

# Contents

Contents 2

Design Brief 3

Overview/Problem Statement 3

Goals and Objectives 4

Product Description 4

User Characteristics 5

Social and Ethical Issues 5

Ease of use of the final solution 5

Inclusivity 5

Accessibility of technical language 6

Copyright 7

Ergonomics 7

Planning a Software Solution 8

Programming Languages 8

CASE Tools 8

Development Approach 8

Relevant Algorithms (Pseudocode) 10

Data Dictionary 13

IPO Chart 14

Storyboard / Concept Design 15

Logbook 17

Term 1 17

Bibliography 20

# Design Brief

## Overview/Problem Statement

This document provides information on the requirements for the project-management software application. Project goals and definitions are given in the introduction. Planning for the software development is described in the following section, with building, checking and modifying the software solution outlined later.

At this moment, project managers and team-workers are relying on unrecorded communication to facilitate their progress in projects, that ultimately determine the quality of their finished product. This facilitation, however, may become convoluted if not done clearly and concisely, and it requires much effort to delegate tasks to other people repeatedly. To accommodate for this problem, a piece of software may be written to allow team members to receive and delegate tasks that accumulate to the completion of a project, so that they may have a better progress flow. Through this, the users should be able to improve their productivity by allocating effort to tasks that have been delegated to them by a managing user for a more visual outline of a project’s progress.

The DeleGator application will be a web-based app available to users with access to the internet. The application will provide access to a user’s account, each being associated with a unique username and password combination. The app will allow team-members to check their given tasks or “stories”, such that they can visually maintain their organisation over the long term.

Because of the nature of our modern project management qualities, this app will follow the trend of integration into digital/web-based application development.

Requirements for this solution are that the user can easily track delegated tasks and uphold the organisation of their project’s development. Objectives for this software should be that it can do what is required of it in a visual manner for ease, with all its operations acting within the boundaries of needing a secure internet connection to access the database from multiple devices.

## Goals and Objectives

The main objective of this project is to allow clients to assign tasks to fellow team-members to encourage collaboration. The DeleGator application is expected to:

1. Provide a web-based interface with “DeleGator.xyz” (or some other domain) to access account information.
2. Function in a simple and intuitive manner.
3. Provide users with a visual history of their tasks and goals as they achieve them.

## Product Description

**DeleGator** – the product that is being described here; the software system specified in this document.

**Project** – activities that will lead to the production of the DeleGator application.

**Client/User** – the person or organisation for which the DeleGator application is being built; people in managerial occupations.

**Developer** – the person or organisation developing the system.

**Stakeholder** – anyone with an interest in the project and its outcomes. This includes clients, users, developers, and peer testers.

Now for the actual description… select project, add stories to categories, delegate stories to people, add people to project etc.

## User Characteristics

**DeleGator Users:** Team-members, workers, or managers. As project-workers are becoming increasingly dependent on their use of technology to complete a project, they will likely be proficient with web-pages and applications and hence, competent in using this designed software solution.

## Social and Ethical Issues

### Ease of use of the final solution

A software application’s usability for the consumer is heavily dependent on the app’s visual elements, its use of technical language, and sensibility of the app’s overall composition.

In bringing ease of use to the DeleGator application, it would be ideal to have a simplified graphical user interface (GUI). The user interface will be a simple, graphical, systematically organised layout which follows many of the predefined guidelines of modern web-applications. In the site, there will be logical heading buttons atop the page that, when clicked, display their respective sections of the application.

### Inclusivity

#### Gender

The progressive push for gender equality in the world has brought a meaningful change to try to minimise the disparity of product use between male, female, and other genders. To fall into accord with this, DeleGator has aimed to keep all used pronouns gender-neutral, often using “you” when referring to the user. Because of this, there will be no presented bias in terms of different treatment towards different genders and thus, will not be involved with the socio-political adversities of such.

#### Economic Background

Within a society, there may exist a great variation of members’ economic backgrounds and because of this, accommodations must be made in order to be able to include people of all wealth backgrounds. To achieve this, the software product will be offered as a free-to-use web-application accessible to anyone who owns an internet-connected device.

#### Cultural Background

People of different cultural backgrounds may interpret the interactions from a piece of software differently, thus it is paramount that the software does not target a specific cultural group, exclude a cultural group, or use the language specific to any group and/or persons.

#### Disability

If an individual has difficulty with perceiving auditory or visual input, (i.e. the user has some form of sensory impairment), then specific requirements will have to be made for the software to allow it to be usable by those who have a disability. Adjustments that will be considered in the development will be the use of a highly contrasting colour palette (predominantly white with black text) to make text and various elements clearly visible to bring ease to people with visual problems such as colour blindness. As for hearing disabilities, it is not planned to have this product generate any audio, hence accommodating for people with hearing disabilities will not be necessary.

### Accessibility of technical language

As students are the primary targeted user for the DeleGator software solution, the language used within the software will have to be understandable at most levels of education. In this, the language encompassed in the solution will be no more complex than simple phrases and statements to inform the user of certain actions and inputs, such as the “Overview, Marks, Account” subheadings, dialogue messages like “Are you sure you want to delete this permanently?” and metalanguage that includes “subjects, marks, average, and assessments.” The simplicity of this language should be comprehensible to the clear majority of users.

### Copyright

Copyright refers to the protection of one’s ideas from other people such that infringement does not take place and does not result in an unfair or unjust profit to a person or group. To avoid this issue, precautions will be taken to make this program rather general, and will not use existing algorithms that are specific to the concept. As for the branding, the only possibility of copyright infringement is on the program’s similarity to other existing software solutions that achieve the same goals of bringing ease to project management, such as Trello or Asana.

### Ergonomics

The ergonomics of a software solution is defined by its ability to bridge the internal processes of the system in the interface between the system and the user. This degree of which a user can innately understand how the interactions with the software will work will determine how ergonomic the piece of software is. It is ideal that this be optimised to allow for ease in use as well as to create a general preference for the DeleGator software solution over alternate programs.

## Method of Implementation

As there are no previous versions of the software, and the fact that it will be designed for use as a web application, the most suitable method for implementation is direct cutover, to be done at the completion of the final product. In this, it will be available as open source to any user that accesses it through a web browser.

# Planning a Software Solution

## Programming Languages

The application will run on a web-based platform. This platform was chosen based on experience with the web suite (HTML, CSS, JavaScript), as well as for ease of access for users, provided they have a device that can connect to web pages on the internet. Additionally, it will be built off the Node.js and Angular.js frameworks to bring ease to the backend development, while using features of semantics-ui.css to style the interface.

