MIGUEL LOPEZ

Building SPAs with Elm

Codemunity

Building SPAs with Elm

Codemunity

This book is for sale at http://leanpub.com/building-spas-with-elm

This version was published on 2019-03-13

Contents

Introduction

Welcome to the *"Building Single-Page Applications with Elm"* book!

We'll apply more advanced concepts in Elm and build a real-world single-page application, we'll see exactly what in a moment.

By the end of the book you will be comfortable building Elm applications by yourself as we will scale the Elm architecture and structure it nicely, we will perform requests and parse JSON to Elm data types as well.

We also have a Github repo[1], where you can find a branch for each chapter that requires coding, so you can check the final code of each chapter and compare to yours.

What should I know?

You should already be familiar with Elm syntax and the Elm architecture.

Don't worry if your not familiar with those topics, we have a free introductory course[2] where you can learn the basics of Elm, and we also have a tutorial series[3] that introduces the Elm architecture. Don't worry, we've got you covered.

What are we going to build?

We will build a service called **RecipeVault**, it is an application to find new cooking recipes, and you can save them as favorites too.

The specific requirements are:

- Users should be able to login
- Users should be able to register
- Users should be able to search for recipes
- Users should be able to save recipe as favorite
- Users should be able to view favorite recipes
- Users should be able to remove favorite

[1]http://bit.ly/2x12PsQ
[2]http://bit.ly/intro-to-elm-course
[3]http://bit.ly/cmty-elm-part1

What tools will be used?

We will use **Cloud9** as our online development environment, this will ensure that all of us are starting from the same point without installing additional tools, if you don't have an account there go ahead and create one now.

We will provide a `jar` file with a dummy API for the `login`, `register`, `save favorites` and `remove favorites` requirements, it has an in-memory data storage, so every time you run the API it will start without data.

For the `search recipes` requirement, we will use the Edamam API[4], so go ahead and create an account there, it's simple.

And of course we will be using `Elm` to build the rest of the application, to speed up the development process we will use the Create Elm App[5] package.

No more accounts I promise, and we will guide you through the setup and installation of all these tools in the next chapter, don't worry.

