University of North Carolina at Charlotte

Ruby on Rails

Movie Review Website

Rachel Conforti, Alaa Alharthi, Devashri Gadgil,
Rahul Sai Kancharala

29 November 2021

Table of Contents

Introduction and Motivation

Ruby is best used for web applications using the Ruby on Rails framework. Knowing that popular applications built-in using Ruby on Rails such as AirBnB (Hospitality Service), GitHub (Version Control Repository), SoundCloud (Online Music Distribution) motivates us to learn ruby on rails. Moreover, Ruby on Rails is an open-source framework; its active community allows us to use a wide variety of tools available online. We decided to build a movie review application using a movie scaffold tool which gave us a quick starting point in building our application. Using a scaffold that operates model-view-controller architecture provides us with a solid structure for our application. Our project allows users to explore a list of movies and add their favorite movie if it does not exist. Each movie has a review and recommendation counter that helps users investigate the movie they want to watch. Working on such an application using ruby on rails benefited us by developing a knowledge base both in learning a new programming language and learning web application development.

The Paradigm of the Language

Ruby on rails is a multi-paradigm programming language. Its paradigms are object-oriented, functional, and procedural. Ruby on Rails is an Object-Oriented language which means that it uses objects to make up your system. The object-oriented languages paradigm's goal is to have as much modularity and reusability as possible. Ruby on Rails also uses a functional paradigm. A functional language is built on the idea that everything is bound to a mathematical function style which in turn is bound to functions. This means that its programs are constructed by applying and composing functions. Functions can be bound to a specific name and can be passed as arguments, and even used as a return for other functions. There are two main concepts around functional programming. The first one is pure functions which are functions

that always produce the same output for a given input. And the second concept is higher-order functions which are functions that can take another function as an argument or return a function. Ruby uses both of these functions within its programming language. Rubys is also a procedural programming language which means that it runs on the idea that its programs are a sequence of instructions. The program is split up into a set of calls, which are functions. Within Ruby, however, there are no function calls, only methods can be called. So, within Ruby, we just create methods outside of the class.

History and Evolution of Ruby on Rails

The creator of Ruby is Yukihiro “Matz” Matsumoto, who blended Perl, Smalltalk, Eiffel, Ada, and Lisp. Ruby was released in 1995 with the hope of making a true object-oriented scripting language. Mr. Matsumoto knew Perl but did not like that it had the “smell” of a toy language. He also knew Python but felt like it was not a true object-oriented programming language and that OO was an add-on more than anything. So, he decided to make his own. Known for the quote “trying to make Ruby natural, not simple” also stating Ruby is simple in appearance yet complex on the inside, just like our human body.

After the release in 1995, Ruby in English started development with its first official release on December 7, 1998. In 2005 a large swell happened around Ruby that encouraged the development of Ruby on Rails, the high-performance web application framework that we used for our project. The Ruby community has been taken over by Ruby on Rails enthusiasts, even getting big enough to start shipping with the Mac OS X out of the box.

Elements of the language

Ruby's Reserved Words

Reserved Words	Definition	Reserved Words	Definition
alias	Creates an alias between two methods (and other things).	nil	A false value usually indicating "no value" or "unknown"
and	Short-circuit Boolean and with lower precedence than &&	not	Inverts the following boolean expression
BEGIN		or	Boolean or with lower precedence than ||
begin	Starts an exception handling block	redo	Restarts execution in the current block
break	Leaves a block early	rescue	Starts an exception section of code in a begin block
case	Starts a case expression	retry	Retries an exception block
class	Creates or opens a class	return	Exits a method
def	Defines a method	self	The object the current method is attached to
defined?	Returns a string describing its argument	super	Calls the current method in a superclass
do	Starts a block	then	indicates the end of conditional blocks in control structures
else	The unhandled condition in case, if and unless expressions	true	Boolean true
elsif	An alternate condition for an if expression	undef	Prevents a class or module from responding to a method call
__ENCODING__	The script encoding of the current file.	unless	Used for unless and modifier unless expressions
end	The end of a syntax block.	until	Creates a loop that executes until the condition is true
ensure	Starts a section of code that is always run when an exception is raised	when	A condition in a case expression

flase	Boolean false	while	Creates a loop that executes while the condition is true
for	A loop that is similar to using the each method	yield	Starts execution of the block sent to the current method
if	Used for if and modifier if expressions	_ _FILE_ _	The path to the current file
module	Creates or opens a module	_ _LINE_ _	The line number of this keyword in the current file
next	Skips the rest of the block		