## CASE Tools

In preparation to building this software solution, relevant CASE tools (computer-aided software engineering) must be considered when undertaking a software solution. The primary CASE tools used to create the DeleGator application will be the Atom integrated development environment and its included functionalities. This includes debugging tools such as creating breakpoints and single-line stepping. In addition to this, algorithm development will be designed with the use of draw.io, to construct flowcharts that will plan the structure and logic of the solution.

## Development Approach

When preparing to build a software-based project, the methodology of how the building process will be executed must be planned, to give the most ease and suitability for resources in making the product. There are five standard approaches to software development to account for a developer’s time constraints, the complexity of the project, and the resources available to them. These are (from most structured to least): structured, agile, prototyping, RAD (rapid application development), and end-user.

The structured approach consists of teams of developers that are purposed to complete distinct stages sequentially throughout the duration of the project. Because of its general necessity for complex projects and its suitability for well-finished and reliable solutions, it is more-so used for large-scale developers with less restrictions on time and cost to produce a high-quality piece of software, often intended for a wide group of target users.

The agile approach, unlike the structured approach, focuses less on the overall predefined stages of the project, and favours team cohesion and cooperation for a generally less-complex software solution. Because of this, detailed predefined requirements are not as necessarily ingrained into the project’s foundations and thus, the project specifications may be loosely defined or have flexibility to adapt over time. As there is a greater significance placed on the coordination of developers in the agile approach, multi-skilled developers are preferred, to allow for optimal speed in development, communication with clients to adapt to changing specifications, and maintenance of the product.

The prototyping approach to software development revolves around effective communication between developers and users. A prototype is made of the developer’s understanding of the problem, and feedback is then given from the user to more clearly shape the design specifications that the product should be modelled after. Each iteration of a prototype brings the product closer to completion, at which point it will be ready for implementation by the clients.

The rapid application development approach is characteristically defined by its intended use: to build a piece of software that minimises cost and time expenses for the user. To achieve this, pre-written segments of code are recycled into the new project, modified to suit the specifications. To further the developer’s ease within the time constraints, the need for formal stages is diminished.

Lastly, the end-user approach is the least formal of all approaches to software development, wherein the user creates a software purposed for their own use. This generally makes use of existing modules and functions and is made for a small budget over a short duration. Like RAD, it lacks formal stages and is done with little planning by the developer.

The chosen approach for this software-based solution is the agile approach. This was favourable due to it being adaptable to changing specifications. This is suitable for a web-application as web apps often require maintenance to adapt to the changing needs of the user. As there is no need to download new versions of the software when changes are made, implementing modifications are simple and hence, appropriate for this chosen approach.

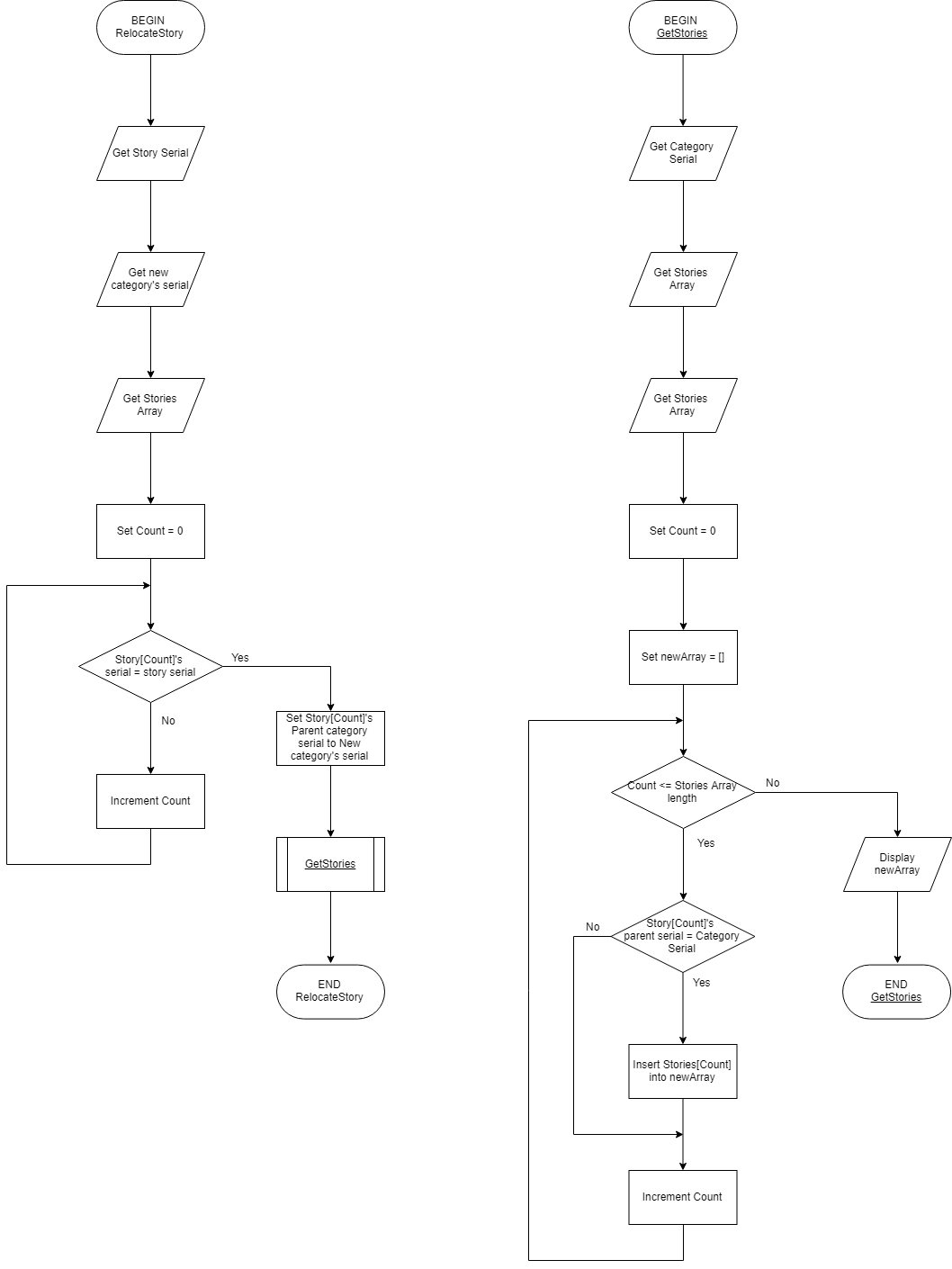
## Relevant Algorithms (Pseudocode)





The AddProject algorithm is executed when the user clicks on the “Create Project” button. This is almost identical to similar algorithms, such as AddCategory and AddStory, hence will not be listed.