[4]https://www.edamam.com/
[5]https://github.com/halfzebra/create-elm-app

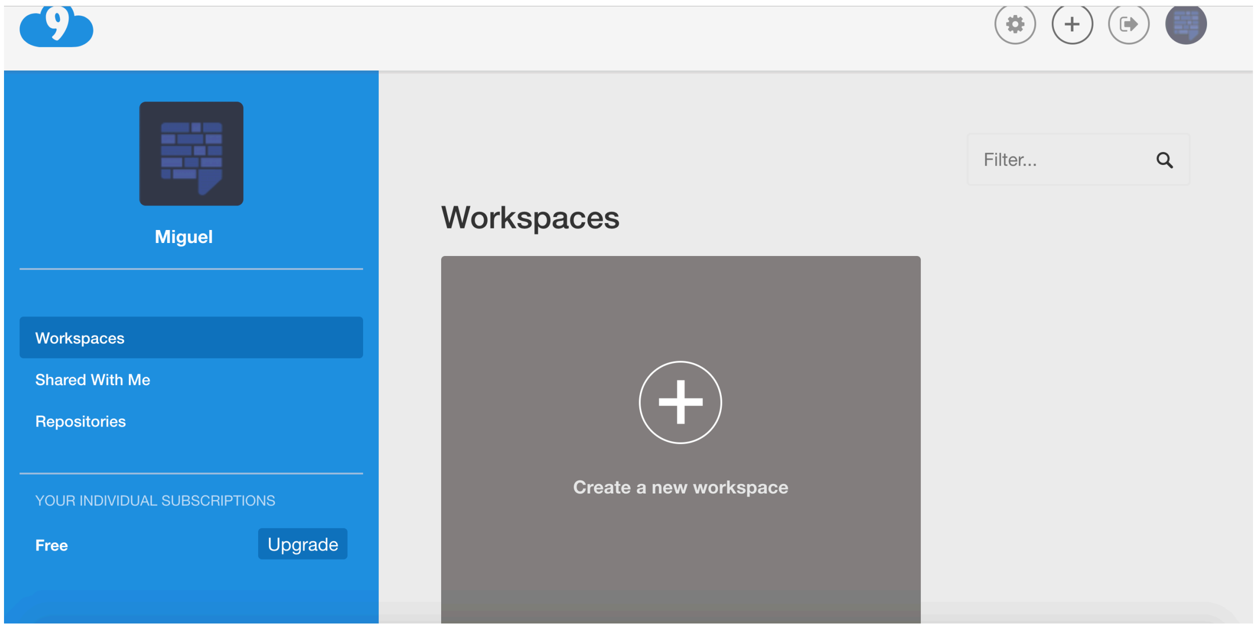
Project Setup

In this chapter we will create our Cloud9 workspace and install all of the dependencies we require to build RecipeVault.

Let's get started!

Creating the Cloud9 Workspace

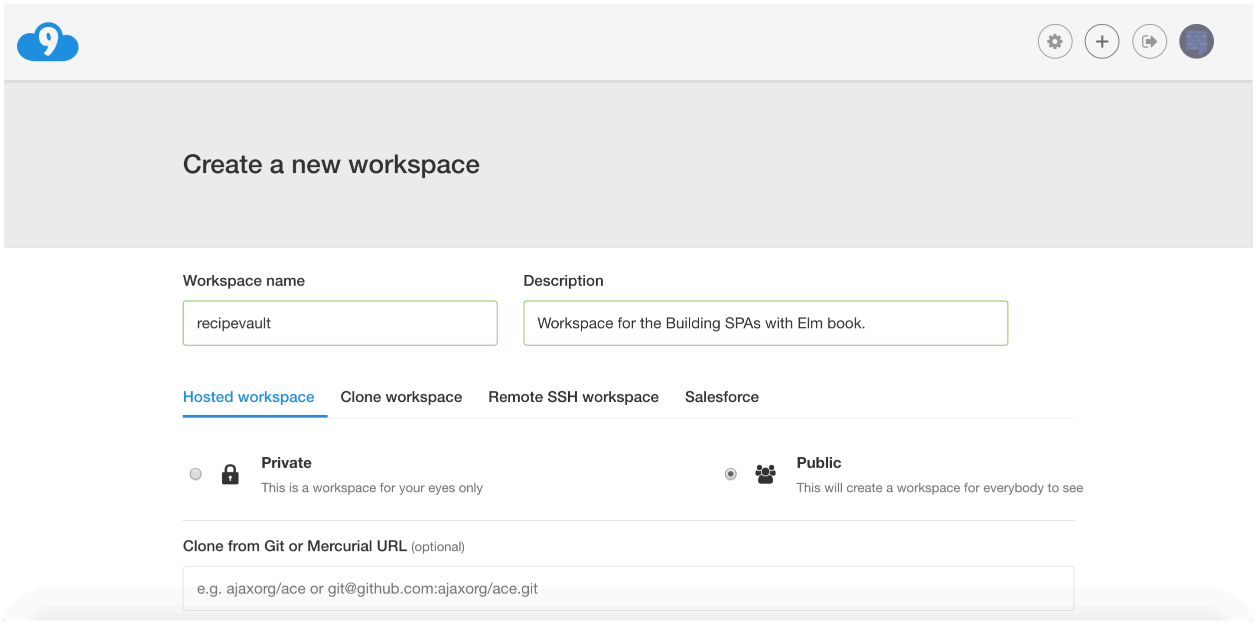
Once you have your Cloud9 account created, go to your dashboard and hit *"Create new workspace"*:



C9 Dashboard

Then we just need to fill out our workspace information:

- Workspace name: recipevault
- Description: Workspace for the Building SPAs with Elm book.
- Hosted Workspace - Public


Create a new workspace

Workspace name: recipevault

Description: Workspace for the Building SPAs with Elm book.

