A description of the overall system functionality

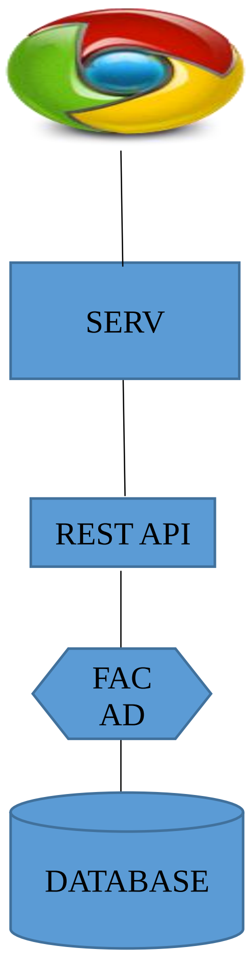
Regarding technical characteristics, our system follows the same strategy as we have implemented many times in class. A node/express server is handling client requests via an API and sends back the appropriate responces. In case the client asks for data, they are querried by the server from a mongoDB that runs on a remote loation. The page that the client sees uses Angular in order to contruct itself. This way network and server workload is reduced as the server doesnt have to construct and send a long page.

Regarding user functionality, the user can do all the required "user stories" from steps I, II and III: Search for wikis according to a search string, seeing details for a wiki, seeing all wikis by category, and seeing all categories. For the last one, we opted for a design different than the one proposed in the description. Categories are loaded at the view not all at one, but by first letter. Then the user can change letter, or filter by search string. This manipulation hapens at the model.

A description of the backend tests (explain why you think that the backend is covered by your test, or what you have to add in order to get there)

Mocha (Artur)

Back-end test approach: INTEGRATION test



We choose this approach because when we run a Façade test, we use real url. This means that we test the entire flow. We have a request from the client with a specific url, our server deals with request, Rest API talks with Façade which communicates with the TestDaB. For a small system like this one, we decided that there is no need to write tests for the interface and rest api because we know that when we run an integration test, those two components cooperate together. The downside of this is that for a larger system, it would be hard to detect which component caused the error, so separate tests would be better for that case.

A description of the frontend tests (what you decided to test and why)

Frontend Test:
The frontend tests of the program is created using the Jasmine test library for Karma test framework. The benefit of using these to make the tests instead of other libraries is do Karma being able to handle tests in programs that run asynchronous code and commands.
On the frontend part we are testing mainly two different things. If the controller and factories are defined in the views and if they are working as they should.
The Controller returns the factories to the view:

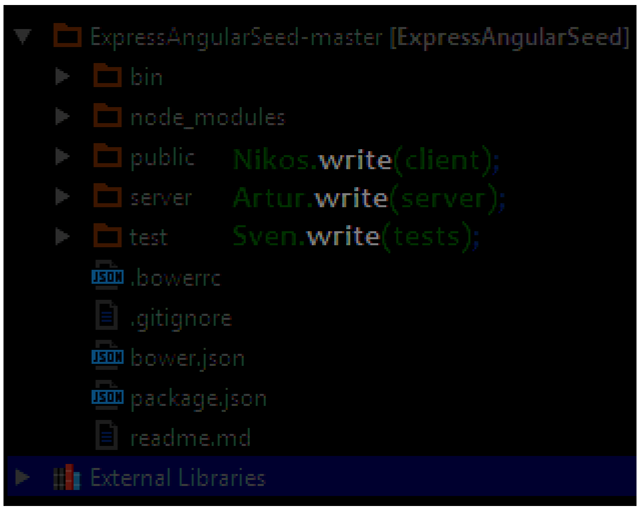
- Expect the scope to return and make WikiFactory available to use.
- expect the scope to return and make the DataFactory available to use.

The WikiFactory gives wiki objects to the view as JSON in various manners. We have four tests on the factory in total:

- The factory returns the correct category in a wiki article
- The factory returns the correct WikiObject with specific title
- The factory finds wikis with a search function
- The factory gets a list of wikis with a specific category

The factory is tested using a mock of the backend of the program written as $httpBackend.
The reason why we decided to test these things was that they are responsible for some of the core features of our program. A lot of the other types of frontend tests that we could make, like testing if the views are returned etc, are partly integrated in parts of our backend testing.

A section stating who did what



```
Nikos.write(client);
Artur.write(server);
Sven.write(tests);
```