e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

FULL STACK WEB DEVELOPMENT OF REDUX-BASED WEB APPLICATIONS WITH DYNAMIC MICROSERVICES (CASE STUDY - IDEA REPOSITORY)

Nishant Kumar Bharali[*1]

[*1]Author, School Of Electronics Engineering (SENSE) Department, Vellore Institute Of Technology, Vellore, Tamil Nadu, India.

DOI : https://www.doi.org/10.56726/IRJMETS33219

ABSTRACT

This research is aimed towards full-stack development and aimed towards software engineering as a full-stack developer. Full stack developers create new documentations, guides, implement new standards to improve digital agency's web development process. In other words, a modern, professional digital agency cannot develop without full-stack web developers. The project is aimed to develop a full stack web application, that is an API for an idea portal (C.R.U.D. Application) application where a user can add, view, update and delete his/her own ideas as per their requirement with user authentication and authorization for new users and perform C.R.U.D. operations on the required data. Therefore, idea repository will act as a central hub for any user to jot down their ideas and access them as per their requirement. It will contribute to more efficient and reliable deployments. The application is implemented using: Springboot (Java 1.8) for the back-end to create API for C.R.U.D. operations, Reactjs with React-Redux and middleware Redux-saga for the front-end for UI management and data retrieval, and MySQL server/workbench for database management.

Keywords: Full Stack Development, Cloud Computing, Networking Model, Front-End Development, Back-End Development, API.

I. INTRODUCTION

Web development, which roughly refers to the activities involved in creating websites for hosting on intranets or the internet, includes full stack development. It is the creation of an entire application, including the client-side (also known as the front end) and server-side (also known as the back end). There have been significant changes in the field of Full Stack Development and how it has impacted the development as a result of the arrival of cloud computing. Numerous services are included in cloud computing, which has several benefits.

II. METHODOLOGY

This research is to provide classified information about ideas a user keeps in a repository. The website will provide different kinds of operations for the user to add, view, delete, save and edit the ideas stored in the idea repository. The user should register to utilize the site using the registration form. If the user already exists, they can use the Login form to login to the portal and access the main page of the application. Thus, using their email and password, the user can successfully log in to the site and can access their ideas. The user can only view other users' ideas but not any other operation to maintain security. Spring security with the help of JWT token was implemented to implement user authentication and authorization.

Once the user is done with the application usage, they can log out of the application as per their requirement.

Continuous Integration/Continuous Deployment (CI/CD) on Jenkins

At the end, through all testing, we can confirm that the application code had a 100% code coverage for JUNIT Mockito testing for the back-end and JEST testing for the front-end. The project was successfully deployed on the Jenkins server which is a deployment tool for continuous integration and deployment (CI/CD), with the successful builds for both, the client-side and the server -side. The deployment of the application was done through the groovy scripts pipeline method, where we manually entered the instructions using pipeline scripts to proceed with the deployment with successful builds with the tests.

III. MODELING AND ANALYSIS

Problem Description: Case study (Model)

Idea Repository will act as a central hub for any user to quickly jot down any project ideas they have. User authentication and authorization should be available with API to perform C.R.U.D. operations.

Functionalities of this component:

The project is aimed to develop an app where the user will be able to login and have the option to add/delete/edit project ideas.

1. The app provides an existing user to login and a new user to register.

a. Login page –username/password - password check for invalid entry for existing user.

2. Upon successful registration, the new user should be brought back to the login page and the saved username / password (hidden) should be prepopulated.

3. On entering the correct username and password, the user should be taken to the main page of the application.

4. If the user already has some ideas in the database, they should be populated in separate card format.

5. The ideas should be sorted by story points.

6. The card should have the information in this format: Title of the project, Description and story points/estimates.

7. On single click of the card, the user should be able to view it in a bigger size – temp view, if we hover away, this goes off.

8. On double click of the card, of an existing card the user should be presented with a form to edit in a pop-up is presented that contains already saved information which can be edited. A delete button / action will be provided.

9. The user should also have the option to add new cards – clicking '+' on the blank card, the edit form with empty fields must be shown.

10. If the user doesn't have any ideas in the database, then he can be presented with 4 blank cards.

11. User should have the ability to logout of the app as well.

Project build-up Process:

Following steps are followed in a build up of a full-stack application:

Step 1: Define Database schema and design aspects of the application

Step 2: Database management on MySQL Workbench

Step 3: Planning, Design and Development

Step 4: Test Execution - JUNIT Mockito test coverage for the back-end and JEST/enzyme testing for the front-end

Step 5: Maintenance and deployment of the application on a server (jenkins here) using CI/CD.

