Contents

1. Introduction

1.1 Overview

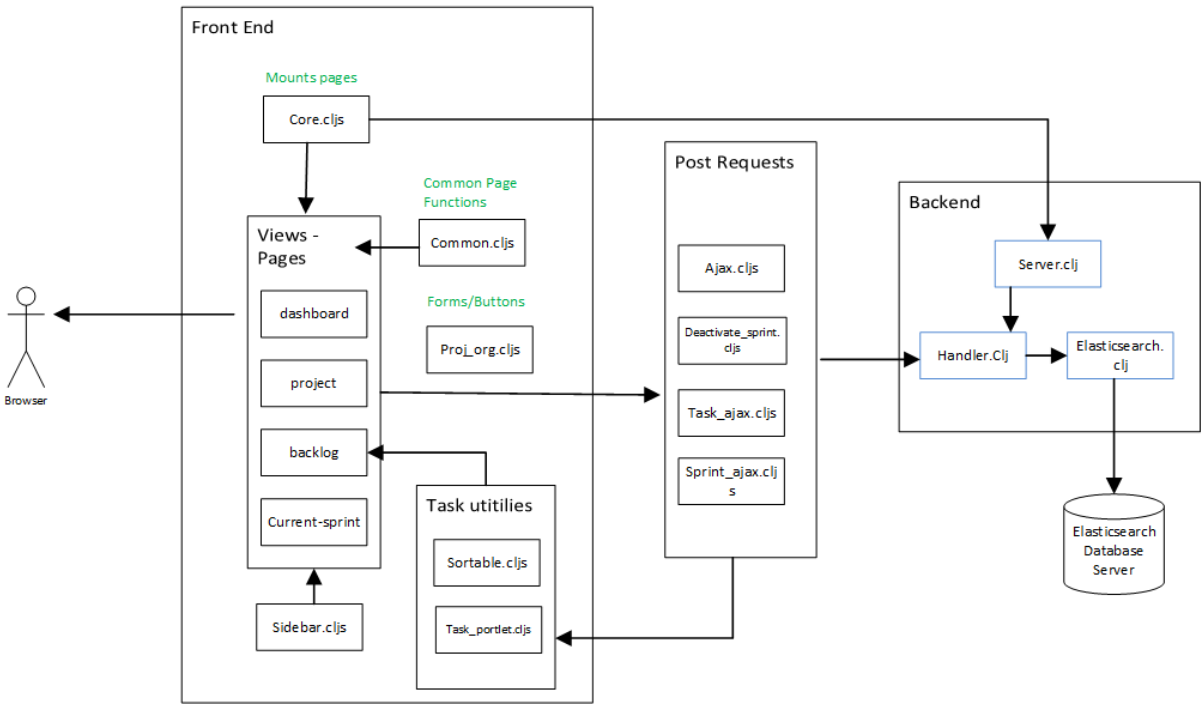
Agile Task Tracker is a project management web app designed with students in mind, to help them implement the agile principles they will be taught throughout their degree in a practical sense. It is our hope as well that it will aid them for their 3rd and 4th year projects. The front end is written in Clojurescript, while the backend server is written in pure Clojure, a functional language that built upon the java virtual machine. The server interfaces with an Elasticsearch database through a Clojure client called Elastisch. Ring is used to wrap these requests so they can be sent on to the database.

1.2 Glossary

AJAX - Is used to allow a client to communicate with a server and/or database.
Compojure - A clojure routing library
Elasticsearch - This is the database we will be using.
Elastisch - This is a clojure client for Elasticsearch

2. System Architecture

System Architecture



Here you can see the architecture of our system. The frontend is all written in Clojurescript.

The Core.cljs is responsible for defining the routes of all the pages that are to be rendered on screen. When the project is compiled, all of the Clojurescript is compiled down to javascript, which is then mounted onto the Server.clj on the backend.