Primitive Data Types

Strings Within Ruby strings are made up of characters surrounded by either double or single quotations:

Example:

```
puts “Hello my name is Jeff!”

or

puts ‘I love Ruby! Don’t you?’
```

Numbers There are two kinds of numbers within Ruby, integer and floats. Integers are whole numbers while floats allow for decimals:

Example:

```
example_integer = 10

or

example_float = 10.1238
```

Booleans There are two options for boolean, true or false. Normally you receive booleans when comparing values.

Example:

```
ex_string_1 = “hello”
ex_string_2 = “world”
ex_boolean = false

if ex_string_1 == ex_string_2
    ex_boolean = true
```

Arrays In Ruby you are allowed to store multiple types of data types in one string. This would be a perfectly valid array in Ruby:

Example:
ex_array = [“Hi”, ‘My name is’, true, 10.8273, 42, false]

Hashes In Ruby hashes work like they do in other languages. It is a key-value pair system. You use => to assign a value to a key.

Example:
ex_hash = { “Pencil” => 2, “Eraser” => 9, “Backpack” => 5 }

Symbols Symbols are a less memory intensive form of strings. They are indicated by a colon.

Example:
ex_symbols = {:mo => “Movies”, :pop => “Popcorn”}

Syntax of the language

The syntax of Ruby is similar to Perl and Python. The improvements that Ruby made upon Python was the use of blocks that are defined by keywords or braces. Ruby is also similar to Perl but is much more organized compared to Perl. Perl is a very messy and free nature programming language, while Ruby requires well-organized code. Perl also requires you to use a sigil before variables while Ruby does not require this. These improvements made Ruby result in much simpler code that's much easier to read. Unlike either of those languages, Ruby keeps instance variables private to the class which you can only access through accessor methods. Another cool feature of Ruby is that it allows you to do metaprogramming which instead of directly accessing the internal variables of a class, Ruby allows you to pass a message to the class and receive a response due to its private nature.

Control Abstractions of Ruby on Rails

The idea behind control abstractions in programming languages is to simplify repetitive actions. These can be actions that form a pattern such as, listing all the items within an array, adding 2

to the previous number however many times, etc. Ruby has a few control abstractions which I listed and explained below.

Conditionals	Conditionals are used when you would like to compare two objects and based on if the outcome was met or not, have it perform a different action. Example: test = 10 If test == 10 puts “test is equal to 10” elsif test == 11 puts “test is equal to 11” else puts “test is not equal to 10 or 11.” end Using four of the reserved words for Ruby, ‘if’, ‘elsif’, ‘else’ and ‘end’ we are able to run this block of code to test if the ‘test’ object equals any of the values we are looking for. We assign the value 10 check to see if it is equal to 10, print that statement if it does. Elsif signifies there are other options we would like to check. Another option is that ‘test’ equals 11, if it is it will print that it is equal to 11. Otherwise it will move to else and state that it does not equal either numbers. Lastly moving to the end the conditional checks due to the reserve word ‘end’ that signifies the end of the code block.
Loops	Loops come in handy for Ruby when you need to do repetitive tasks and can do it very quickly. Example: for x in 1..6 do puts x end Using the reserved words ‘for’, ‘in’, and ‘end’ we are allowed to quickly have this loop print out “1 2 3 4 5 6”. Which without the loop would have taken six individual lines of code which does not sound bad but if you needed to do this hundred times would be a large waste of memory, resources, and code.
Subroutines	Subroutines are useful because they are Ruby’s versions of functions. You can save yourself from re-using large amounts of code by using functions. Example: def sayUsersName(name) return = name + “ is my name” End name = sayUsersName(“Rachel”)

	puts name

Abstractions in Ruby on Rails

Ruby's core idea is that all data no matter what it is represented by an object which means that Ruby is a highly abstract language. Since Ruby is an Object-Oriented programming language it is naturally set up to do abstractions. Modules within Ruby are a group of methods, classes, and constants. So, when you call a method you are enacting data abstractions and allowing this method to be called like an object. For example, calling the math method sin(x), you are enacting abstraction because you do not know the process of achieving this answer, you just plug in the method. Another way Ruby does abstraction is through classes. Grouping together information about the object and then based on its access control condition decides what information is visible and what is not. And lastly Ruby has access control abstraction, private, protected, and public. The high level of abstraction that is built into the Ruby language allows for the language to do a lot of work in the background while presenting to the coder a very logical and easy-to-read language.