Define the scope of the application:

The scope of the web application is the area of your Application where the development will occur and matter. Following points help determine scope:

- The features that are important for the business
- Scenarios which have a large amount of data
- Common functionalities across applications
- Technical feasibility
- Limitation on the reusability of business components
- The complexity of test cases
- Least amount of bugs and glitches in the web-application

Analysis

Planning, Design, and Development:

During this phase, you create a development strategy & plan, which contains the following details -

- Technology tools selected
- Framework design and its features
- In-Scope and Out-of-scope items of development
- Development code-base and environment preparation

- Schedule and Timeline of scripting and execution
- Deliverables of unit and integration testing

Proposed Work:

This is a collection of REST APIs and Service automated tests, with Jenkins based integration for simple and easy execution of integration tests. The APIs are for managing the various service requests, all these service run independent of the data sources that may utilize them. They are independent of all external entities.

JWTs are unique among web tokens in that they include a list of claims. Information is sent between two parties through claims. The specific use case will determine what these assertions are. A claim might specify, for instance, who issued the token, how long it is valid, or what authorizations the client has received.

JWT, or JSON Web Token, is an open standard that allows a client and a server to exchange security-related data. Every JWT has a set of encoded JSON objects, including claims. To ensure that the claims cannot be changed after the token is issued, JWTs are signed using a cryptographic technique.

Dots (.) are used to divide the three parts of a string known as a JWT, which is serialised using base64. In compact serialisation, the most common serialisation type, the JWT looks like this: xxxxx.yyyyy.zzzzz. Two JSON strings are what you'll get after decoding:

A. The signature

B. The header and payload.

The JOSE (JSON Object Signature and Encryption) header specifies the type of token, in this case a JWT, and the signing method.

The payload contains the claims. This is displayed as a JSON string, which normally only comprises a dozen fields, in order to keep the JWT minimal. This information is frequently used by the server to verify that the user is authorized to do the requested activity.

The token can't have been changed, thanks to the signature. With a secret that is known to both the issuer and the receiver, or with a private key that is known only to the sender, the party that creates the JWT signs the header and payload. The receiving party verifies that the header and payload match the signature when the token is used. The receiving party checks that the header and payload match the signature before using the token. Although any overlaying standards may require claims, there are no mandatory claims for a JWT. In OAuth2.0, for instance, using JWT as the bearer access token necessitates the use of iss, sub, aud, and exp. Compared to others, some are more typical.

OAuth bearer tokens are a common use for JWTs. An authorization server creates a JWT at a client's request and signs it so that no one else can change it in this example. Following that, the client will send this JWT along with its REST API request. To determine whether the JWT is legitimate, the REST API will compare the JWT's signature to its payload and header. The claims can be used to grant or deny the client's request once the REST API has verified the JWT.

A JWT bearer token can be thought of as an identity badge that allows you to enter a secured building. The claims state that the badge comes with special permissions; that is, it might only allow access to some parts of the building. In this analogy, the reception desk is the authorization server, or the badge issuer. Additionally, the company logo, which is analogous to the JWT's signature, is printed on the badge to ensure its legitimacy. Similar to the claims in a JWT, the badge's permissions determine whether or not the holder can enter a restricted area when they attempt to do so.

