STUDENT Daniel Jameson

Summary

This document serves to demonstrate the web-frameworks project being carried out by Student T00158237.

It is a web application that facilitates the use of a login, registration and data page of the application.

Web-Frameworks

Part 2

Contents

[Introduction 2](#_Toc149162924)

[Views 3](#_Toc149162925)

[Controllers 6](#_Toc149162926)

[Routes 7](#_Toc149162927)

[Models 8](#_Toc149162928)

# Introduction

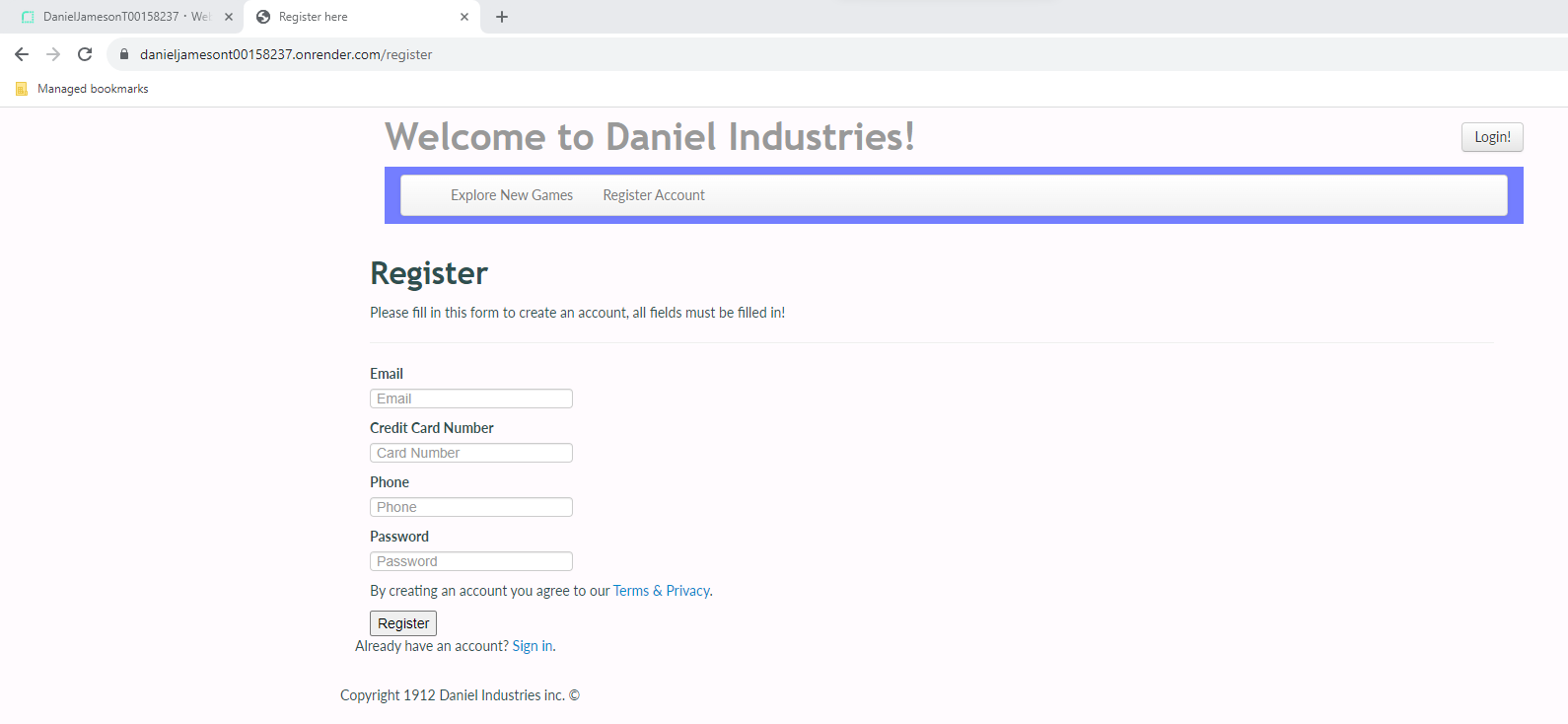
This project is going to describe how a web application is made using a web-frameworks technology stack. In this case, the tech stack involved is MongoDB, Express.JS, Angular.JS and Node, or (MEAN) stack for short. This document serves as a look into the making of such an application. It was made based on the book Getting MEAN 2 by Simon Holmes, as he built an application using the same tech stack. It is going to feature pictures and a description of the application throughout the build process and aims to describe to the reader the process of making the application functional.