Ruby on Rails Writability, Readability, and Reliability

Writability

Ruby writability is very good, even a beginner to Ruby can achieve programs in a few lines of code. The fact that Ruby turns everything into objects makes it very easy to read, understand, and produce functional code.

Readability

The strong suite of Ruby is its readability, it is easy to read depending on if the coder follows the common conventions of indentation. Many people will say reading Ruby code is like reading a book. However, the downfall of Ruby's readability is that it can take extra passes through the code to get the flow of logic within the code.

Reliability

Ruby's attention to detail and organization makes it a programming language that has good reliability. It highly encourages developers to write tests so you can quickly weed out errors and lessen the time you spend trying to fix basic things.

Ruby on Rails Strengths and Weaknesses

- Strength:
    - **Speed of Development:** The debug and test cycle will be shorter in dynamically typed language, hence the developer does not have to wait for the compiler to test the new changes in the code. Also, it allows scaffolding of complicated applications in a few lines of code.
    - **Open-source software**: The fact that Ruby on Rails has a wide, active community results in a huge number of helpful tools and gems that are available to be part of your project.
    - **Operates Model-View-Controller architecture**: Since web application development is a complicated project and involves different layers of software, following the MVC principle in ruby on rails makes it easier for developers to understand how to develop a web application in a short time. Also, it promotes the extensibility of the project.
    - **General purpose language**: It is broadly applicable across application domains. Twitter, GitHub, Twitch, and Airbnb have been built on Ruby.
- Weaknesses:
    - **Runtime and Performance**: Comparing Ruby on Rails to other web application frameworks such as Node.js shows that RoR has a slow runtime speed, and it could be noticed in huge applications.

- **Lack of flexibility**: Starting a project in RoR requires the developer to handle a lot of conflagration such as database migration, and routing configuration.
- **Cost of Mistake**: Since RoR has more dependent, and coupled components, making wrong decisions in the application will cost you high. It is hard to fix mistakes in high coupling software, and that might impact the performance.

Overview and the Features of Ruby on Rails

The application we decided to make was a web application using Ruby on Rails. We decided that we wanted to make a Movie Review website because we thought this would be a good way to show the basics of Ruby and the Ruby on Rails framework. We used a scaffold to create the base of our project. Scaffolding within ruby on Rails is defined as a quick way to generate major parts of your application by quickly creating models, views, and controllers. This made starting this project much easier because we quickly set up our scaffold using “rails g scaffold Movie title:string, description:text, movie_length:string, director:string, and age rating:string”. This quickly set up our views, models, and controllers. From there we really just had to go in and tell Ruby how we wanted these to interact with one another. For example, users_Id has many movie reviews and movies have many reviews, etc. Another feature that we were thoroughly impressed with that is in the Ruby on Rails framework are gems that go in the gemfile. There is a whole website dedicated to just Ruby gems, the website is https://rubygems.org/. The way we used gems is by adding gems that added authentication, layout, and database functionality. We used devise for our user authentication so when the user signs up it will import the user's email and password into the database and remember their account and encrypt their password. We used bootstrap-sass to enable our website to use the bootstrap framework. Bootstrap is what gives our projects a nice and neat layout and adjusts the size of the application to your window size automatically as well. Another gem file that we used within our project was ‘seed_dump’. This gemfile will allow you to manually push any information into the database using the ‘rake

db:seed:dump’ command in the terminal. If you wish to install all of our gemfiles for our project, you will use the ‘bundle install’ command in the terminal and it will automatically install any and all missing gem files for you. Gem files are very easy to implement, you put the gem line of code such as ‘gem 'bootstrap-sass', '~> 3.4', '>= 3.4.1'’ into your gem file, bundle install the file, and follow the documentation and you are good to go! We will also require that you install a yarn package that is automatically prompted when using the RubyMine IDE, which we highly suggest. Another nice feature of Ruby on Rails is that it will run its own tests on your website, allowing you to catch any basic yet major issues with your website, saving you a lot of time and headaches down the road while also ensuring quality. These features are by far the easiest to implement in the language, because it's done for you!