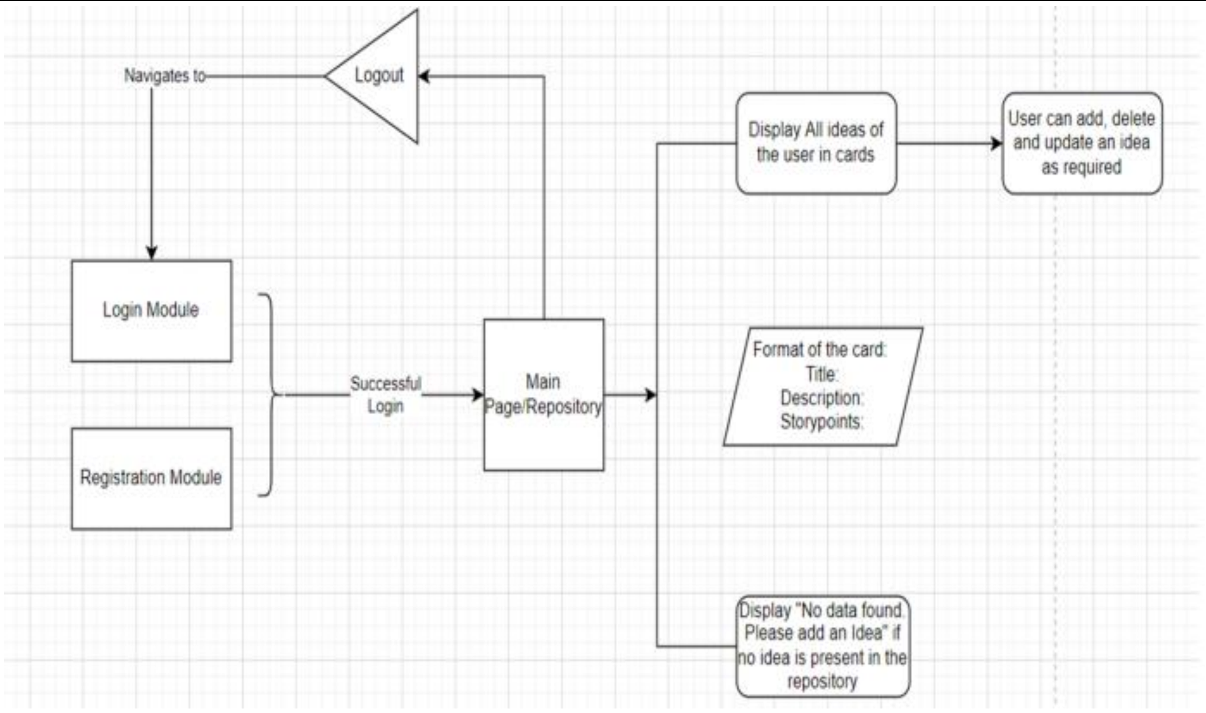

Navigates to
Logout
Display All ideas of the user in cards
User can add, delete and update an idea as required
Login Module
Successful Login
Main Page/Repository
Format of the card: Title: Description: Storypoints:
Registration Module
Display "No data found. Please add an Idea" if no idea is present in the repository

Figure 1: Research Layout.

The project features a login page, a registration page and the main page of the application.

On eclipse, is our backend managed, where 5 significant folders are maintained:

1. Models - Users and Ideas model classes
2. Controllers - Rest Controller -REST APIs / endpoints
3. Services
4. Security
5. Repositories
6. Testing

We have configured the same in our jenkins file too which has a separate file in the project.

All the services that we will be running through Jenkins are configured in this jenkins file only using the exact same name for the service that we will be using there.

IV. RESULTS AND DISCUSSION

4.1. Discussion of Software Tools and Fundamental prerequisites required:

1. Frontend:

1.1. Reactjs:

An open-source JavaScript library that is used to create a user interface (UI) for web applications is known as React.js or React in general. On the other hand, cross-platform mobile applications can be built with React Native. Similar to React, the underlying framework makes use of native mobile components rather than web components.

The V layer of the model-view-controller (MVC) software development pattern can be referred to as React. The MVC approach divides your application into three parts. The model handles data storage and retrieval in a database. The controller is in charge of receiving user input and passing it on to the model, while the view is in charge of displaying the user interface. You can only use the view component with React; For the model and controller sections, you will need to use different tools.

React.js should be preferred to an Angular-like framework because:

- Data binding at one level.
- It works well with user interfaces based on components.

- Building blocks that are very adaptable.
- Isomorphic JavaScript.
- It aids in the creation of enormous applications.
- Everything can be seen and controlled from a single location.
- The community lends a lot of support.

React is now more popular than any other front-end development framework.

How so?

- Development of dynamic applications is simple: In contrast to JavaScript, where coding frequently becomes very complex very quickly, React makes it easier to create dynamic web applications because it requires less coding and offers more functionality.
- Increased efficiency: Utilizing Virtual DOM, React speeds up the development of web applications. Instead of updating all of the components once more, as conventional web applications do, Virtual DOM compares the components' previous states and updates only the changed items in the Real DOM.
- Parts that can be reused: Any React application is made up of components, and a single app typically has multiple components. Reusing these components throughout the application significantly shortens the development time of the application because they retain their logic and controls.
- Data movement in one direction: Data flows in one direction through React. As a result, developers frequently nest child components within parent components when creating a React app. Debugging errors and pinpointing the exact location of a problem in an application becomes easier when data flows in a single direction.
- Very little learning curve: Because it mainly combines fundamental HTML and JavaScript concepts with a few useful additions, React is simple to learn. However, just like with other frameworks and tools, learning how to use React's library effectively takes time.
- Dedicated tools that make it easy to debug: A Chrome extension that can be used to debug React applications has been released by Facebook.