The following algorithm conveys the logic behind relocating a story to a different category and then updating the display to show the story in its new category. This works by effectively changing the selected story’s parent serial (which indicates the serial of the category it belongs to) to the serial of the new category. Once this is changed, the stories can be updated as if displaying for the first time. It is displayed by looping through all elements in the StoriesList array (see data dictionary) to determine the stories with parentSerials which match that of the category’s serial. Once it has found all the stories belonging to a given category, it returns them all in a new temporary array to display them in their respective category. This is repeated for each category within a project, to display them all on one page.



These algorithms convey the primary functions behind the planned application (excluding login and signup functions), as to demonstrate how the program executes its main purpose: to log and display tasks.

## Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Data Type | Length | Description |
| Username | String |  | The entered login name of the user. |
| Password | String | Min length: 6 digits | The user’s entered password. Converted to an MD5 hash before manipulation. |
| Email | String |  | The user’s entered email |
| usersList | Array of Records |  | List of all users, including their Username, Password and Email. |
| projectsList | Array of Records |  | List of all projects in the database, including their owner (username), and average. |
| project | Record |  | A single project record. |
| categoriesList | Array of Records |  | List of all categories, including their owner (username) and parent *project*. |
| category | Record | Max value of “mark” property: 100% (1) | A single category. |
| storiesList | Array of Records |  | List of all stories, including their owner (username) and parent *category*. |
| story | Record | Max value of “mark” property: 100% (1) | A single story. |
| Count | Integer | Max value will be length of usersList, projectsList, categoriesList, or storiesList. | Count to loop through above arrays (projectsList, categoriesList, storiesList…) |

## IPO Chart

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| **Username (string)** |  |  |
| **Password (string)** |  |  |
| **Login button click** | Check username, password with database | Display overview page |
| **Project name (string)** |  |  |
| **Username (string; can retrieve from cookies)** |  |  |
| **“Add project” button click** | Insert project into array of projects | Display updated list of projects |
| **Username (string; can retrieve from cookies)** |  |  |
| **Current project’s serial (integer)** |  |  |
| **“Add Category” button click** | Insert category into array of categories | Display updated list of categories when |
| **Project clicked** | Search through array of categories, inserting a category into a new array if its parent serial matches the serial of the selected project |  |
|  | Return new array | Display list of categories in the array |
| **Mousedown over a task/story, serial of story** |  |  |
| **New mouseover to a different category** |  |  |
| **Serial of the new category** |  |  |
| **Mouseup** | Change parent serial of story from old category’s serial to the serial of the new category | Display task inside new category |

## Storyboard / Concept Design

(Next Page)

## Logbook

### Term 1

#### Week 1

Created project files, copied pre-written modules into the new file, such as any related database functions, a HTML template, and set up the AngularJS framework. Angular.js was chosen due to previous experience being had with it in previous projects, and with all of its additional packages such as cookies and ease in making http requests, this was ideal for this web-based application.

#### Week 2 (backed up 08/02/2019)

Recycled a generic layout for the HTML page of a header at the top and a menu on the left side. Also imported a set of styles from a previous project with retro-styled 3d buttons and circular inputs. With this I reused an existing function to add a record of the user’s username, email and password to the database when the signup button is clicked. Similarly, this was done for logging in (see *Planning a software solution* for the relevant algorithm).

Example use of the calling the md5 hashing function, which turns stings that look like this:

*Password*

Into this:

*5F4DCC3B5AA765D61D8327DEB882CF99*



The “md5.createHash()” in the frontend code (previous image) is so that when the user enters their password, it immediately converts it into a hash so that when it’s sent over the network to the server, it cannot be intercepted and decrypted. This is also to ensure the security of the users in the database, so no passwords are listed anywhere.

#### Week 3

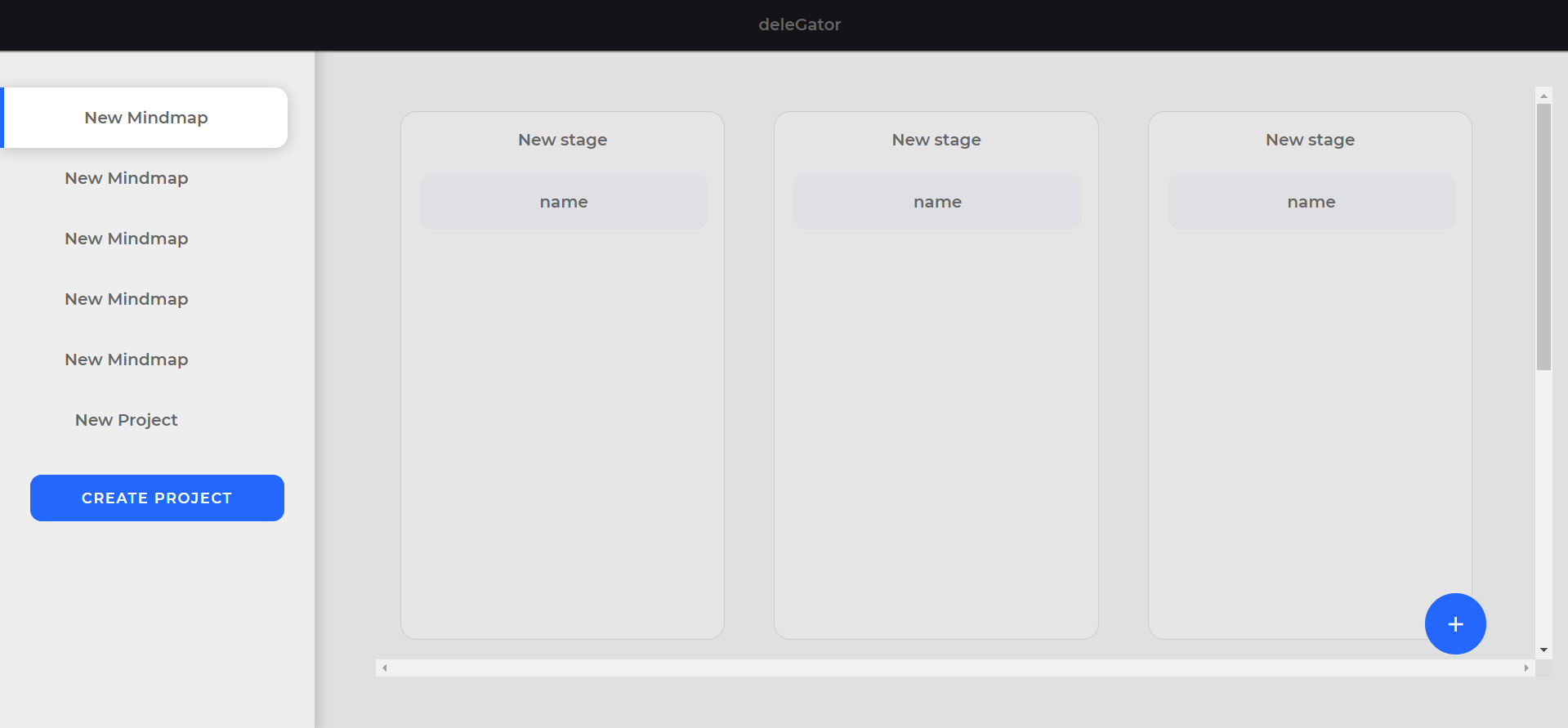
Continued work on tying up loose ends on the login functionality. When finished, I began to work on the server-side code to add a project to the database, owned by a distinct user. This utilised some existing functions from previous projects, such as the “add user to database” function while modifying the relevant variables and parameters. The request from the frontend to the server looks like this http request:

Sample http get request code. Sends “*params”* object to “*domain/myPageManager”,* to then get a response, “*res”.*



#### Week 4 (backed up 22/02/2019)

On the main page, I experimented with my first conceptualisation to use a simple theme of a header, a sidebar/navigation bar, and a body for the content appropriate to the project tab that the user has selected (as seen in screenshot). Throughout this ideation, I mostly styled the page to suit this concept, until this result was achieved:



In evaluating this, I would say the overall style is too “bubbly,” which slightly emits a non-professional standard of design, which may call for reconsideration with a new design concept.

#### Week 5

Began planning the logistics of creating projects and categories. The logic behind this was that when the user clicked the “create project” button (left side underneath list of projects) or the “+” floating action button (bottom right), a new project would be inserted into the database and subsequently displayed in its respective list. Each new project is made such that it belongs to a specified owner (by username) and has its own unique serial, which is just an incrementing number. This system of giving each subject a serial was just to make it easier to identify and manipulate the correct subject instead of another subject by the same owner with the same name, to avoid complications or potential errors.

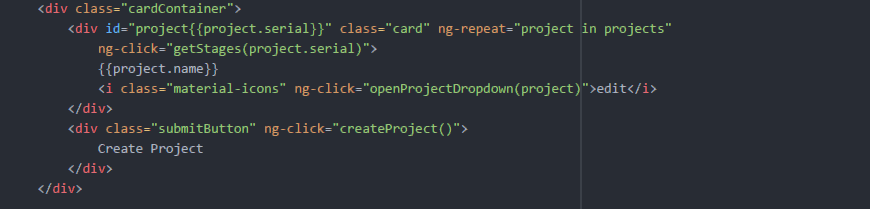
### Term 2

#### Week 1

# Building the Software Solution

## Source Code

### Useful/Common functions



The above code depicted above is an example of the technique used to display a varying number of same screen elements. This is used for:

* The projects menu list,
* The stages, and
* The list of stories within each stage

This uses Angular.js ng-repeat to iterate through the array “projects”, creating multiple of the same div with class “card,” where different properties can be altered by the properties of “project.” In this case, each “project” is an object that looks like:

{

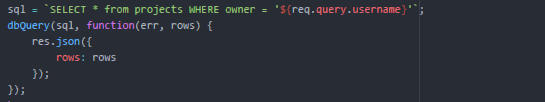
serial: 123,

name: “project\_name”,

owner: “username”,

}

The name can be accessed in the html with {{project.name}}, and functions relating to individual divs can take properties of “project” as parameters. This is useful for onClick functions, taking in the “serial” of a project, to then execute the relevant actions.



This code is an example of how data is fetched from the database. A query is set in the form of an SQL instruction, where parameters are taken from the user’s http request (as seen in the ${req.query.username} above). This instruction is passed into the dbQuery funtion, which executes the instruction in the database, then followed by running a callback function. If data is to be returned from the database (in that the sql instruction was a “SELECT” command), this data will be returned in the parameter “rows”. To send data back to the client, res.json() is used to send an object with either data from the database (if there is any), or it acts as a control parameter, sending a boolean indicating success or failure.