Experiments

Questions and Answers

Is Ruby as easy to read and write as everyone says?

> Yes it is, Ruby supports both writability and readability. Even though the developer is new to programming, he/she can learn it faster once he understands how Ruby components are working. The only downside that we saw was that once you start getting a larger project the logical flow of the program can get a little meddled. It might take a few passes and looking around to understand how the components work together.

Is Ruby’s built in database easy to work with?

> Yes it is, we were a bit intimidated by the database in the beginning since none of us had any experience with databases. But, once we found the seed_dump gem file it made pushing information into the database a lot easier.

How does learning Ruby on Rails differ from other web development frameworks like Node.js?

Ruby on Rails enables the developer to write a few lines of code compared to Node.js. Also, writing a few commands to migrate the database and start the application in Rails gives a promising start-up in your application, unlike in Node Js, you need to start from scratch So, learning Rails will take less time than learning node js.

Is the Ruby community as large as it is praised to be?

Yes, I feel like there are alot of people using Ruby and who are passionate about Ruby and the Ruby on Rails framework. However, it does feel like it has gone downhill since 2014-2015. When researching questions, extensions to make our project more robust, and overall just general knowledge most of these posts are from 2011-2015. Finding up to date information on Ruby on Rails 6 was tough. A lot of the extensions that we were really excited about using were made with RoR 3 or 4 and were ultimately broken or above our knowledge in this system to make it work. But, we do see why people love Ruby on Rails. It is quick, easy to pick up, and is pretty logical.

Observations

Our observations found that many popular implementations of Ruby on Rails are outdated compared to current Ruby versions, so they are broken and need many workarounds. For example, giving a review using the stars rating feature did not work as expected in Rails version 6. Since Ruby on Rails six deals with javascript files that differ from the rails versions below 6, using stars raty.js requires a particular configuration. Also, Ruby on rails components are dependent on each other, and any wrong decision you make will cost you to change different parts of the project, which delays your submission of the final product. Finally, despite the lack of flexibility above, Ruby on Rails has a built-in database(sqlite3), and the seeds file allows you to populate it with data which provides an easy way to handle the database service and

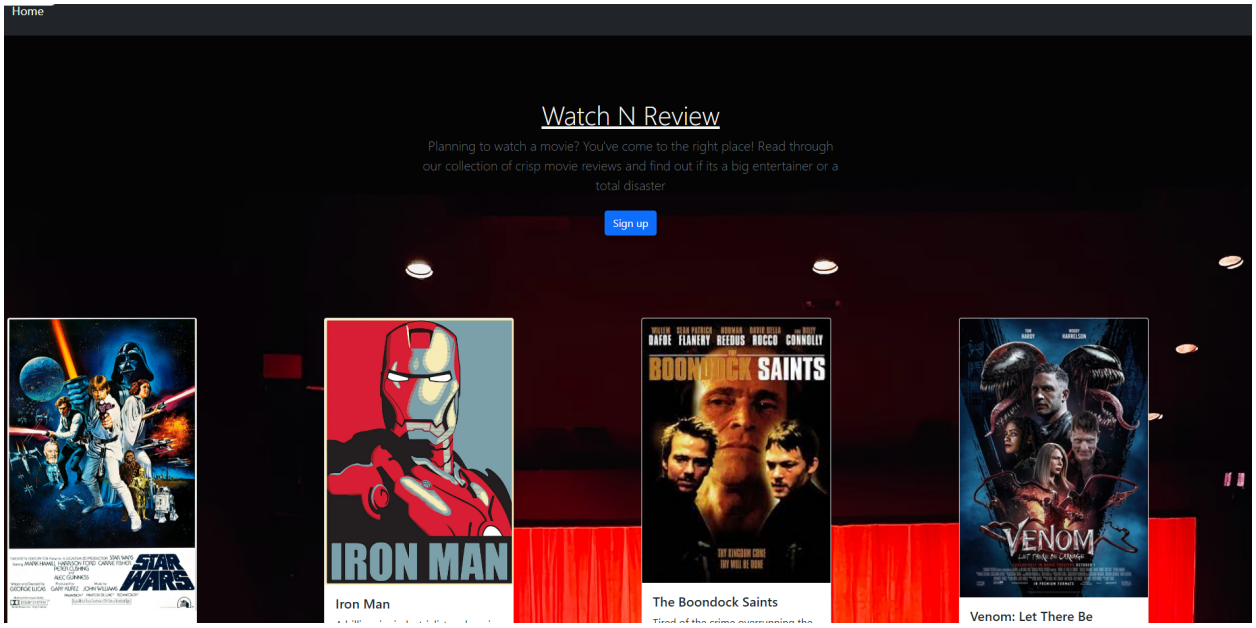
configuration. Besides, using a movie scaffold helps us to have a solid starting point to develop our application.