The React library's popularity and adoption by a large number of businesses are more than justified by the aforementioned reasons. Let's get to know the features of React now.

1.2. React-Redux:

Redux is a JavaScript state management library that is freely available.

The official React binding for Redux is React Redux. React components are able to read data from a Redux Store and send Actions to the Store to update the data thanks to this feature. By providing a sensible method for managing state through a unidirectional data flow model, Redux aids apps in scaling. Conceptually, React Redux is simple. It re-renders your component, checks to see if the data it needs has changed, and subscribes to the Redux store.

Flux served as a model for Redux. Redux looked at the Flux architecture and cut out any unnecessary complexity.

- There is no Dispatcher concept in Redux.
- Store will directly receive and handle the Action objects.
- While Flux has many Stores, Redux only has one

1.3. Middleware, Redux-Saga:

A Redux store can use the middleware library Redux Saga to asynchronously interact with resources outside of itself. Accessing browser storage, carrying out I/O operations, and making HTTP requests to external services are all examples of this. These procedures are also referred to as side effects. Redux Saga aids in the organization of these side effects in a manner that is simpler to control.

Natively, a redux store can only dispatch actions and update its state with the help of its root reducer. An action is a description of an event that takes place in your application and an intention to alter its state. Values derived from dispatched actions or accumulated by a reducer are incorporated into the newly updated state of your application.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

Because it is necessary to enable useful Redux features like time travel (replaying past actions), reducers must be written as a pure function. Actions are just things that are passed to the reducer and are by definition deterministic. Thus, we face a challenge; Your side effects cannot be included in your Redux application in any way.

An action and a reducer are separated by a Redux middleware. As long as the middleware intercepts this, executes its logic, and returns a plain object for the reducer, this makes it possible for actions to contain something other than a plain object.

Functions can be passed into the Redux store dispatch using Redux Thunk, a popular alternative to Redux Saga. The dispatch checks to see if the function is a function or an action object before either executing the function or passing the action object directly to the reducer. After that, these functions can execute any complicated asynchronous logic they want and create a straightforward action object that can be sent to the reducer.

The fact that a separate set of actions is defined in your Redux application and is only captured by watcher functions (as part of your saga) makes Redux Sagas slightly different. The saga will dispatch the resulting action to the reducer of your application after capturing the action and carrying out the necessary logic. The saga is essentially a separate thread from your application. It performs complex asynchronous tasks by listening for specific actions from your main application and updating your application's state when they are finished.

2. Backend:

2.1. Java:

Java is a computing platform and programming language that was first made available by Sun Microsystems in 1995.

Java is a class-based, object-oriented, general-purpose programming language made to have fewer implementation dependencies. For the development of applications, it is the go-to choice as a computing platform for most. As a result, Java is fast, safe, and dependable. In laptops, data centers, game consoles, scientific supercomputers, mobile phones, and other devices, it is frequently utilized for the development of Java applications. Java is a programming language, compiler, core libraries, and run-time (Java virtual machine) that is defined by a specification. The Java run-time lets software developers write code in languages other than Java that still runs on the Java virtual machine. The Java virtual machine and the Java core libraries are typically linked to the Java platform.

The following properties were included in the design of the Java language:

➢ Independent of the platform: Java programs don't directly access the operating system; instead, they use the Java virtual machine as an abstraction. Java programs are thus extremely portable. All supported platforms, such as Windows or Linux, can run a Java program that adheres to certain rules and is standard-compliant.

➢ Programming language with an emphasis on objects (OOPs): In Java, all elements are objects, with the exception of the primitive data types.

➢ Control of memory automatically: When creating new objects, Java manages the allocation and de-allocation of memory. Objects without an active pointer are automatically deleted by the so-called garbage collector.

2.2. JSON:

JSON, or JavaScript Object Notation, is a simple format for transferring data. Humans can read and write with ease. Machines can easily generate and parse it. JSON is a text format that uses conventions that are familiar to programmers of the C-family of languages, including C, C#, C++, JavaScript, Java, and many others. However, JSON is completely independent of any particular language. JSON is a great language for transferring data because of these properties.

The two structures that make up JSON are:

• A group of name-and-value pairs. This is implemented as an object, record, struct, dictionary, hash table, keyed list, or associative array in various languages.

• A list of values in order. This is represented as an array, vector, list, or sequence in the majority of languages.

Universal data structures like these exist. They are supported in one way or another by virtually all modern programming languages. It makes sense that these structures should also serve as the foundation for a data format that is compatible with programming languages.

They take on the following forms in JSON:

An unsorted collection of name-value pairs is an object.