### Full source code (next page)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

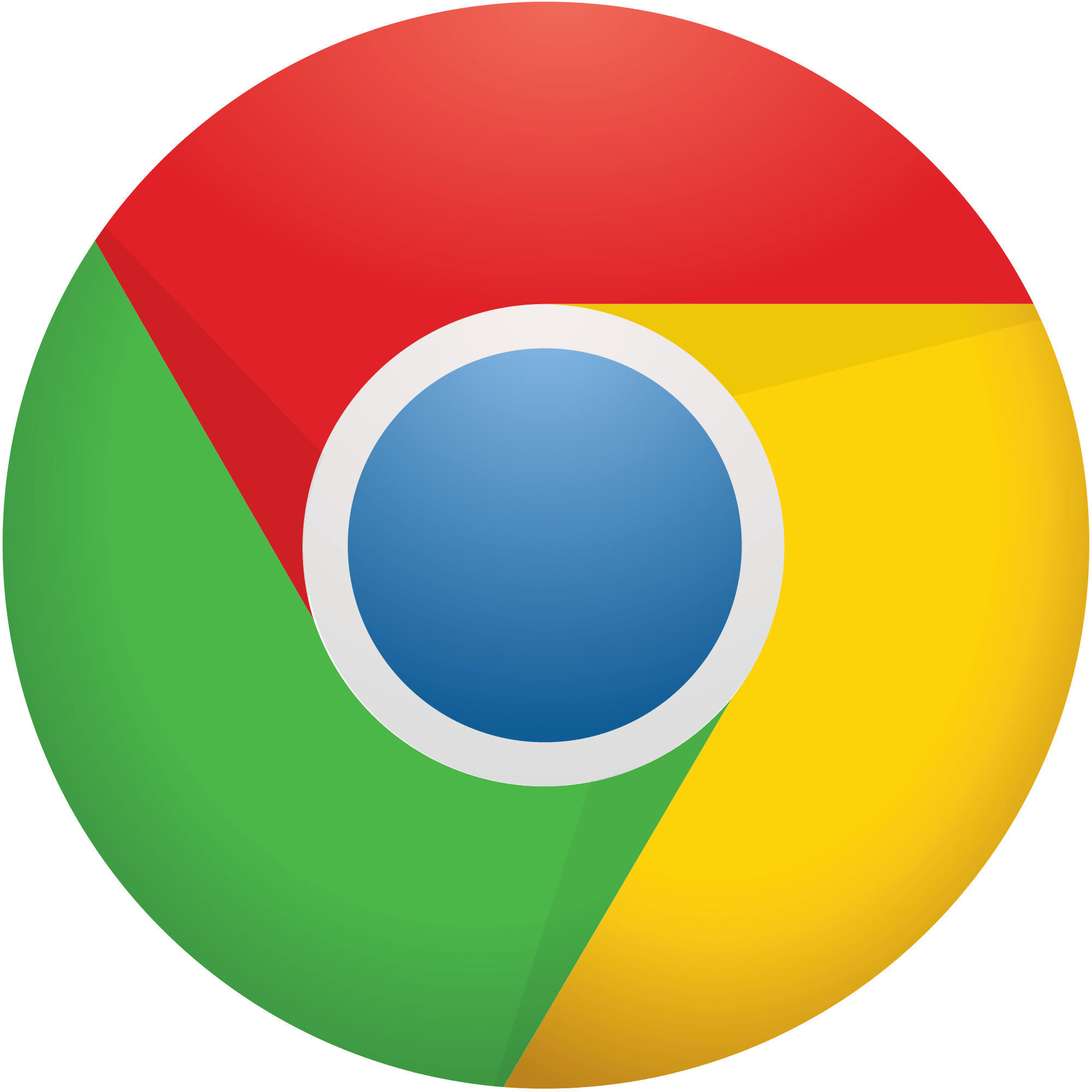
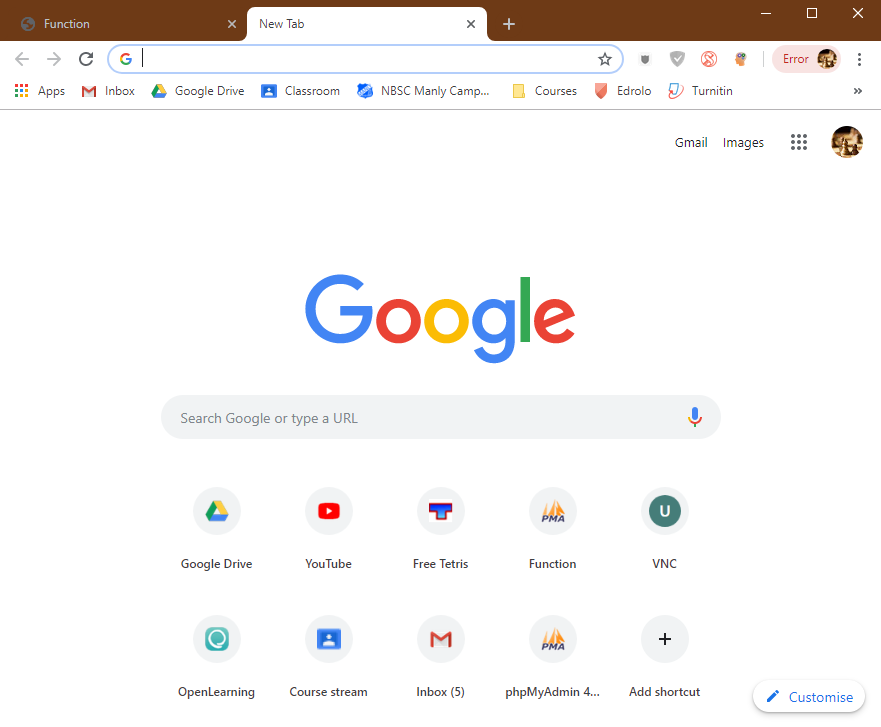
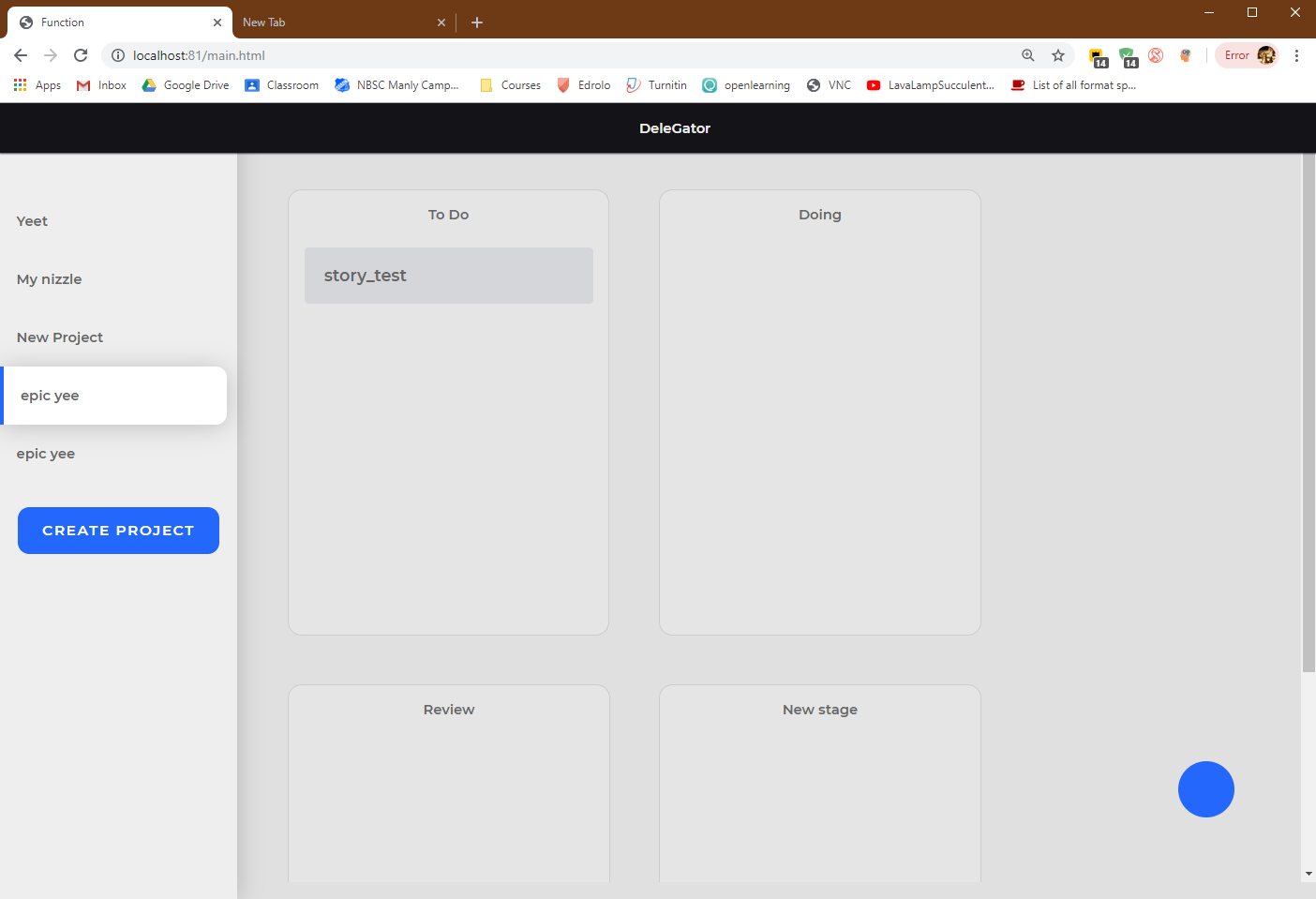
## Peer Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
|  | **What were your thoughts on the overall functionality of the software? Any specific comments?** | **What were your thoughts on the general aesthetics of the software? Any specific comments?** | **Any ideas for future improvements/how is it a piece of shit?** |
| Riley |  |  |  |
| Lewie | The software is extraordinarily functional and very intuitive, with the ability to group tasks by ”story” being one of the coolest and most innovative uses of the “drag and drop” formula. | The aesthetics of the software are very nice, with the neutral grey tones working well, thematically, for a piece of organizational software. The rounded corners are also a nice touch | I feel like the only improvement that I would recommend would be the ability to add tasks to the story. |
| Hayley | The ability to create new projects and then smaller tasks and steps within those projects makes it very easy to abstract an assignment! It’s a great way of designing an organisation program and its very functional! | The crisp and clean layout of the program made it easy to navigate, this together with the neutral colour palette made it very aesthetically pleasing. | I’d love to see the moveable tasks function completed! Also being able to add and remove specific tasks within the sections in each project would also be great. |
| Moe |  |  |  |
| Kate |  |  |  |

