1 Introduction

The whole project will be divided into 3 parts:

- Front end, including all the pages and client logic functions.
- Back end, including all the server logic functions.
- Marker.

Front end languages we will use are HTML, CSS and Javascript. The first two are pretty easy to learn because actually these two cannot be considered as **programming languages**. The main problem here is Javascript. This is mainly because it is too flexible to use especially when used in a collaborative project. It will become harder to debug while the project is getting larger. However at the beginning writing Javascript won't be a problem for you given all of you can write Python very well. I will talk about the details of writing Javascript in **How To Do** section.

Back end are the easiest part in our project because we don't need to providing many functions for users. As we chose Django as the web framework, we don't need to take much care about database. Another issue in back end is configuring all the applications running on server. This requires shell programming skill and quick-learning.

Marker is the core of this project. We need to overcome several problems to fully implement a good marker:

- Code style checker (Partially Done)
- Test case checker (Partially Done)
- Sandbox
- Task queue
- Communication

The first two have been partially implemented but haven't been fully tested yet. Sandbox and Task queue can secure our server. We need to decide how to communicate with back end in communication module.

2 What To Learn

2.1 Languages

- HTML. (click to learn)
- CSS. (click to learn)
- Javascript. (click to learn)
- Python.
- Shell. (at least can understand and modify others' codes)
- SQL. (at least can understand others's codes)

2.2 Frameworks

In front end, we will use Bootstrap as CSS framework and JQuery as the tool to get elements in HTML and communicating with back end using Ajax. (click to learn Bootstrap, click to learn JQuery)

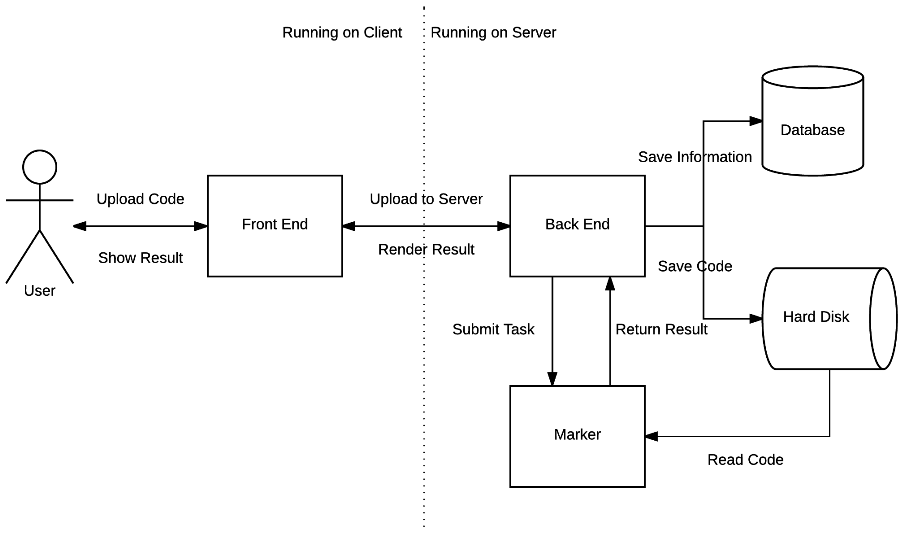
In back end, we will use Django. Django provides URL routing, template and database model functions for us. I will provide a barebone project for you to fast start to code with Django.

2.3 Tools and Applications

- MySQL. We will use MySql as database on server.
- Nginx. We will use Nginx as web server.
- GitLab. This will be used as our private git repo and issue tracker.
- Git. Used as our version control system and collaboration system.
- IDE or editor. I suggest to choose Sublime Text or PyCharm. You can use PyCharm for free using your student email.

3 How To Do

In order to give you a brief understanding of how the project works, I wrote a figure of how data flows between different parts.



After user submits his code, front end will upload the code to back end. And then back end saves the information of this submission into database and the code into hard disk. After saving, back end will submit a new code marking task to marker and then waiting for the code checking report. Once checking report is received, back end will render the result and send it back to front end.

So we can simply divide the whole project into these 3 parts and assign these sub-projects to each of you. I listed the details of tasks in each sub-projects here.

3.1 Tasks

- List user cases. Write a document describing fully details of how user will use our project.
- Front end
  - Design pages

    - Index Page, including several information of us and links to courses.
    - Course Page
    - Assignment Page
    - Code Submit Page
    - Result Page

  Some of these pages may be merged into a single page and use Ajax to communicate with back end. First try to list all the functions provided for users on each page based on user case document and then list all the elements on each page. Then you can either draw prototypes or implement pages in a simple way. After this step finishes, we will hold a meeting to talk about which part should be improved.
  - Implement pages. Once all of these have been done, implement pages using Django Template module, Bootstrap and JQuery. While implementing, write the protocol document to let back end know what should be given during rendering templates and Ajax.
  - Test pages. Because our project now is simple enough, we can test the pages by hand. We will try to introduce some web testing framework after I come back.

- Back end
  - Install and configure all the applications running on server.
  - Design modules
    - Design all the view functions and models based on user case document.
    - Finish protocol document with front end.
  - Implement modules.
  - Test modules
    - Unit testing should be finished during implementing. We can just use Django test module to test our functions.
    - Integration testing will be done after complete implementing modules.

- Marker
  - Read my code and test my code.
  - Implement a task queue and communicating API.
  - Implement API for writing test cases of assignments in future.

3.2 Code Conventions

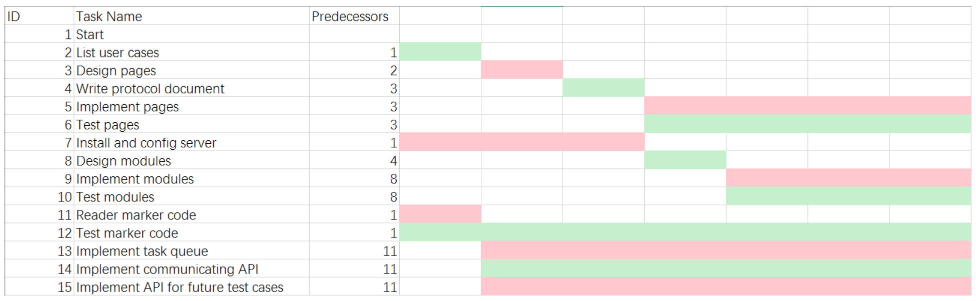
- Python code convention
- Javascript code convention

3.3 Workflow

Read all the articles under Collaborating. This workflow will be done in your daily developing.

After a new feature has been submitted into master branch, someone should check master branch out on server.

I wrote a gantt diagram to show the sequence of tasks.



ID	Task Name	Predecessors							
1	Start								
2	List user cases	1	🟩						
3	Design pages	2		🟥					
4	Write protocol document	3			🟩				
5	Implement pages	3				🟥	🟥	🟥	🟥
6	Test pages	3				🟩	🟩	🟩	🟩
7	Install and config server	1	🟥	🟥	🟥				
8	Design modules	4				🟩			
9	Implement modules	8					🟥	🟥	🟥
10	Test modules	8					🟩	🟩	🟩
11	Reader marker code	1	🟥						
12	Test marker code	1	🟩	🟩	🟩	🟩	🟩	🟩	🟩
13	Implement task queue	11		🟥	🟥	🟥	🟥	🟥	🟥
14	Implement communicating API	11		🟩	🟩	🟩	🟩	🟩	🟩
15	Implement API for future test cases	11		🟥	🟥	🟥	🟥	🟥	🟥