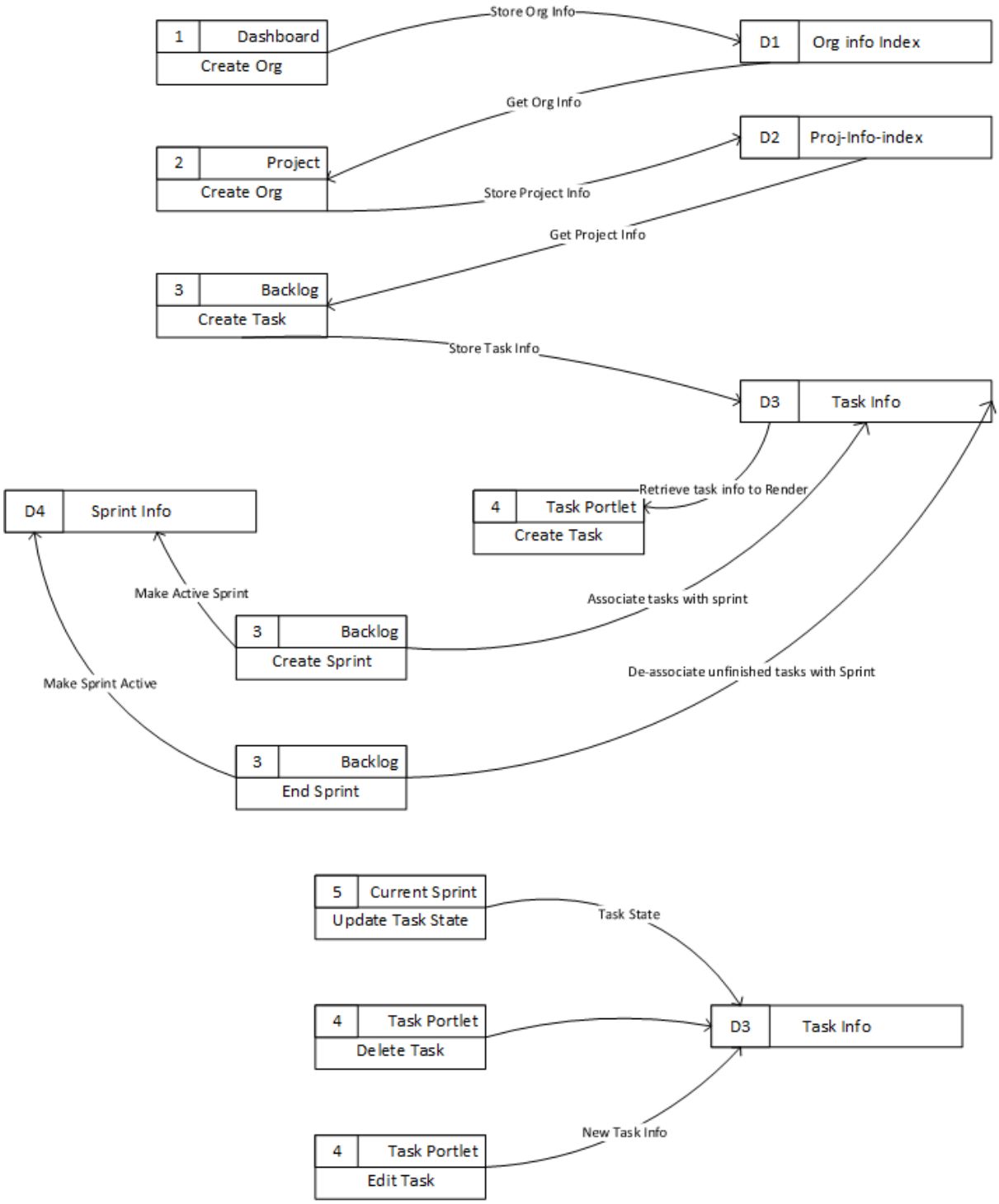
In the section labelled View Pages you can see the pages that the user will be interacting with.

- The Dashboard page is the first page the user will land on. Here they can make a new Organisation or enter an existing one. In terms of DCU an Organisation would represent a module, such as CA326
- On the project page, they can see all the projects that are associated with that Organisation.
- On the backlog page users can see all tasks associated with a project. They can also create tasks on this page.
- The tasks are created logically first on the frontend without being rendered. We post this information through the task-ajax.cljs component to the handler.clj, which will determine which Elasticsearch function will be called to communicate with the database.

- If the task was created successfully on the database, then as part of the response from the database it will return the task id. We will use this id to now query the database for all the information of this task. When we get a response with the requested information, then the handler for that ajax query will use the namespaces in the section labelled to render the task on the backlog page.
- The tasks will be rendered in sortable columns, so that they can be dragged from one column to another. This mechanism is how we let the user update the status of the task.
- On the backlog page the users can move tasks from the backlog onto a sprint creation column. Once the user enters the required information in the sprint creation form and then hit save, every task will now be associated with the newly created sprint, and the task status will be updated to “to-do”.
- The current sprint page lets you see all the tasks for the current sprint. The user will be faced with 3 columns, To Do, In Progress and Completed. The users can update the status of the tasks by dragging them from column to column, and they can also edit information about the task such as the hours logged on the task and the description.