# Documenting to meet Installation of software solutions

## Installation Guide

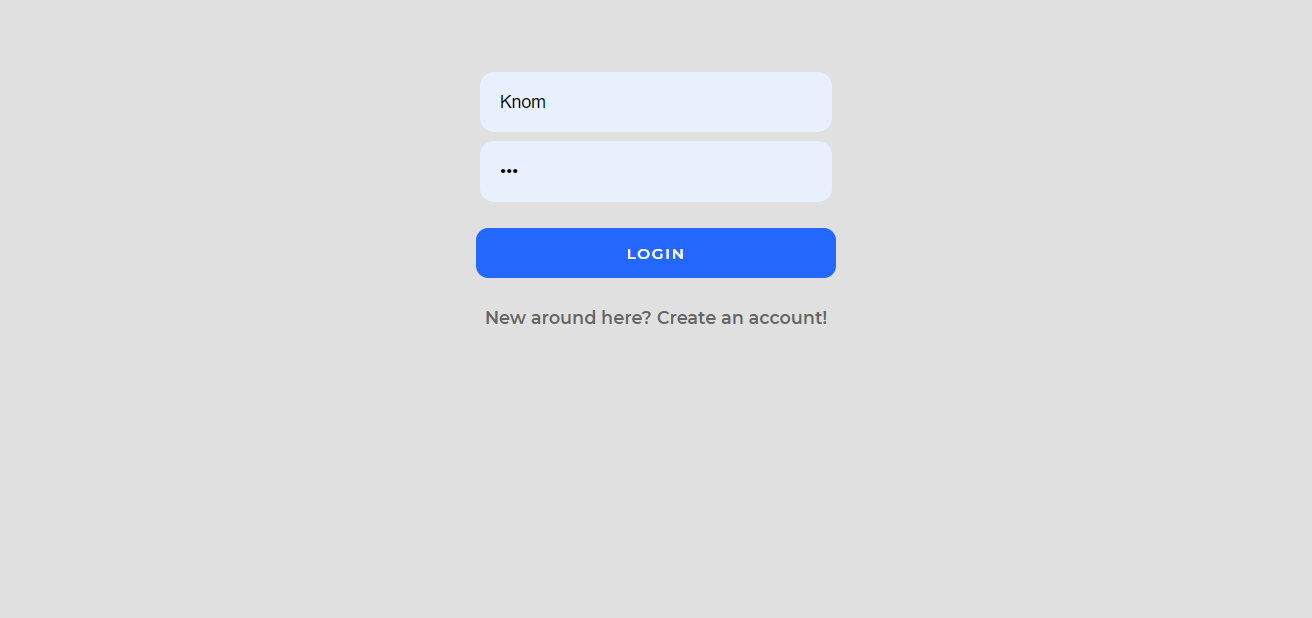
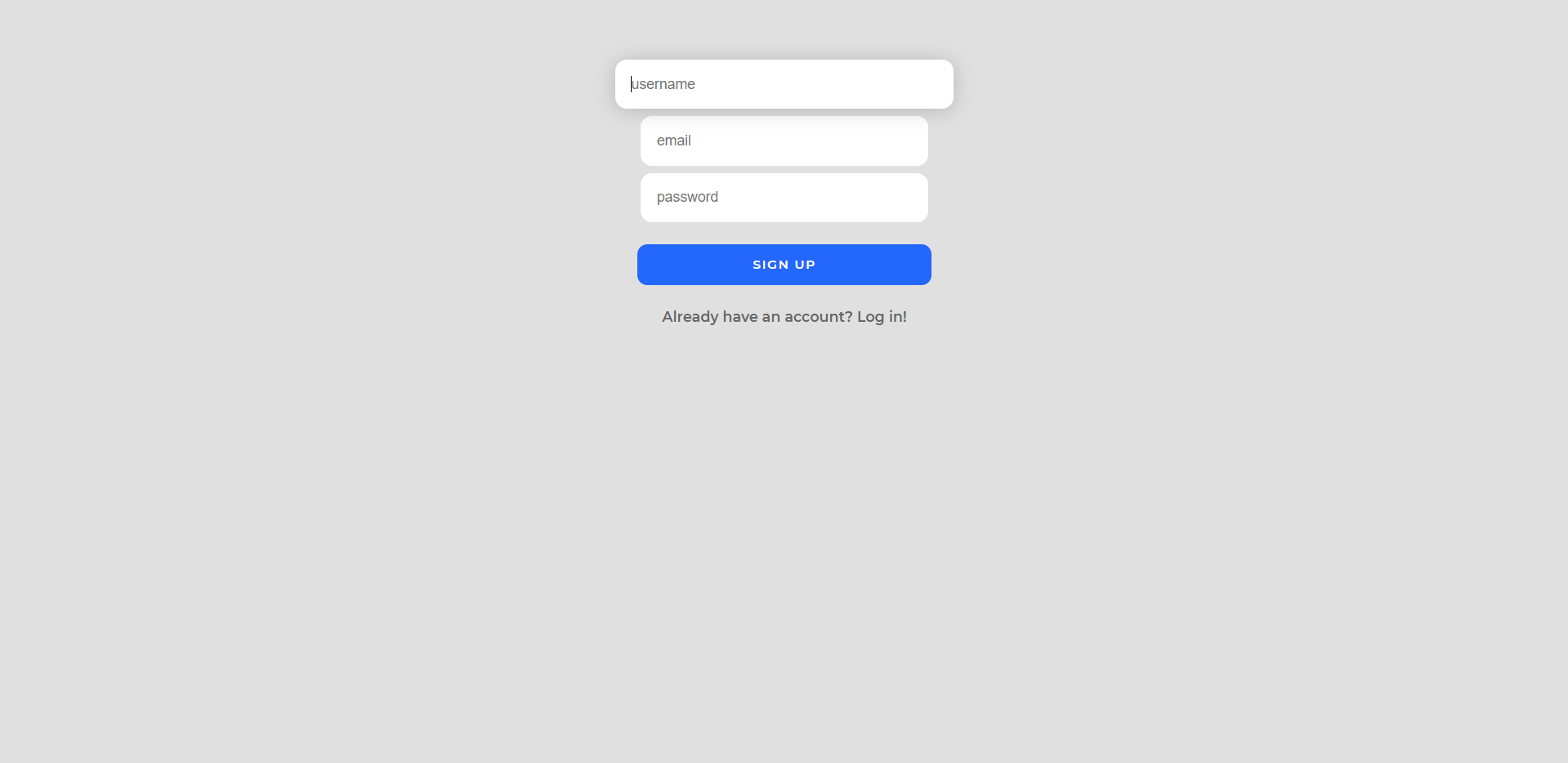
The DeleGator application is an online web-based program, opposed to an executable file stored on the user’s computer. To access it, open a web browser (Google Chrome, Firefox, Internet Explorer), and navigate to delegator.xyz in the address bar.