My application is a clone of the website Steam, which is a website that facilitates the purchase of games via customer details. At this point in the project, the data is still primitive, as it is manually stored in the controllers for the project. It has a basic schema for customers and games which store basic details about each field. It will support three pages, a login page, a registration page and a page where the user can select games for purchase. This will attribute the games to their account and facilitate their purchase.

The Github link can be found at the following address: <https://github.com/DogPope/webframeworks/>

And the Render Link can be found here: <https://danieljamesont00158237.onrender.com/register/>

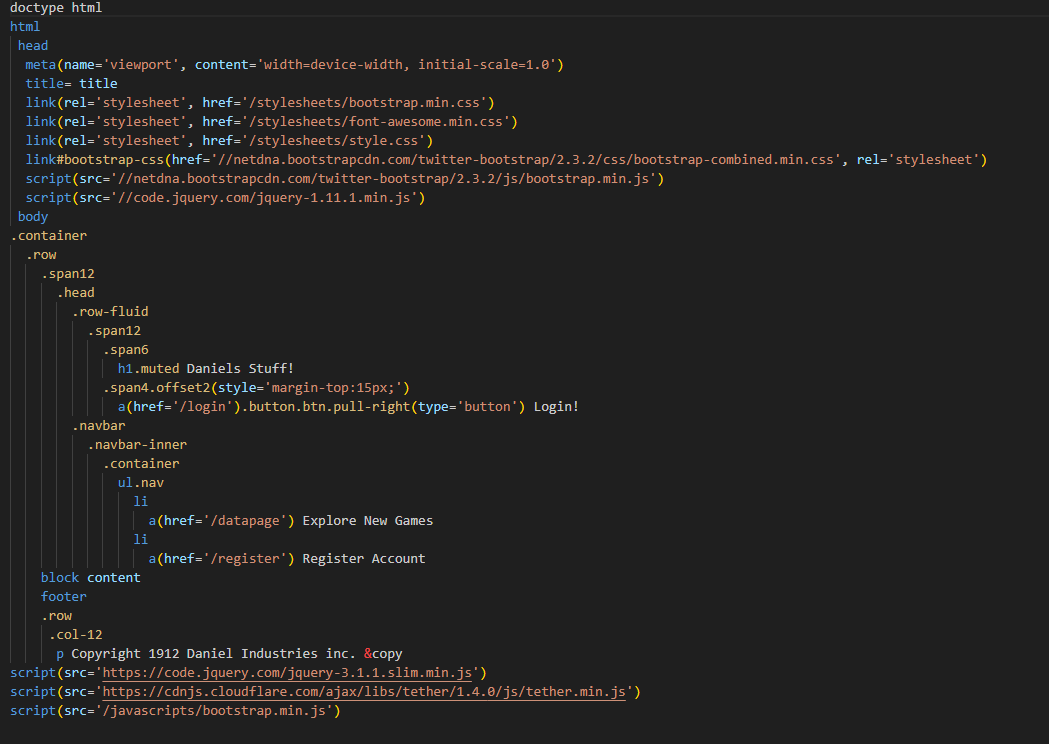
* \* The Node\_Modules file was not uploaded to Github because of its size. As a result, it may or may not work depending on where it is deployed. The Render.com version of the site will currently work as long as it is deployed, through putting npm install for the build command and npm start in the Start command. It should be noted that the main page, <https://danieljamesont00158237.onrender.com/>login should work to display the site when it is deployed.
* Edit: Never mind, it uploaded the Node\_modules file after adding it to .gitignore for some reason.



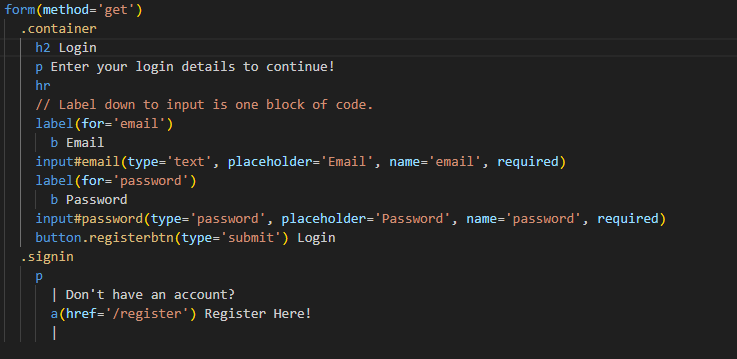
# Views

This section describes the views on display for this project. The views folder for this project contains the data and layout for the application, as well as having facilities for the user to Log in and Register. At this point in time, the data is hard-coded in through the use of the controllers. It is going to access outside variables located in the controller in the locations.js file.

