**System requirements**

**Front-end (web-server application)**

Tasks:

1. Deploy any web server, like WordPress or any other (if somebody is familiar with it).
2. Create a free domain, like [www.ifiretracker.io](http://www.ifiretracker.io), and tune the webserver to have access. Hint: possibly, some services like Railways have already provided their domains to deploy our domain for free. It should be like [www.ifiretracker.heroku.com](http://www.ifiretracker.herolu.com)
3. Create 3 pages (Authorization, Upload files, Display results) and deploy them on the server.
4. While creating pages, it’s necessary to create API requests and responses to the Back-end to process the data.
5. If somebody is familiar with Flask (it's a Python library for web development), perhaps it should be the best solution to communicate with the Back-end server.
6. Think about delays with processing. What should we display on the web page while we are waiting for the results, uploading huge files, etc?
7. Think about how to display effectively the map results, graph results, etc. Use Python libraries to display the results.
8. **Describe all types of interactions with the front end in terms of functions. Like, authorization request: user\_id, pass. Authorization response: OK/NOT\_OK/ERROR\_404.**

**Back-end (Authorization Server, Application Server, Database Server) (total 4 team members)**

Possibly it should be divided physically into 2-3 parts according to the recommendation from Guilio today, like an Authorization Server, Application Server, and Database Server. Or we can combine Authorization and Database servers into one physical (on one hardware). But anyway, as a software, it should be three independent servers.

In the software part, I suppose we should use Python libraries to integrate the parts of our service. It’s simpler if we will use Python for developing data science models on the server.

**Authorization server**

Tasks:

1. Find best practices on how to organize this functionality on the internet.
2. This server should get requests, search users in DB, and return responses to the web server. To do this use Flask (parse and pack the information from the third-party side) and API (send/get requests/responses to DB and web-server) to send and retrieve the information.
3. As additional functionality, this server can support some techniques to increase the robustness of the connection, like encrypting, two-factor authorization, etc. But keep in mind adequate steps for authorization to avoid hating our system from users.
4. Logging system. All transactions and requests should be logged in DB. For example, user login/logout - make a record in DB. And think about how to clean these tables after some time.
5. Alerts: If there is some suspicious activity, we should identify it. Some web servers have provided this functionality for web applications.
6. **Describe all types of interactions with the Authorization Server in terms of functions. Like, authorization request: user\_id, pass. Authorization response: OK/NOT\_OK/ERROR\_404.**

**Database server**

Tasks:

1. Deploy any DB using services that were provided by Heroku etc. Sometimes, it should be deployed in one click. Just need to choose what you want.
2. The DB should process any request from the Application and Authorization servers, as well as from the Web server to save the results in DB. To do this, it needs to write PLSQL/SQL scripts to process requests/responses.
3. Define a schema of the database, the relationship between the tables, what information, and which type of information could be stored in the database.
4. Create tables in DB. Fill out with test data.
5. Should we store the input data in DB? Which size of the data will we feed to the model? Based on the answers it should think about interaction with Application Server.
6. **Describe all types of interactions with DB in terms of functions. Like, authorization request: user\_id, pass. Authorization response: OK/NOT\_OK/ERROR\_404.**

**Application Server**

Tasks:

1. **Model 1 (Table data)**
2. **Model 2 (Image data)**
3. The models can receive the data, process it, and return the answer to the web server to display them.
4. The languages which are used to develop code should be Python or R. The communication between the Application Server and other parts should use Python.
5. Look for best practices to predict fires and locations on the Internet.
6. Look for the Christchurch data about fires for previous years.
7. Think about how to use the data in models, and which models to use.
8. Based on data and best practices which we can find on the internet we should define the model and its functionality clearly.
9. **Describe all types of interactions with other parts of our service in terms of functions. Like, get\_fire\_spots\_request: table\_data, file\_id, MD5. get\_fire\_spots\_response: coordinates, radius, probability.**