## User Manual

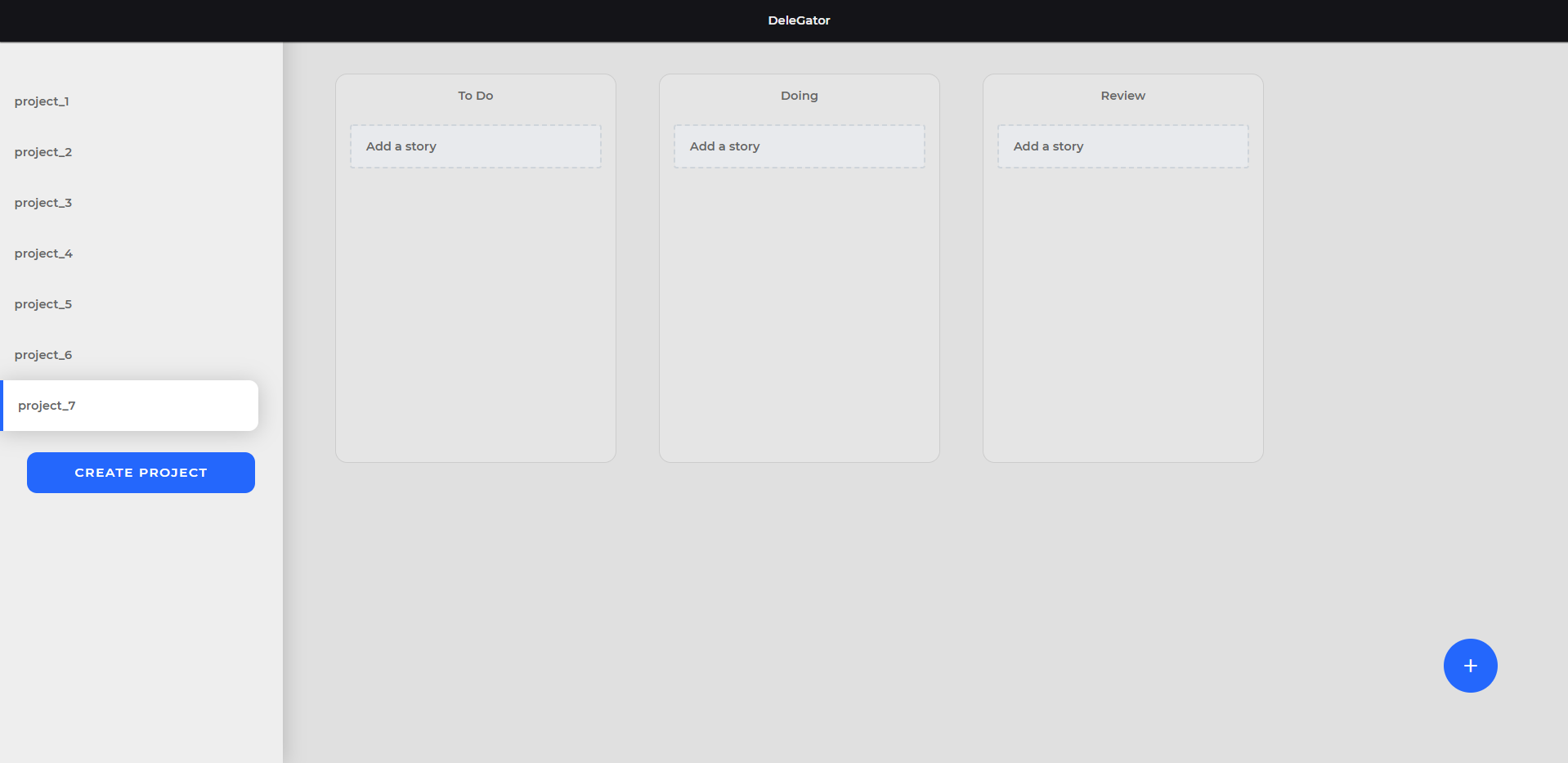
### Logging in / Signing up

Upon connecting to the Delegator web application, one will be faced with a login screen, displaying a textbox for the user’s username or email, a textbox for their password, and a button to submit their information to then take relocate the page to their “home page.” Other options are to sign up, which can be accessed by clicking the “sign up” link giving similar input options for the user.