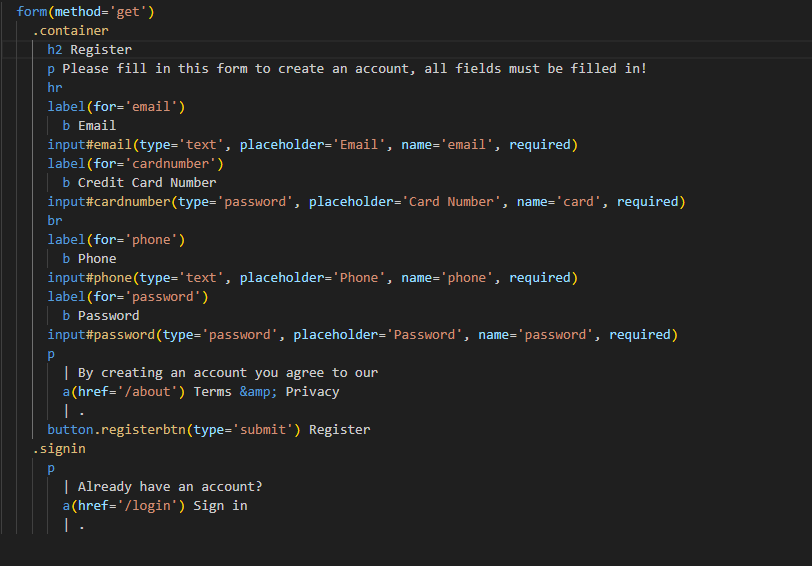
* layout.pug - In the layout file, I have changed the header to a new one that I took from bootsnipp.com, converted to pug.js and transferred to the Layout file. You’ll forgive me for not remembering which Navbar I actually took, I changed the style elements for it and cannot remember what the original looked like. Otherwise, I would link to it here. The code for the Layout file is pictured below.



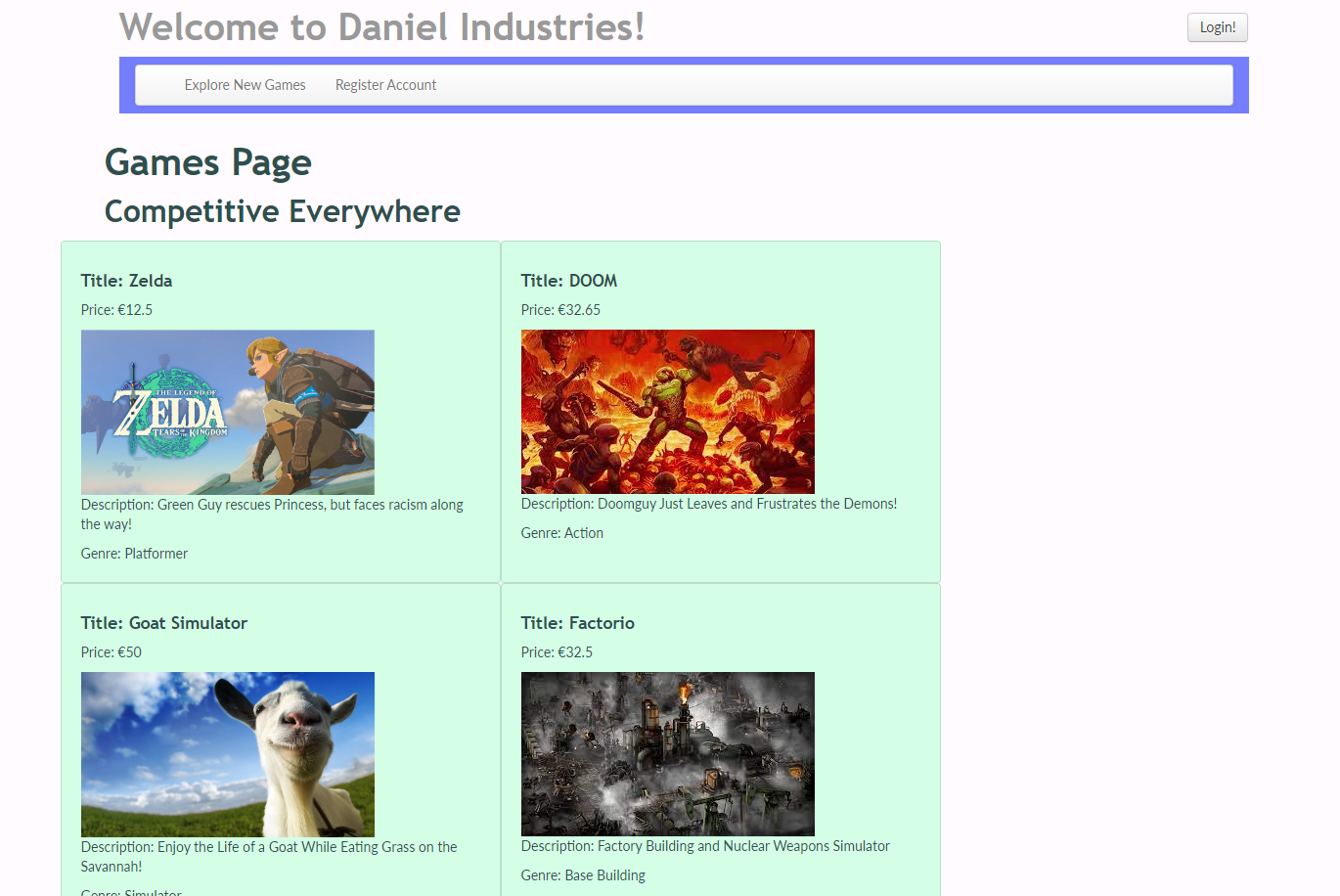
* login.pug – This file contains the login forms for already registered users to login. It has a simple form that prompts the user for their email address and password to use as login information. Because of the use of pugs extends function, it uses the layout.pug file for the header and footer, making the main part of the file relatively small. The file only contains form information and links to other pages, and references the header, footer and navbar. It can be seen below.



