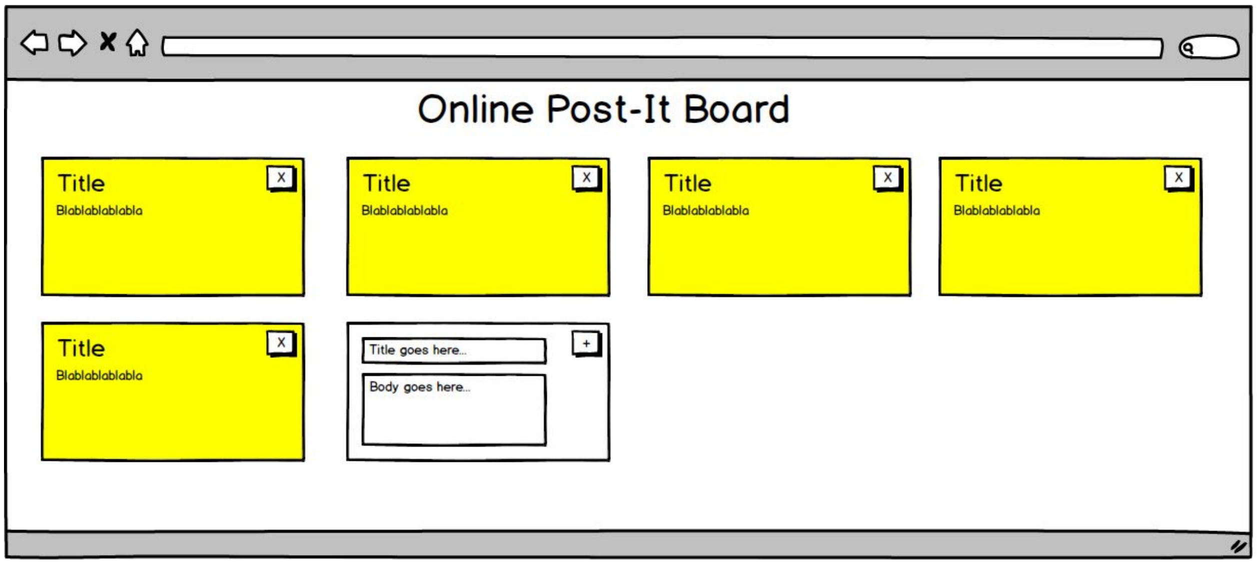
Aptitude Test #24

Introduction

In this test you will write a web application called "OnlinePost-ItBoard". The concept is simple: a web application where users can create, edit and delete Post-Its.

A mockup of a possible UI follows:



Functional Requirements

Basic Requirements

A basic implementation must allow the user to access a web page where he can create Post-Its defining a title and body, edit the title and body of any Post-It and delete any Post-It.

The title must have between 0 and 120 characters. The body must have between 1 and 1000 characters.

Post-Its must be visually arranged in a grid layout like real Post-Its would be on a board (see the mockup above).

In this basic version, the application has no backend and the post-it data is only stored locally in local storage (or a similar browser API).

Nice to Have Requirements

Below are listed several nice to have requirements, that can be implemented if time allows it.

- Markdown support: the body of the Post-It can be written in markdown which will be rendered when editing is finished.
- Tagged Post-Its: each Post-It can have several textual tags that are defined by the user. A tag can be any not empty string of text with less than 40 characters. It should be possible to filter all the Post-Its by tag.
- Server backend: implement a server backend that stores the post-its in a data base. It is not necessary to have the concept of individual user.
- Reactive: the web application must keep itself up to date, that is, if two users are connected and one of them adds a Post-It the other user must see the change happen without refreshing the page.

Technical Requirements

The static version can be served by any static web server. If the backend nice-to-have requirements are implemented, that must be done using Node.js.

The web application must work as a single page app, meaning that the web server only serves static content (and exposes a REST api, if the backend nice-to-have requirements are implemented).

The front-end code can use ES6 features (but not experimental features) and should be organized using a module system. React or Backbone.js should be used to implement the view layer.

Other than the restrictions above any library can be used.

Whenever a build system (e.g., grunt) is in place or not is **irrelevant** for this test.

All the code should be as polished and high quality as possible. It should be documented (at least with jsdoc) and have tests.

Delivery

It is important for us to interact with you (even to discuss approaches, problems, decisions, etc) so please feel free to ask any questions to the persons assigned to accompany you during the length of the test.

At the end of the allotted time for the aptitude test all the artifacts produced (code, git repo if available, documentation, etc) should be send or made available to Feedzai.

After delivery a meeting will be scheduled where you will do a small presentation with a live demo, a code walkthrough and any other topic you find relevant.

Finally, we kindly ask you **not to publish the artifacts produced in this test in any public repo** so not to influence future candidates.

Good luck!