users using the Alberta IoT application. This system will manage the propagation of user data from web services and local hardware.
- Personal Information
    - Personal Information data will be mapped against the particular users collecting their personal data viz. First name, Last name, Education, Position, Languages spoken, Experience Level, Address, Location etc.
- Business Information
    - The Business Information data will be stored as a business profile for each and every user. The data stored will be like Company Name, Website, Description, Office Phone, Company Classification, Company location etc.
- Challenge Statement
    - The challenge statement data will allow for permission based action between the web application as well as the web service. There will be a user to access the specific challenges he/she has created and also a common list of data to checkout all the challenges going on the platform.

Technology Enablers

Figure 1 highlights the key set of components to support implementation of the Alberta Science Park application.

- Web Server
    - The application will reside on a web server. The web server will be required for providing access to the Alberta Science Park IoT application interface and the REST api.
- Data Collection
    - The data collection will be implemented on a gateway machine that will propagate user and challenges data from the portal. This module will also be responsible for sending data to the web database through the REST api.
- Event Listeners
    - This module is the main function of the REST api. It will handle incoming events and request made over http from clients and web service users. It will then hand off the messages to the message handler when implemented on the portal in near future.

The Alberta Science park IoT application is built predominantly upon the Django framework. The Django framework uses a 3-tier architecture very similar to the popular Model, View, Controller (MVC) architecture. Figure 2 can be seen below and is a diagram of the implemented framework in respect to the web application. In Figure 2 below you can see that a Django project can have multiple applications within itself, and that the Alberta Science park IoT System is using two applications in tandem.

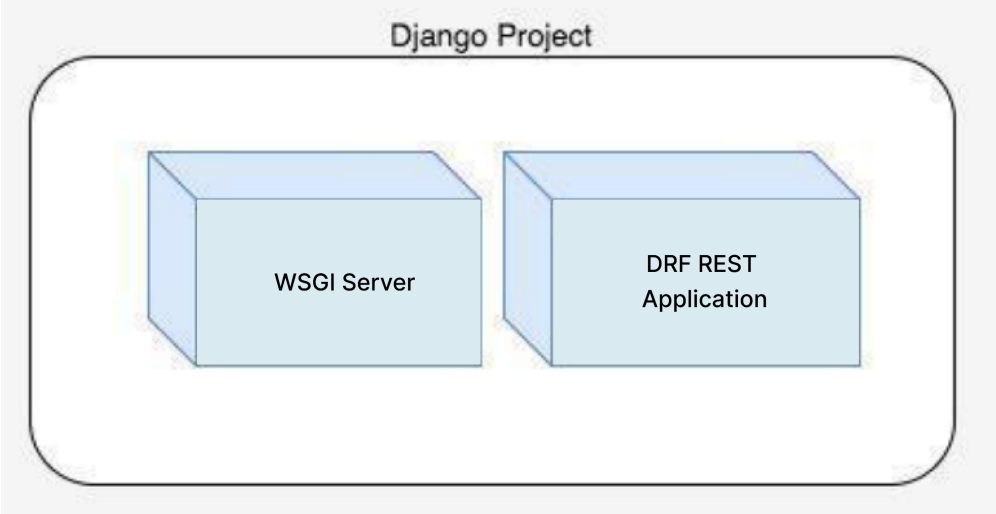

Figure 2: Web Application Layers Diagram

In Figure 2 above you see that there are two applications within the single Django Project. These two applications work together to accomplish the goals of the Alberta Science park IoT System. The Alberta Science park IoT Application is the application that is built upon the standard Django framework and is modeled after an MVC architecture.

Figure 3 below is a diagram of the Rest application layers. In the diagram below you can see that there are three main layers: Models, Views, and Controllers. Inside the Models you can see that there are the actual data objects that our application uses to store and display, as well as the REST Serializers that are able to serialize our data into JSON and make it easily consumable by the REST api. The Controller layer handles the GET/POST requests made by users and our application to send and receive data. The View layer is the piece of the application that allows our webpage to be created and displayed with data from the models.