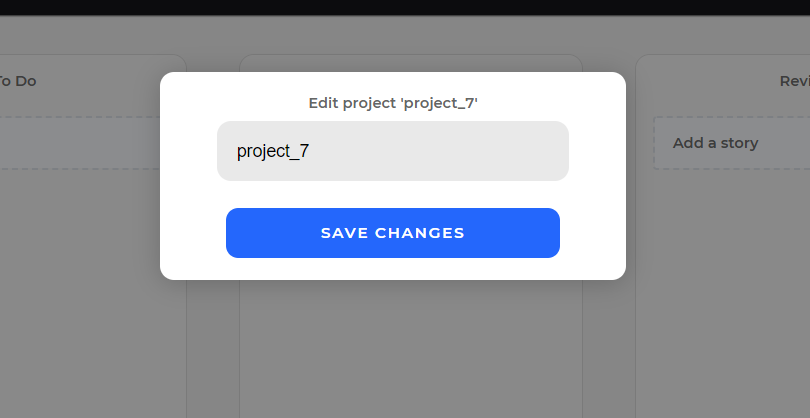


### Home page elements

Upon signing up or logging in, the user will be relocated to their home page, which consists of multiple on-screen elements. To the left, one may see an empty list containing a blue “Create Project” button, with the rest of the page being blank. With use, the left-side menu will display a list of all the user’s projects, and the remaining body of the page will consist of tiles or “stages”, each of which containing smaller cards or “stories”. Stages are designed to outline the different categories that stories may be grouped into, with stories being the actual smaller, specific tasks/notes (how they are used are up to the user).

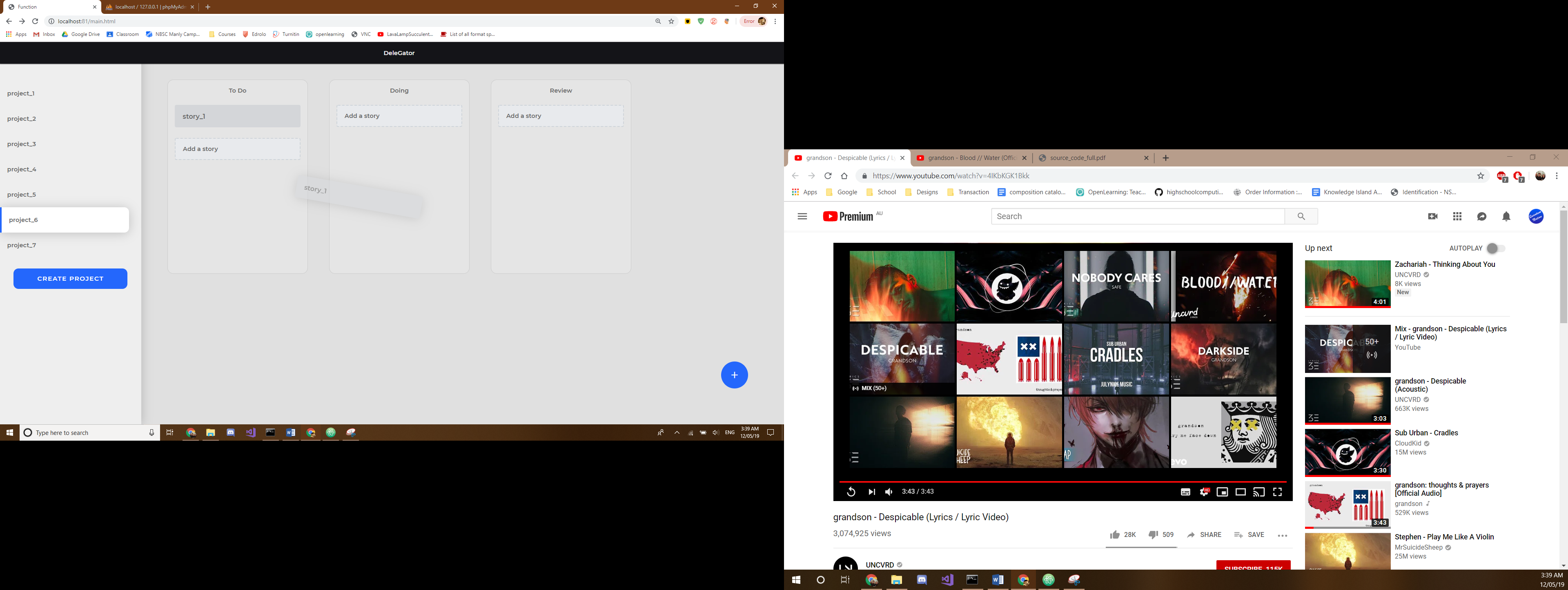


On clicking the “CREATE PROJECT” button in the left-side menu, a dialogue box will appear prompting for the user to enter the name of their new project. When a name is entered and the “Save Changes” button is clicked, the new project will appear in the left-side menu. Entering this project by clicking on it will display all the stages for that project. Each new project comes with 3 default stages (titled “To Do”, “Doing”, and “Review”, respectively). To add more stages, the user must click the circular FOB (floating action button) in the bottom-right corner. Again, this will prompt the user to enter a name for the new stage.



### Working with stories

To create a task “story” within a stage, the user must click the “Add story” button inside the respective stage. Like creating a project and creating a story, this will prompt the user for a name. Upon entering a name and pressing “save changes”, the story will appear in the chosen stage. Stories can be reallocated to different stages by clicking and dragging the story and releasing the mouse inside another stage. To rename stories, hovering the mouse over a story will make a pencil icon appear. Clicking this will cause a prompt box to open with a text box for the new name for the story. When a new name is entered and the changes are saved, the story will now be renamed to that new text.



## Evidence of testing methods

The DeleGator web application was tested across multiple web browsers to ensure quality and compatibility with all user systems.

# Evaluation and suggested modifications

# Bibliography

* Catalogue.pearsoned.co.uk. (2018). [online] Available at: <http://catalogue.pearsoned.co.uk/samplechapter/078972670X.pdf>
* Davis, S. and Fendall, J. (2012). *Software design and development*. 2nd ed. Parramatta Education Centre.
* W3schools.com. (2018). *JavaScript String() Function*. [online] Available at: <https://www.w3schools.com/jsref/jsref_string.asp>
* <https://semantic-ui.com/> “Free and Open Source (MIT)”