High Level Design

Agile Task Tracker
DFD



4. Problems and Resolution

Setting up communication with Elasticsearch
One of our early roadblocks with the project was trying to figure out how to communicate with our database, Elasticsearch. It took us a while to realise the correct workflow of

- Making an AJAX POST call on the cljs frontend
- This POST is then wrapped in some Ring Middleware, which can convert Clojure data structures to JSON
- Then this is passed on to our Handler.clj. This uses a routing library called Compojure which lets us determine where from the frontend the request has come from, and based off parameters we have passed along with the request, decide what method to call from our Elasticsearch.clj
- This Elasticsearch.clj uses a Clojure client for Elasticsearch called Elastisch. It is here that we can define the query we will make to the database.
- The response of which is passed back along the route to either a success or error handler. These handlers then determine what we do with the information we have received from the database.

Creating task portlets dynamically
Initially when designing the backlog page, we implemented a jquery sortable function to allow us to drag and drop tasks in a predefined column. However, this was only for static elements of a page and required us to map the id of the task to the associated portlet and call directly by id the toggleable state.

Working in a functional language
We had never undertaken a project to this scale before and this was our first time using Clojure, Clojurescript and Elasticsearch. As both myself and Bernard are interning at Cellusys where we will be coding using the above, we decided to code this web app in this functional language. It caused some steep learning curves but we are happy with the demo we are submitting as it has the base functionality of an Agile dev tracker. One particularly difficult section was to try and model our system as UML and other OO orientated modelling methodologies do not lend themselves well to functional languages.

We decided to model our system instead using a System Architecture model and a DFD to describe how information is based between functions in our CLJ and CLJS namespaces.

5. Installation Guide

For Installing on linux distro.
Java JDK (at least version 7)
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Make sure you have the java JDK installed.

Leiningen
https://github.com/technomancy/leiningen/blob/stable/README.md

Download lein script from stable branch
Move to ~/bin
Make it executable “chmod 755 ~/bin/lein”
“./lein” to install

Elasticsearch (v1.7.6 Do not update)
https://www.elastic.co/downloads/past-releases/elasticsearch-1-7-6

Download and save the DEB package
Navigate to download directory
“sudo apt install gdebi-core”
“sudo gdebi elasticsearch-1.7.6.deb” to install

“sudo nano /etc/elasticsearch/elasticsearch.yml”
Delete the # by cluster.name
Change it to (without quotes) cluster.name: agile-task-tracker
Similarly for node.name (this time with quotes):
Change it to node.name: “agile-task-tracker-node-1”
Save and exit nano

To start/stop/check the status of elasticsearch

- First check if your system uses init or sytemd with:
  - ps -p 1

- If it uses sytemd
  - sudo systemctl start elasticsearch.service
  - sudo systemctl stop elasticsearch.service
  - sudo systemctl status elasticsearch.service

- If it uses init:
  - sudo -i service elasticsearch start
  - sudo -i service elasticsearch stop
  - sudo -i service elasticsearch status

Install plugin to view elasticsearch database on a web browser:

- sudo /usr/share/elasticsearch/bin/plugin -install mobz/elasticsearch-head/1.7.6
- visit the page at: http://localhost:9200/_plugin/head/

Download Agile-Task-Tracker

"git clone https://gitlab.computing.dcu.ie/kinses38/2017-CA326-kinses38-agiletasktracker.git"
Navigate to "code" directory

If running in dev mode

"lein figwheel"
Wait for compile to finish
Open web-browser and enter "localhost:3449" when terminal shows
"Prompt will show when Figwheel connects to your application"

If running in prod mode

"lein clean, do ubjerar"
Navigate to "output"
"java -jar agile-task-tracker.jar"
Open web-browser, "localhost:3000"

To run tests

Follow instructions at https://gist.github.com/julionc/7476620
Navigate to code directory
Enter "lein doo phantom test once"

Hardware Recommendations

We recommend running this demo on linux, with a minimum of 4gb and a dual-core processor, harddrive space usage is minimal as compiled it only requires 11mb when deployed.