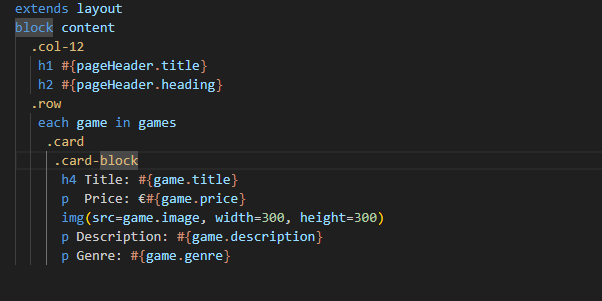
* registration.pug – This page includes a registration form, although it is slightly more detailed than the login form in the previous page. It requests more information from the user because to create an account, the user is going to have to supply more information such as a credit card number and a phone number to back up their account or to leave room to expand with two factor authentication to be added later.



* datapage.pug – This page is where the main data of the application is displayed. It contains variables in the form of headings displayed to the user and then goes on to describe the product to the user. This process involves using a pug loop to display the data that is hard coded into the controllers. It is stored as an array in its controller. When it is looped through, the contents of the array are then displayed to the page. Here is a picture of the completed Data Page with some sample data coded in.

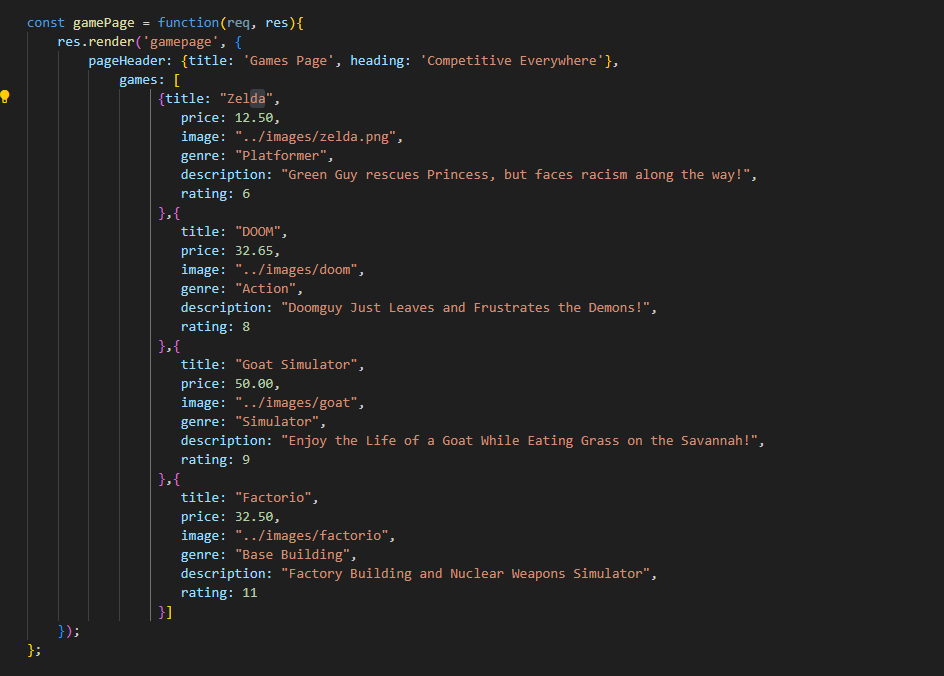


And here is a picture of the code used to render it.



# Controllers

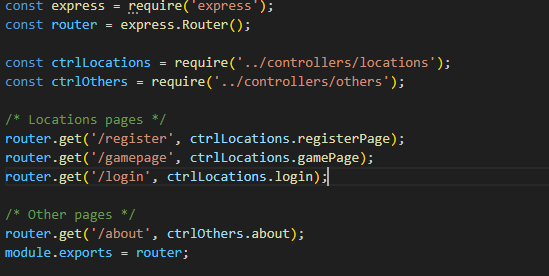
This section is going to describe the various controllers on display for this project. Controllers in the app\_server folder are used for controlling the data stored in them for export to the various views in the project . In locations.js, the main variables for the project are currently stored. They are hard-coded and imported directly from their pug files. I was mainly playing around at this point by putting in different variables and trying to send them for use as headings to see the different methods pug has for implementing said variables. The only really notable thing here is the data for the product page, as it is stored as an array and looped through in its corresponding file. Here is a picture of the data for the games page.



At this point, others.js still has its link to generic-text. I’m not sure why, I might find some use for it later, so am reluctant to get rid of it. It could be a useful part of my framework for later, we will see.

# Routes

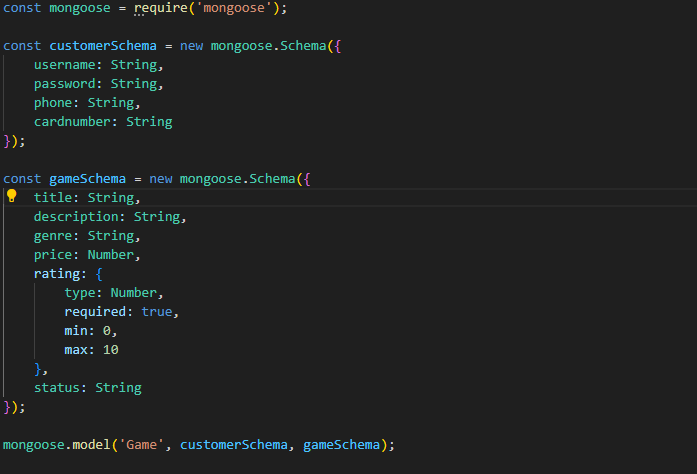