Program Sample

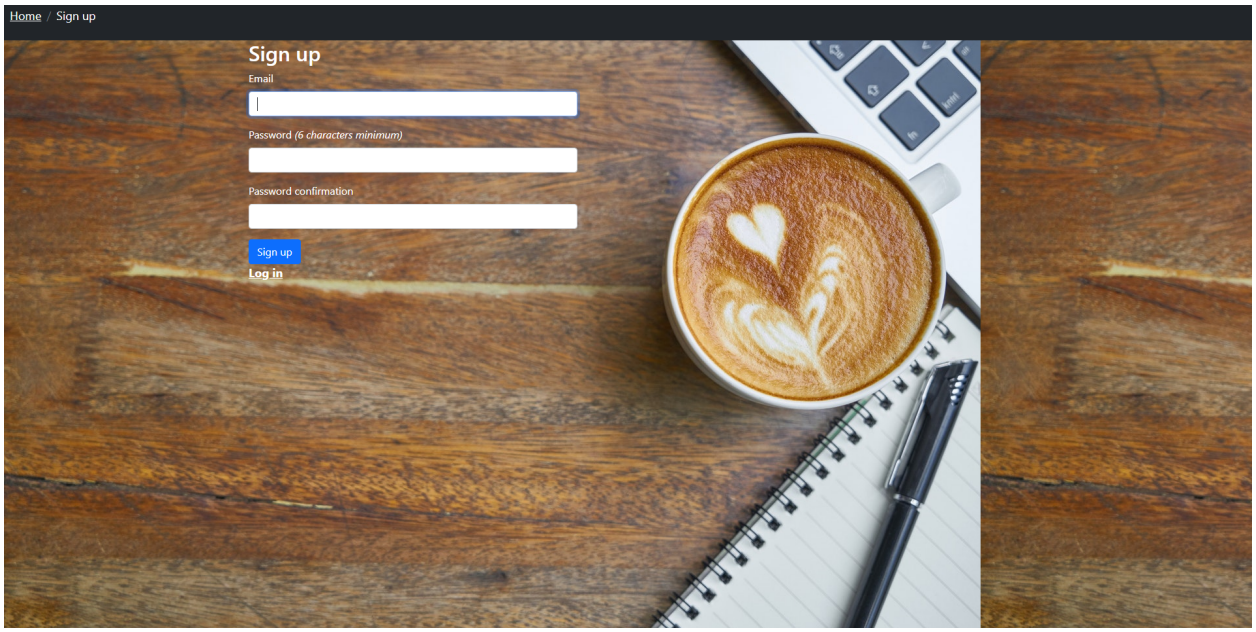
Below are photos of the sections we have developed for the web application. We have decided to make a movie review application called Watch N Review. We have created the following scenes below:

1. Home page
2. Account creation - sign up and login screen
3. Movie selections
4. Movie information and pre-existing reviews
5. Creating movie review and recommendation
6. Add new movies to review
7. Edit movie information