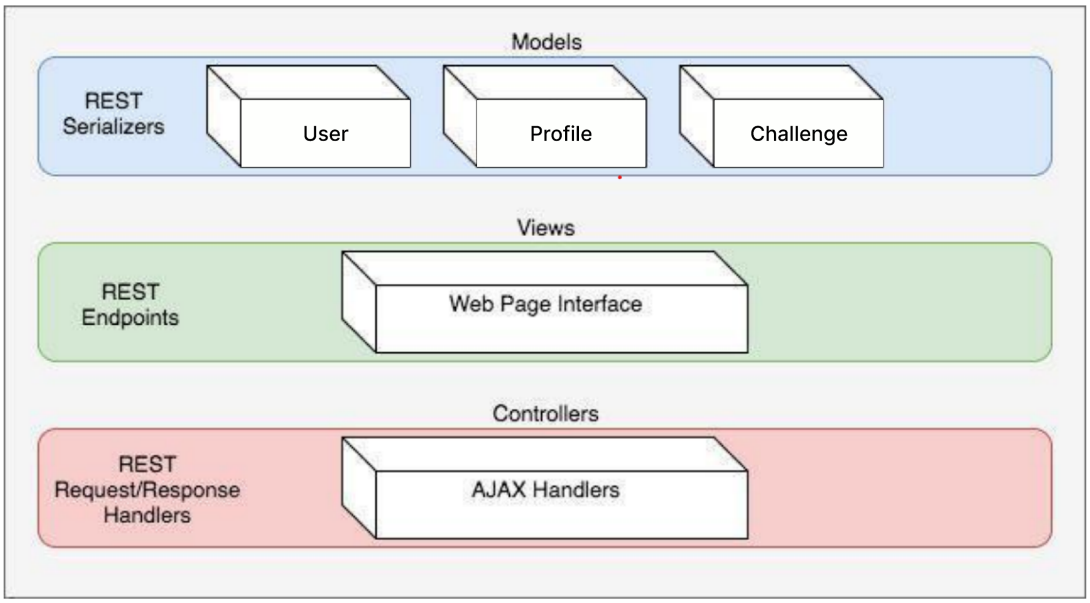

Figure 3: Layered View of Alberta Science Park Architecture

The Alberta Science park IoT application will use a fusion of two architectures Model, View, Controller architecture, and RESTful architecture. Inside our Django project you may recall that we have created two separate applications. The MVC will be used in the implementation of the web application interface and the RESTful architecture will be used for the backend, which will handle http request to api endpoints including ajax request from the REACT web application and external request from data collecting gateway machines. An overview of how the RESTful architecture works with our standard Django MVC architecture can be seen below in Figure 4.

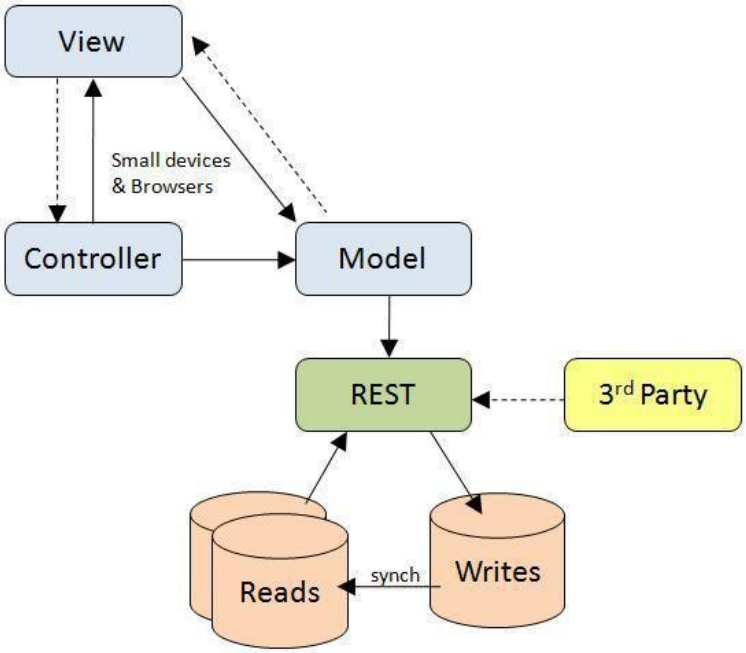

Figure 4: REST & MVC Architecture Diagram

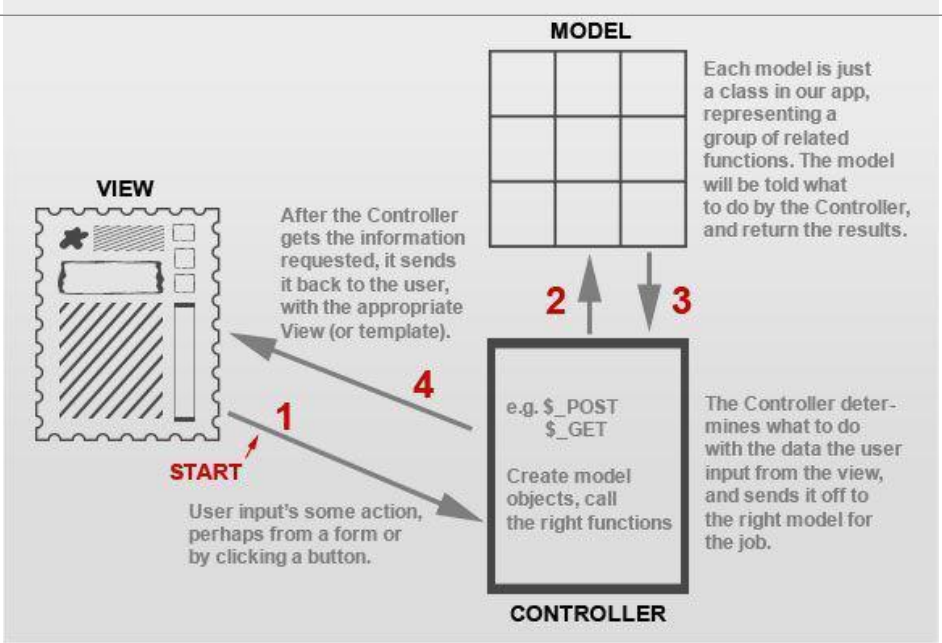

Figure 5: MVC Diagram

Module and Interface Description

Django

MVC framework vs Django MTV framework.