This section describes the process of finding the URLs necessary for the project to function. It currently contains some variables for the requests that the controller has to process. It uses the router in express to access using a common route prefix. It takes the variables from the controller and uses them to give variables to the page being loaded using HTTP get requests. It uses module exports to accomplish its goal.



# Models

This section describes the various data models that this project incorporates into its solution. The models folder contains the database information. It has two files, db.js and locations.js.

* db.js – This file contains mongoose dependencies, a string for connection to the Mongoose database, and attempts to connect to MongoDB. If it is successful, it sends a success message to the command prompt and a failed attempt to connect is sent if it fails to connect.
* Locations.js – The locations.js file currently contains schemas that are going to be necessary for the database to write to. It contains two schemas, one for a customers data and the other for a games data. The schemas are not terribly complex at the moment, but room for expansion is always good. The game data contains fields for a title, description, genre, price, rating and status. The customer schema contains basic information such as username, password, phone number and a credit card number for charging the user for a purchase. I will probably add to these data fields later, maybe with dates and sequestering data into sub-schemas.
* The two schemas are then sent to a database that is associated with the application using the mongoose.model method, along with its title.



* The schema that is in my Mongoose Database does not quite match the schema that is in the locations.js file currently. This is because I want to be able to see later whether or not adding more things into a framework is a task that is worth doing. It may be a complicated process, but I will try to add all the relevant fields as they become appropriate to add. Below, I have included a screenshot of the MonogDB instance with data added into two tables, one called customer, another called steam. The steam table is named after the Steam website.