An object has a left brace at its beginning and a right brace at its end.

The following follows each name: comma (,) separates the name/value pairs from the colon (:).

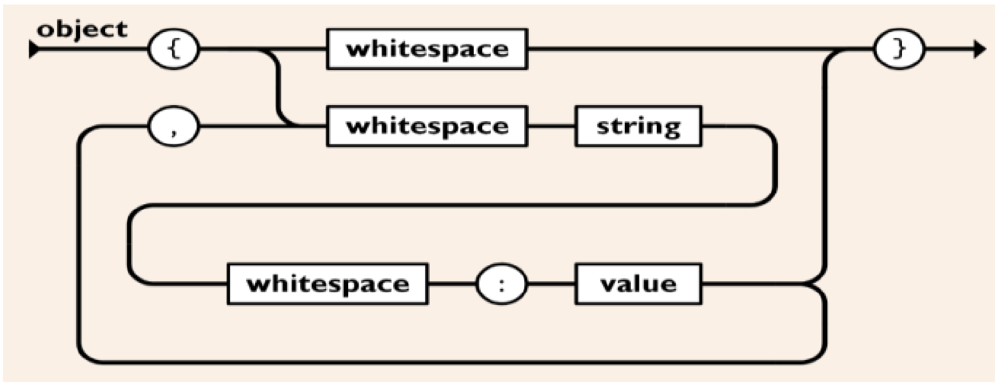


Figure 2: A Figure of JSON Object

2.3. REST API:

An application programming interface (API) or web API that complies with the constraints of the REST architectural style and permits interaction with RESTful web services is referred to as a REST API (also referred to as a RESTful API). Roy Fielding, a computer scientist, came up with the acronym REST, which stands for representational state transfer. REST is not a protocol or a standard; rather, it is a set of architectural constraints. REST can be implemented in a variety of ways by API developers.

A representation of the state of the resource is transferred to the requester or endpoint when a client request is made via a RESTful API. Through HTTP, this representation, or information, is made available in one of several formats: HTML, JSON, XML, Python, PHP, or simply text. Despite its name, JSON is the most widely used file format because it can be read by both humans and machines and is language-independent.

Another thing to keep in mind: Headers and parameters are also crucial to the HTTP methods of a RESTful API HTTP request because they contain crucial identifier information regarding the metadata, authorization, uniform resource identifier (URI), caching, cookies, and other aspects of the request. There are request and response headers, each with its own status codes and information about the HTTP connection.

An API must meet these requirements in order to be considered RESTful:

➢ A client-server architecture that uses HTTP to manage requests and consists of clients, servers, and resources.

➢ Stateless client-server communication means that no client data is stored between get requests and that each request is independent and unrelated.

➢ Data that can be cached, which makes client-server interactions easier.

➢ Optional code-on-demand: the capacity to provide the client with additional functionality by sending executable code from the server to the client upon request.

2.4. HTTP Methods:

The fundamental API interactions that we observe on a daily basis are driven by HTTP methods. A crucial step toward a much deeper appreciation of the API space is comprehending these methods, their idempotency, and their safety. To indicate the desired action for a given resource, HTTP defines a set of request methods. These request methods are sometimes referred to as HTTP verbs, despite the fact that they can also be nouns. Although each implements a distinct semantic, a number of them share some features: For instance, a request method can be cacheable, safe, or idempotent.

GET

The GET method makes a request for an image of the given resource. The only thing GET requests should do is retrieve data.

HEAD

The HEAD method requests a response in the same way that a GET request does, but without the body.

POST

The POST method sends an entity to the specified resource, frequently resulting in state changes or server side effects.

PUT

The PUT method uses the request payload to replace all existing representations of the target resource.

DELETE

He specified resource is deleted using the DELETE method.

TRACE

Along the way to the target resource, the TRACE method conducts a message loop-back test.

PATCH

A resource is modified in part using the PATCH method.

2.5. Web services:

A set of open protocols and standards known as a web service makes it possible for data to be transferred between various applications or systems. In a manner analogous to inter-process communication on a single computer, software programs written in a variety of programming languages and operating on a variety of platforms can use web services for data communication/exchange.

The client that invoked the web service would receive functionality from the web service.

2.6. Micro Services:

Software is made up of small, independent services that communicate over clearly defined APIs in a microservices architecture and organizational approach. Each of these services is owned by a small group of individuals.

Applications built with microservices architectures are quicker to develop and easier to scale, facilitating innovation and shortening the time to market for new features.