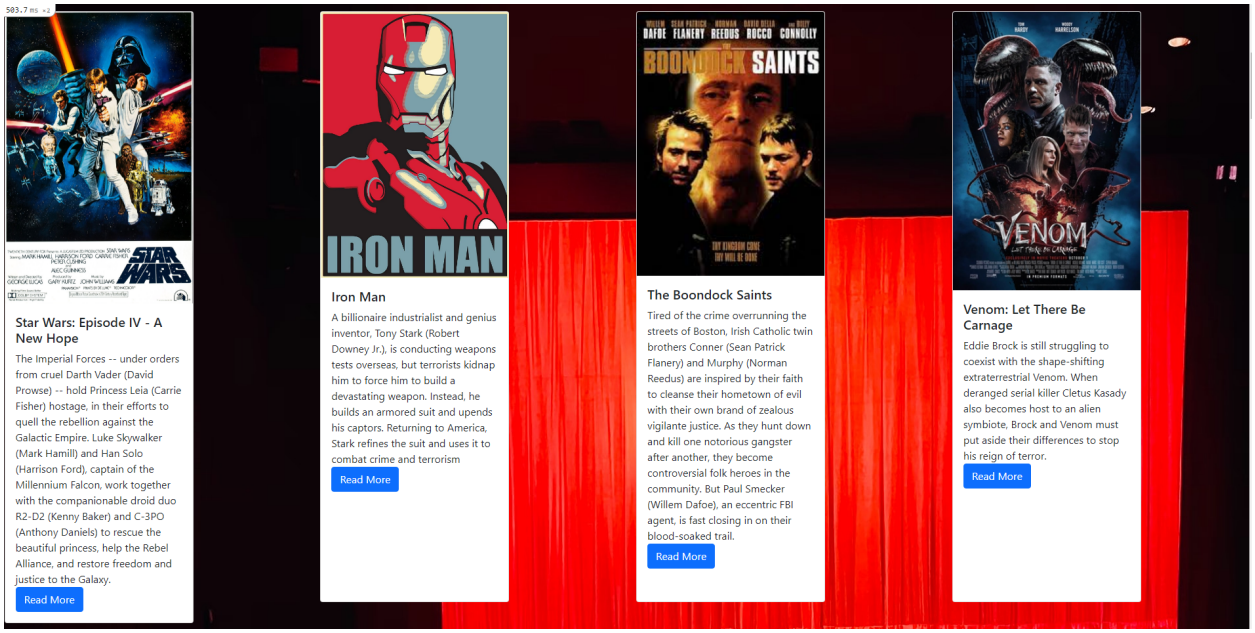
Home Page



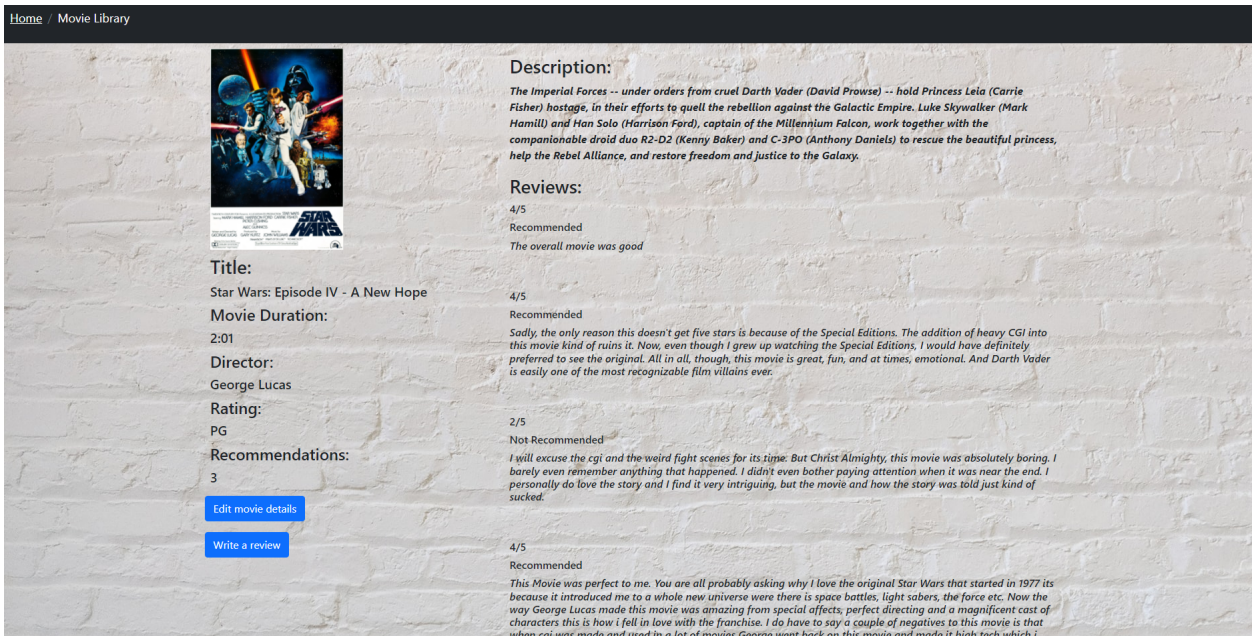
Account creation or login page



