**Language Justification**

Given our group’s knowledge of Python, it was decided to utilise this along with the flask framework to design our treasure hunt. Given the time frame of the project it was more sensible to build on our knowledge of Python rather than trying to pick up a new language such as PHP. Flask and Django were evaluated as potential options along with Python but Django is drastically more complex than Flask and given the complexity and scalability of the app, this extra complexity would not be required and therefore add overhead to the program that was not needed. Flask on the other hand is a lightweight python framework that required little config to deploy and thus ideal for our needs.

Within flask, there is not set framework of structuring your code so a Model, View, Controller framework was adopted to enforce coding standards and the structure of the code. Flask has no inbuilt library for this so it was decided to build a basic MVC framework within the app to achieve this, the use of Python objects is used to achieve this allowing you to create each model, view and controller to each be their own class and thus instantiated when they are needed.

The model is used for all communication with the database and will stop repetition of query common queries from the database as well as common actions that may take place across many different pages. The view is used to show data to the user, Flask uses jinja templates to render HTML to the user, the ease of which a HTML page can be rendered and the option to pass in data generated from Python would mean it is ideal for deploying new pages quickly which is needed in a Kanban environment. The controller is used to control the data flow within the application and show different views to the user depending on what they need to see.

The use of MVC will ensure that each member of the team will keep to a loose structure when creating their code and it will be a lot easier for others to modify and understand it as each page will use the same structure and all database operations will be done in the same place. MVC is ideal for parallel programming and therefore idea for Kanban development as it allows different programmers to be working on the model whilst another works on the controller and won’t cause any conflicts within the code. Especially within multiple programmers working on the same database system, changes to the database will not affect changes to the view etc until the data returned by the model is updated thus ensuring that all changes are contained and deployed in a structured manner.

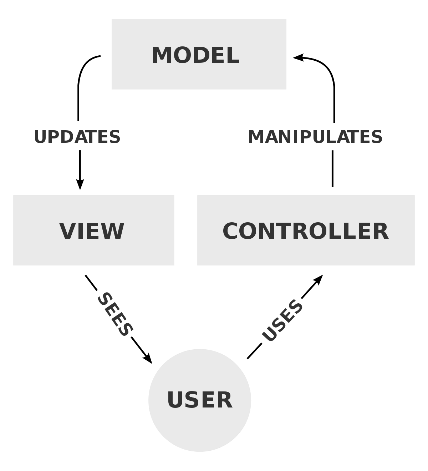


Figure – MVC control flow diagram

A standards.md file was created to give some loose guidelines for the code that was to be produced would adhere to our created standards such as naming conventions. As a group we decided to use common industry standards such as camel case and the use of doc strings to define functions thus making it clear for other members of the group what each code does. When each part of the program is done, it will be commented fully for the use of other programmers.

To ensure the best user experience for our users, early group meetings established the need to use Asynchronous JavaScript And XML (AJAX) within the project, this makes the users feel like they are using native app as the page does not need to constantly need to reload to give users updated data. The use of the controllers within our project allows us to easily map URL endpoints to python functions and therefore handle requests made to our server over AJAX.

In order to provide continuity within the project, we chose to use JSON for passing all data between models, controllers and views – this has the advantage in Python of being similar to a dictionary and therefore little to none code is needed to convert between the two. JSON was chosen over XML given the support for both within Javascript and its wide use within industry as the standard for responses from APIs.

To reduce code usage, a common directory of files such as CSS and JS will be created to ensure that no one was reinventing the wheel when it comes to writing common or shared parts of the web app – this wil reduce the development time as well as reducing the amount of bugs as code does not need to be rewritten by multiple people.

Within our web app, a database is needed to store the data such as the teams that are participating and the questions that they will need to answer along the way. Looking at our group’s experience, we all had experience with SQL based databases and therefore it made sense to utilise these. Researching Python’s support for databases, it was decided to use an SQLite database within the project to store the data, connections to this and the operations on it will be handled within the models within our Python code.

To facilitate multiple developers working on the code at once, Github will be used as version control for the project. Each developer will use their own branch (unless working with someone else on code) to do their work in and test within, this isolates it from other users changes until they want to include it and reduces the risk of other code interrupting developers work. Once each section is done, the code and relevant user stories will be validated by atleast one other developer to check that it meets the standards and the requirements for the project and if successful, a pull request will be opened to add it to the master branch and any merge conflicts resolved. The validation process ensures that rogue code is not put into the master and that the master always contains working code from which the program can be deployed; it also ensures that all user stories are met and that both developers have interpreted the story in the same way and thus reduces cases where the stories are not fully met.