Private — This is a workspace for your eyes only

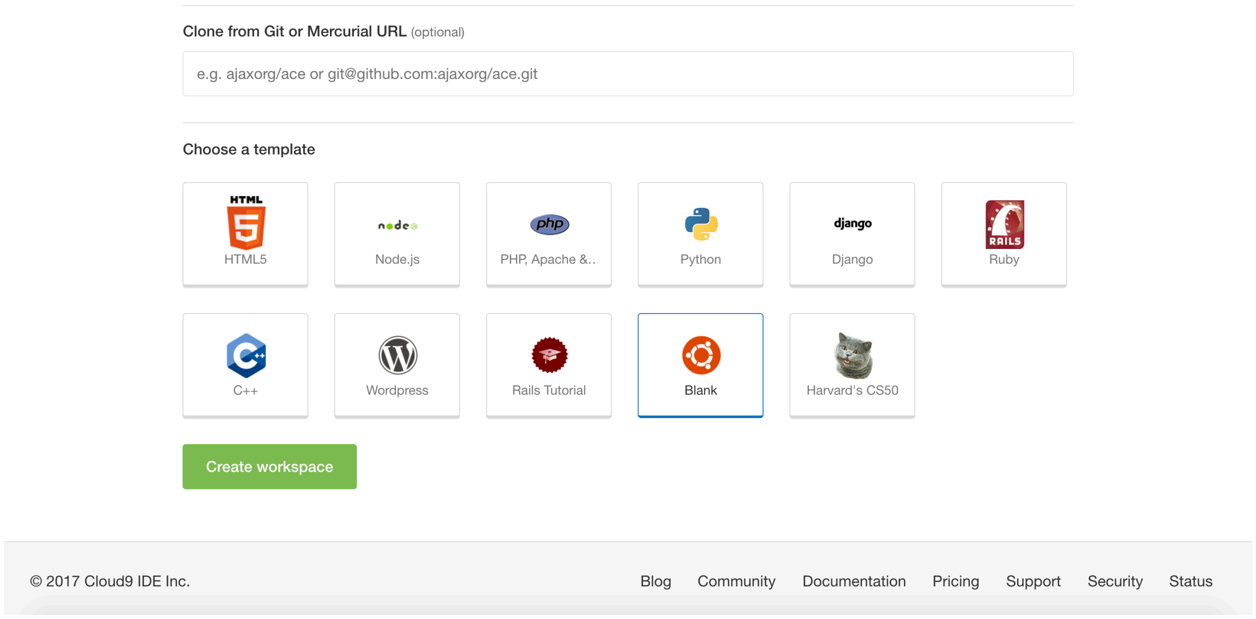
[✓ Public] — This will create a workspace for everybody to see

Clone from Git or Mercurial URL (optional)

e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git

C9 Workspace Creation 1

- Repository: we won’t use a repository
- Choose a template: Blank


Clone from Git or Mercurial URL (optional)

e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git

Choose a template

C9 Workspace Creation 2

We chose a **Blank** repository because the templates are used for project structure mostly, all workspaces come with Node and Java installed.

Hit **Create workspace** now.

Once the workspace is created, it's time to install our dependencies.

Installing the Node Dependencies

Now that we are inside our workspace, in the lower part of the screen you'll find a terminal, there we will be running commands to install our dependencies.

We will install a specific version of Node, which we already know works with the tools we'll use, to avoid possible issues.

Let's run the following commands one by one:

- `nvm install 7.8.0`
- `npm install -g elm@0.18.0`
- `npm install -g create-elm-app@0.6.3`
- `npm install -g http-proxy-cli`

We're using specific versions to avoid conflicts when new ones are released.

Those are the Node dependencies we need, the first three are self-explanatory, however we need an extra package because **Create Elm App** only listens to port 3000, and **Cloud9** only exports ports 8080, 8081 and 8081, so we will proxy all calls from port 8081 to port 3000, port 8080 will be used by our API.

NVM does not persist the Node version we just installed, this means that if we open a new terminal or restart the one we have open, we will have another version and no packages installed, so let's run a command to fix that:

```
nvm alias default node
```

Installing the Java Dependencies

Now it's time to install the requirements for our API, we start by installing Jabba[6], which is like **nvm** but for the JDK, then we will install the correct version of **Java**.