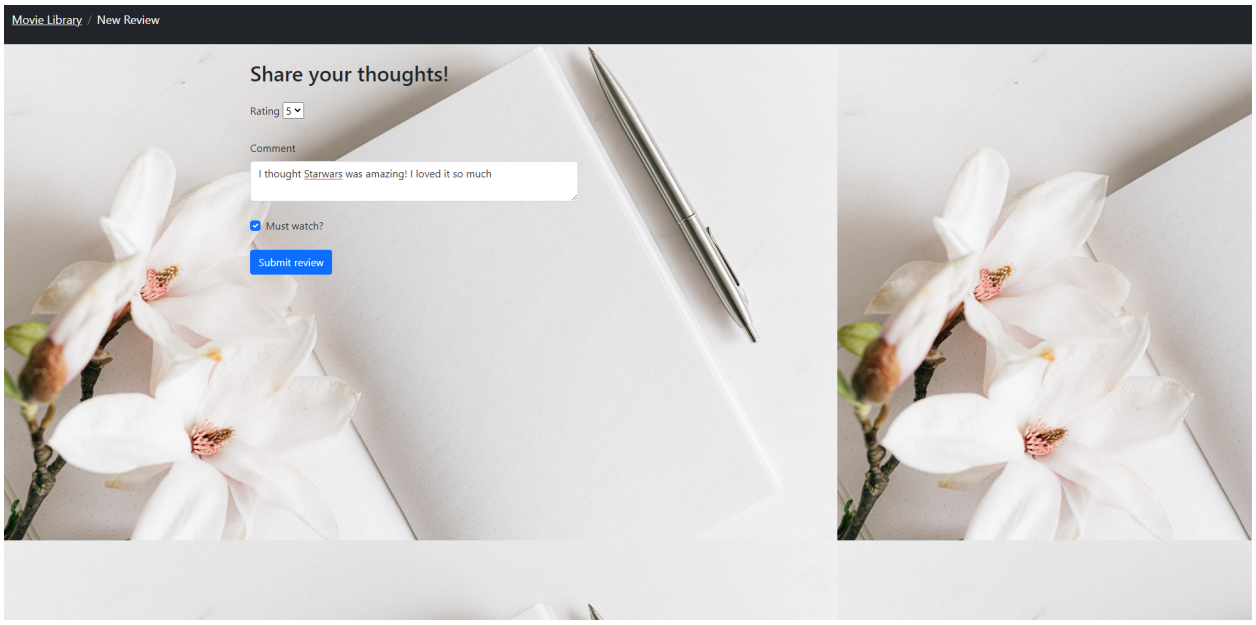
Movie Selections



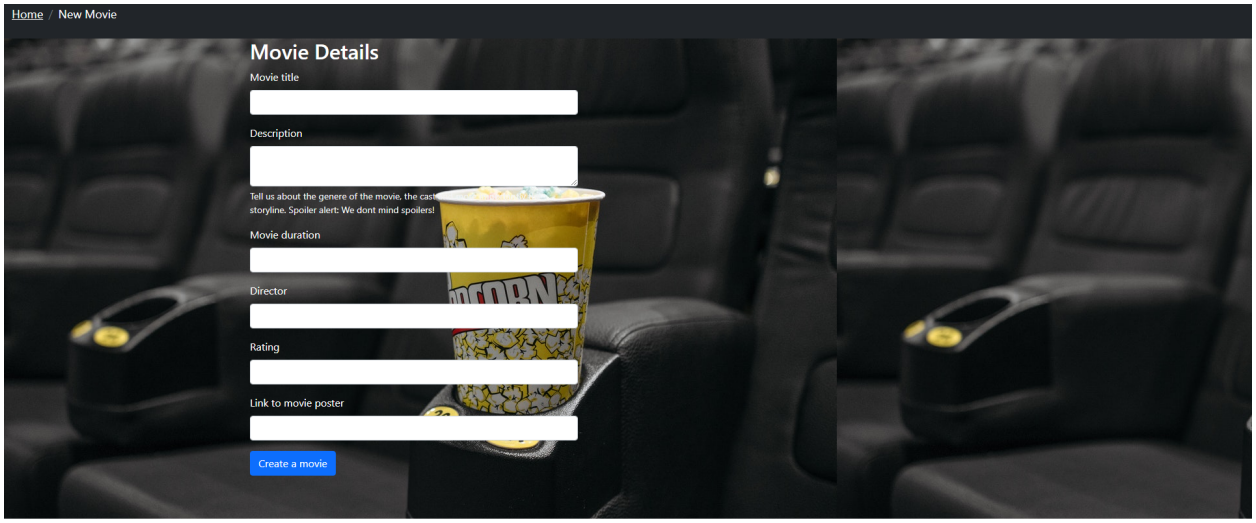
Movie information and pre-existing reviews