An application is built using a microservices architecture, in which each application process is run as a service by independent components. Utilizing lightweight APIs, these services communicate through a clearly defined interface. Each service has a single purpose and is designed with business capabilities in mind. Each service can be updated, deployed, and scaled to meet demand for particular application functions because they operate independently.

A microservices architecture allows for the development, deployment, operation, and scaling of each component service without affecting the operation of other services. There is no requirement for services to share any of their implementation or code with other services. APIs that are clearly defined are the means by which each component communicates with each other.

Each service focuses on solving a specific issue and is made for a particular set of capabilities. It is possible to divide a service into smaller ones if developers add more code to it over time and it becomes more complicated.

With microservices, each service can be scaled independently to meet application feature demand. Teams are able to appropriately size infrastructure requirements, precisely measure the cost of a feature, and maintain availability in the event of a spike in demand thanks to this.

Continuous integration and delivery are made possible by microservices, making it simple to test new ideas and roll back if they don't work. Experimentation is made possible, code updates are made simpler, and the time to market for new features is sped up thanks to the low cost of failure.

The architectures of microservices do not adhere to a "one size fits all" policy. Teams are free to select the best method for resolving their particular issues. As a result, teams working on microservices can select the most appropriate tool for each task.

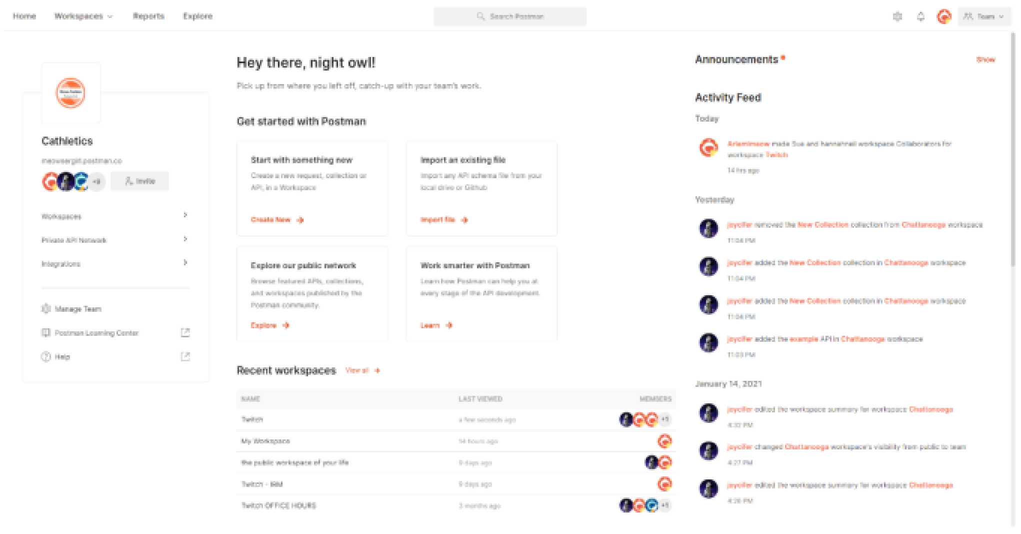
An application's resistance to failure is increased when it is service independent. If one component fails in a monolithic architecture, the application as a whole may also fail. Applications handle total service failure with microservices by reducing functionality rather than crashing the application as a whole.

2.7. Postman Software:

Postman is a development tool for APIs, or application programming interfaces, that makes it easier to create, test, and modify APIs. Postman is an API testing tool that is scalable and can be quickly integrated into the CI/CD pipeline. An Indian engineer started it in 2012 to make the testing for APIs easier. Application Programming Interface (API) calls enable software applications to communicate with one another. This tool contains nearly every feature that a developer could possibly require. It makes API development simple and easy for over 5 million developers every month. It can make different kinds of HTTP requests (GET, POST, PUT, and PATCH), save environments for later use, and convert the API into code that can be used in different languages (like JavaScript).

Postman Software is now preferred by more than 4 million people for the following reasons:

1. Accessibility
2. Utilization of Collections – Postman permits collection creation for Postman API calls. Multiple requests and sub-folders can be created by each collection. Organization is quicker this way.
3. Collaboration – Importing/exporting collection and environments makes sharing files simple. Collections can also be shared via direct link.
4. Developing Environments
5. Creation of Tests – Each Postman API call can include test checkpoints that help ensure test coverage, such as checking for a successful HTTP response status.
6. Automation Testing – Tests can be run in multiple iterations using the Collection Runner or Newman, saving time for repetitive tests.
7. Debugging – The Postman console makes it simple to debug tests by letting you see what data has been retrieved.
8. Continuous Integration – Because it can facilitate continuous integration, development practices are upheld.