Our project will be using django at its core. This allows us to set up the website using a commonly used MVC framework. The MVC framework consists of 3 main components; Model, View, and Controller. The model portion of the framework consists of all of the classes that we will need for the project. The View is basically what will appear on the webpage. The controller is what links the Model and the View together.

In Django the underlying MVC architecture is actually slightly different from the classic MVC approach. Models are still Models in Django, but a View is actually called a Template, and a Controller is actually called a View. This means that a Django Template is actually what you see on the webpage, and a View links the classes in a Model together with a React Frontend Technolgy.

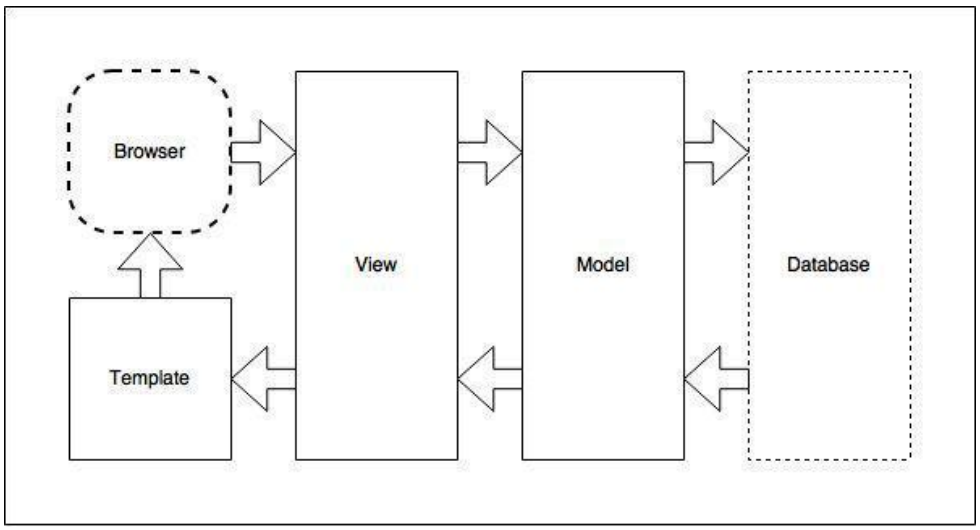

Figure 6.1: Django MTV Diagram

Flow of data in a Django MTV framework.

Figure 6 shown above showcases a brief overview of the flow of data in a basic Django application, beginning with a request from a browser and resulting in a web page produced back to the browser.

When a request is made to view a webpage provided by a Django application, it is first referenced in a list of url patterns located in a file “url.py”. The url patterns in this file will link directly to the View portion of the MTV framework by accessing a file called “views.py”

The file “view.py” basically holds all of the functionality for the Django application (which explains why we reference these as “controllers”), and uses the classes defined in your Model to manipulate the data before sending the data to a template.

The Model keeps all of the models in a file labeled “models.py”. Once a class is defined in this file, any objects created from each class will automatically be added to an SQLite database that is maintained inside the Django app. The requested data from the database will then be returned back to the View, and then returned to template.

Templates are used to dictate how the processed data will look on a webpage after it has been requested. A template consists of all of the basic utilities that can be included in any html document. Each page in a Django project will require its own template.

Urls.py

The url patterns in the “urls.py” file include “index”, “register”, “login”, and “create/personal-profile/”. Each url pattern sends a request to the View which calls a function by the same name in the “views.py” file. For example, the url pattern for “index” uses a line of code called “views.index”, and will call the function “index”, from the file “views.py”. This means that “views.py” will only consist of the 4 functions listed above. This “urls.py” file handles the url routing for the django application.

Models.py

In Django, database tables are created via python classes. These classes are individually referenced as a “model” and all together we call our database entries the “Models”. In the Alberta Science park IoT System we currently have many models of which let’s consider these three : Challenge Statement Data, Profile(User/Business) Data, and the User Data.

Views.py

“Views.py” will consist of 4 functions; “index”, “UserRegistrationView”, “UserLoginView”, and “PersonalProfileCreateView”. These functions return data to specific web pages within our web application. Each of the functions in the View are referenced by the “urls.py” file, and are called after being requested by the corresponding URL address in the Django application.

The View is not responsible for how the data is displayed, but rather what data will be displayed. The View is the section in which all of the functionality for the web application will reside. It is responsible for requesting queries from the database,

and then manipulating and organizing the data before passing it off to a React -Frontend for displaying to a bowser.

Implementation Plan

The following section outlines the steps and milestones that need to be completed so that the Alberta Science Park System can be implemented on time. Figure 7.1 below depicts a rough project timeline and includes 10 major implementation milestones.

Conclusion

As a team, we feel that this project has great potential to very useful to our client. It will enable them to more effectively use their application to match-make possible personas and learn about the interests, and thus making a large impact in their productivity.