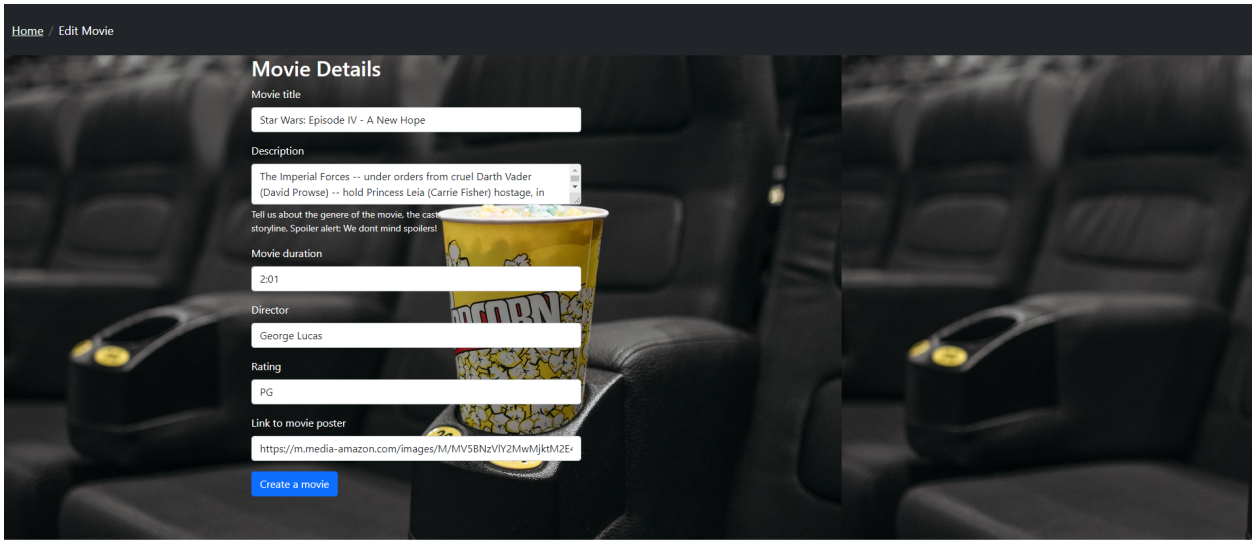
Creating movie review and recommendation



Add new movies to review



Edit movie information



Conclusion

We were honestly not too sure what to expect when we decided to pursue Ruby on Rails as our language. We all admitted that we all had no experience in Ruby, that all of our interests in this project were rooted in web development and that we all knew this would allow us to do something in that area. So, we decided to make a simple movie review application that would allow us to cover the basics of Ruby on Rails and still produce at least a semi-viable project. We learned a lot about Ruby and Ruby on Rails and how quick and easy it is to implement something and how easy it is to expand a project with the help of the community if it works for newer RoB versions.

References

https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/ (strengths and weaknesses)
https://www.netguru.com/blog/pros-cons-ruby-on-rails (strengths and weaknesses)
https://www.educative.io/edpresso/data-types-in-ruby (primitive data types)
https://docs.ruby-lang.org/en/2.2.0/keywords_rdoc.html (ruby's reserved words)
https://sloboda-studio.com/blog/ruby-on-rails-vs-node-js-which-to-choose-for-a-startup/ (Node.js and RoR)