Again, run the following commands:

- `curl -sL https://github.com/shyiko/jabba/raw/master/install.sh | bash && . ~/.jabba/jabba.sh`
- `jabba install 1.8`

Same as NVM, Jabba does not persist the Java version installed, so let's run an extra command to achieve that:

```
jabba alias default 1.8
```

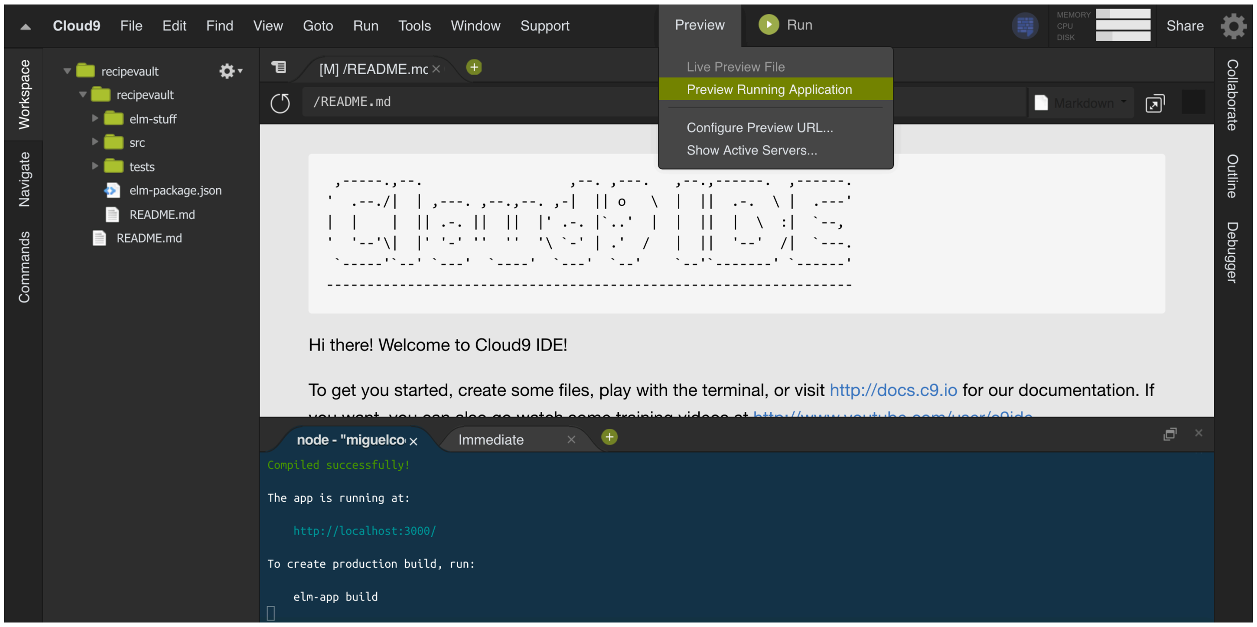
[6]https://github.com/shyiko/jabba

Creating and Running the Project

Let's create our project now, thankfully with **Create Elm App** it's really easy, run `create-elm-app recipevault` and after it's created let's go inside the project's directory, `cd recipevault`, run `elm-app start`, and normally that would be all that has to be done but remember we need to have a proxy in place.

Let's open another terminal, you can do so with the *"+"* button next to our terminal, and choose *"New Terminal"*, and inside run `http-proxy --hostname 0.0.0.0 --port 8081 3000`.

Now to preview our application and make sure everything it's working, click on the *"Preview"* menu, and select the *"Preview Running Application"* option.



```
 ,-----.,--.                  ,--. ,---.   ,--.,------.  ,------.
'  .--./|  | ,---. ,--.,--. ,-|  || o   \  |  ||  .-.  \ |  .---'
|  |    |  || .-. ||  ||  |' .-. |`..'  |  |  ||  |  \  :|  `--,
'  '--'\|  |' '-' ''  ''  '\ `-' | .'  /   |  ||  '--'  /|  `---.
 `-----'`--' `---'  `----'  `---'  `--'    `--'`-------' `------'
-----------------------------------------------------------------
```

Hi there! Welcome to Cloud9 IDE!