Hey there, night owl!

Pick up from where you left off, catch-up with your team's work.

Get started with Postman

Start with something new — Create a new request, collection or API, in a Workspace

Import an existing file — Import any API schema file from your local drive or Github

Explore our public network — Browse featured APIs, collections, and workspaces published by the Postman community.

Work smarter with Postman — Learn how Postman can help you at every stage of the API development.

Recent workspaces

NAME	LAST VIEWED
Twitch	a few seconds ago
My Workspace	14 hours ago
the public workspace of your life	9 days ago
Twitch - BIM	9 days ago
Twitch OFFICE HOURS	3 months ago

The Postman v8 desktop app Home screen

Figure 3: Postman Software Layout

2.8. Spring Framework and Spring boot:

Spring Framework is used by millions of developers worldwide to create code that is reusable, easy to test, and has high performance.

Java platform Spring Framework is open source. It was first written by Rod Johnson and released in June 2003 under the Apache 2.0 license. Spring is thin in terms of transparency and size.

Any Java application can use the Spring Framework's core features, but there are extensions for building web applications on top of the Java EE platform. By enabling a POJO-based programming model, the Spring framework aims to simplify J2EE development and encourage good programming practices.

A project called Spring Boot is based on the Spring Framework. It makes setting up, configuring, and running both web-based and simple applications faster and easier.

It is a Spring module that gives the Spring Framework the RAD (rapid application development) feature. Because it requires little Spring configuration, it is used to create a standalone Spring-based application that you can simply run.

In short, Spring Boot is a combination of Embedded Servers and the Spring Framework.

XML configuration (deployment descriptor) is optional in Spring Boot. The software design paradigm employs convention rather than configuration, reducing developer effort.

Spring Boot Java applications can be created with either the Spring STS IDE or Spring Initializr.

Spring Boot Framework ought to be utilized because:

- Spring Boot employs the approach of dependency injection.
- It has powerful transaction management capabilities for databases.
- It makes it easier to integrate with other Java frameworks like Struts and JPA/Hibernate ORM.
- The application's cost and time to develop are both reduced as a result.

Numerous other Spring sister projects, in addition to the Spring Boot Framework, aid in the development of applications that meet the requirements of contemporary businesses. The following are some sister projects for Spring:

- Spring Data
- Spring Batch
- Spring Security: It is a security framework that provides robust security to applications.
- Spring Social: It supports integration with social networking like LinkedIn.
- Spring Integration

2.9. MySQL Server & Database:

Oracle developed MySQL, a relational database management system (RDBMS) based on structured query language (SQL).

In a way of an example, a database could be anything from a straightforward shopping list to a picture gallery or a location to store the vast amounts of data in a business network. A digital store that stores and organizes data according to the relational model is known as a relational database. Rows and columns make up this model's tables, and the relationships between data elements all adhere to a strict logical structure. The set of software tools used to actually implement, manage, and query such a database is called an RDBMS.

For building and maintaining everything from powerful, data-driven B2B services to customer-facing web applications, MySQL is an essential component of many of the most widely used software stacks. MySQL backends are used by internet-critical organizations like Facebook, Flickr, Twitter, Wikipedia, and YouTube due to its open-source nature, stability, and extensive feature set, as well as Oracle's ongoing development and support.

The underlying technology:

Docker is based on a technology written in the Go programming language that makes use of a number of Linux kernel features to provide its functionality. The isolated workspace known as the container is provided by Docker through the use of a technology known as namespaces. Docker creates a set of namespaces for each container that you run. A layer of isolation is provided by these namespaces. Access to that namespace is prohibited.

2.10. GIT:

Git is a Distributed Version Control System (VCS) that was created in 2005 by Linus Torvalds, the Linux creator. It is open source, which means that it can be used for free. Currently, it is the version control tool that is used the most.

Version control systems: Are software tools that assist a software team in managing source code changes over time. A particular kind of database is used by version control software to track each code change. Developers can use rollback or revert to go back in time and compare earlier versions of the code in the event of a mistake, assisting in the correction of the error and minimizing disruption to all team members.

2.11. Jenkins:

Jenkins is a server for continuous integration (CI) that works with a lot of different technologies and tools. By adopting a Continuous Integration (CI) process, you can guarantee that every developer's code is regularly merged into a shared trunk. The product is reconstructed and tested automatically when a change is committed to the repository.

You can automate a variety of day-to-day tasks with Jenkins, including building, code analysis, checking out the sources from source control, and performing various levels of testing and deployment.

