1 Project Documentation

1.1 How does it work?

1.1.1 *Project Structure*

The overall structure of our project relies heavily on JavaScript. Our backend is built on a NodeJS, which handles data to and from a MongoDB database. Our frontend is comprised of Aurelia, which is a JavaScript framework used for single page applications. All three of these components make up the structure of our application, and are constantly being used by each other.

1.1.2 *Theory of Operation*

The MVP Platform is an online application, so navigating to the website is the first step. A user can then sign in using their Google Account, at which point they'll be able to use our application. The whole of our application works in a simple manner, as all a user has to do to get started is click on the ”new book” button on the main page. Users can also fork a book on the website if they wish to work off somebody elses work. The next step is to start creating chapters, and then individual scraps from those. A user can add images, paragraphs, or other work to a scrap. The option to take others' scraps and chapters is available as well. When the book is finished, having it reviewed and printed are the only parts left to accomplish.

1.2 Hardware & Software Requirements

1.2.1 *Hardware Prerequisites*

The MVP Platform is a web application with search and user functionalities requiring a database. As such, it requires a decently powerful machine.
The platform has been successfully used on a machine with

1.2.2 *Software Prerequisites*

To begin setup of the MVP Platform, there are a few things that need to be installed.

- NodeJS and NPM: NodeJS is the runtime for the application itself, and NPM is the package manager included with it for installing 3rd-party libraries.
- MongoDB: This is the database used to store things like user favorites and login tokens.
- ElasticSearch: this is the search engine behind the search feature.

1.3 Installation Instructions

1.3.1 *Setup*

1) Clone the source code:

```
$ git clone https://github.com/CS461-MVP/mvp_platform.git
```

2) Move to the source code directory:

```
$ cd mvp_platform/dev
```

3) Install all scrap library dependencies:

```
$ cd scrapjs && npm install && cd ..
```

4) Install all backend Node dependencies:

```
$ cd backend && npm install
```

1.3.2 Running the backend

To run the server, simply run:

```
$ npm start
```

This will start the server running on localhost, port 8000. To verify that the server is running, in another terminal window run:

```
$ curl http://localhost:8000
```

This will return a 404, as there is no root page, if the server is successfully running.

1.3.3 Running the frontend

The frontend is a static, single-page webapp. To serve this, a standard `nginx` or `Apache` static file server is recommended; alternatively, for testing, the pages can be served with Python's built-in HTTP server:

```
$ cd /path/to/clone/mvp_platform/dev/frontend
$ python -m SimpleHTTPServer 8080
```

The website may now be visited at `http://localhost:8080`.

1.3.4 Deploying for Production

For detailed instructions regarding setting up `Nginx`, creating a service, and setting up a reverse proxy, please find the nearest sysadmin. Such a setup is outside of the scope of this documentation. In brief:

1) Create a service that runs the backend as shown above.
2) Install `nginx`.
3) Create a configuration file that:
    - Create a rule that matches requests sent to `/accounts, /login, /favorites, /books, /chapters, /scraps, /images, /search, /users`. In this rule, reverse proxy the request to the instance of the platform running on port 8000.
    - For all other requests, serve static files from `/dev/frontend`.
4) Restart `nginx` to put the rule into effect.

2 How did we learn new Tech?

By far the leading factor of our team learning new technology was having each other as knowledge points. Some members of the team had previous knowledge working with a JavaScript framework, and others had experience working on a backend. One of the most beneficial qualities of our team was our constant communication. This made learning and asking questions very easy and fast, and aided in our own growth as developers.

Another major learning point was following documentation online. Unfortunately, the Aurelia framework isn't as polished as other JavaScript frameworks that have been around longer. This posed as a minor problem, but didn't stop us from accomplishing everything the team needed to.