To get you started, create some files, play with the terminal, or visit http://docs.c9.io for our documentation. If

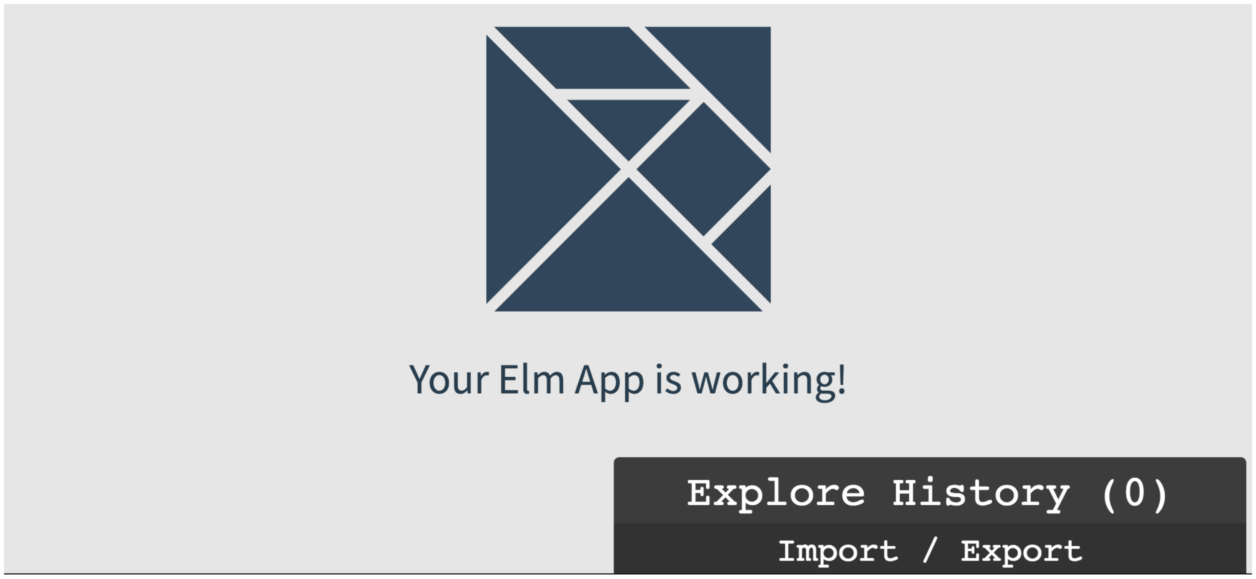
```
Compiled successfully!

The app is running at:

   http://localhost:3000/

To create production build, run:

   elm-app build
```

C9 Preview

This will open up another tab, and it will try to load the default URL, which points to port 8080, we need to set the port explicitly to 8081, if you need to read a bit more about this, check out C9's official documentation[7], it's really good.

If everything is working fine, you should see the following picture inside your newly opened tab:

[7]https://docs.c9.io/docs/multiple-ports



C9 Application Preview

Or you can simply access your URL in your browser, `http://<workspacename>-<username>.c9users.io:8081`.

Setting Up the API

Our API as we already discussed is made in Scala, and it is a dummy API with an in-memory data store implementation, this means every time you run the jar it will have no data.

First, let's download the jar file, open a new terminal and make sure you are inside our **recipevault** directory created by **Create Elm App**, run `cd recipevault` as soon as the new terminal is created, now run `pwd` and your currently directory should look like `/home/ubuntu/workspace/recipevault` if you created the project with the same name as ours.

Now that we are on the same directory, let's download the jar file by running `wget http://bit.ly/recipevault-api -O recipevault_api.jar`, once the download is complete, we can run it with th command `java -jar recipevault_api.jar`, you should see the following output:

```
Server online at 0.0.0.0:8080/
Press RETURN to stop...
```

The simplest way to test it is to paste the following URL in your browser `https://<workspacename>-<username>.c9u` the expected response is *"Request is missing required HTTP header 'Authorization'"* so don't worry.

Creating a Helper Script

Let's create a bash script that will allow us to run our three tools with a single command. Create a file named `start.sh` inside our project root directory:

```
elm-app start &
http-proxy --hostname 0.0.0.0 --port 8081 3000 &
java -jar recipevault_api.jar
```

We are just running the commands in parallel. To be able to run our script, we need to give it executable permissions, run `chmod +x start.sh` and it should do the trick.

Test the script by executing it with `./start.sh` in a terminal, and make sure to test the Elm application as as the API as we did previously.

Conclusion

We are done with setting up our project, it wasn't that hard, if you followed the steps you shouldn't have run into errors because of the tools we chose, however if you did, just get in touch[8] with us and we'll gladly assist you.

[8]https://www.codemunity.io/contact