You can detect defects much more quickly by setting up Jenkins to run these tasks each time a developer changes the source code, thereby maintaining the quality of your applications and reducing time to market.

Jenkins is very customizable, and there are a lot of plugins that let you use tools like source control, the shell, and batch scripts. Jenkins supports Java, works on a variety of platforms, and connects to other corporate systems.

➢ Using Projects and Pipelines:

Pipelines and simple projects are the two types of commands that can be specified in Jenkins.

For task management, earlier versions of Jenkins only offered projects (formerly known as jobs). Projects must be manually created and configured in the UI, and if you need to create multiple projects, additional maintenance work may be required. Project configuration is independent of your code.

Pipelines, in which all build, test, analysis, and deployment tasks are stored in a single pipeline and saved as a Jenkinsfile, are also supported by more recent versions of Jenkins. Because they are suitable for organizing complex activities that run on multiple machines, Micro Focus suggests using pipelines. You can also save a pipeline in your source control code system with the rest of your code as an additional artifact and follow common CI best practices by storing it in a file format.

The examples in this guide are created using projects rather than pipelines for simplicity's sake.

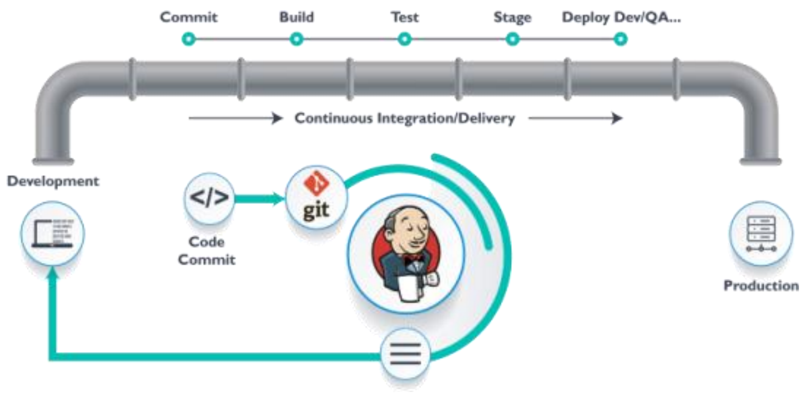


Figure 4: Jenkins workflow

4.2. Approach for practical experiment for a User:

➢ **Developed using:**

* React

* Semantic UI - A development framework that helps create good-looking and responsive layouts using human-friendly HTML.

* Maven - Dependency Management

* ES6 – The next version of JavaScript. It allows us to write real JavaScript classes.

* JSX – Allows us to place HTML in JavaScript without concatenating strings.

* Webpack – Allows us to pack all of our JavaScript files into one single bundle

* Springboot - RESTful Spring Boot Microservices with Spring Data JPA (Spring Data REST)

* Hibernate - JPA provider

* MySQL - Database

* JWT – For token based user authentication and authorization.

➢ **Prerequisites:**

* Install node

* Install npm

* Install Java (jdk 1.8 is preferred)

* Install Maven 3.x

* Install Eclipse or Spring Tool Suite

* Install Visual Studio Code

➢ Notes:

* This React app is named "experiment" using create-react-app. In the case of create-react-app, you can either install it globally (i.e. $ npm install -g create-react-app) or install it locally (i.e. $ npm install create-react-app). If you choose the latter, you will have to specify its path whenever you invoke it.

(e.g. path/to/node_modules/.bin/create-react-app).

* Check Installed Versions

```
Node: $ node --version
v16.14.2
NPM: $ npm  --version
8.5.0
Create React App: $ create-react-app  --version
5.0.0
```

➢ **Quick Development Setup (for the Case Study):**

* git clone https://github.com/NishantBharali/projects

* Import as maven project in eclipse or sts. Run springboot application on embedded Tomcat server localhost:8090.

* Create database:

Create **experiment** named database on MySQL on localhost:3306.

Then set _**spring.jpa.hibernate.ddl-auto = create**_ in src/main/resources/application.properties in the springboot application.

Once the database tables are created, set it to the value **update**

* Run npm install and then npm start in VSCode command line

* Hit http://localhost:3000/

* You can now view experiment in the browser proxying request /login from http://localhost:3000 to http://localhost:8080/

➢ **Versions:**

* Node: 16.14.2

* NPM: 8.5.0

* Create React App: 5.0.0

* Spring Boot: 2.6.4 RELEASE

* React: 17.0.2

* Maven: 3.8.5

➢ **Deployment:**

* API

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

Run mvn clean install to generate a WAR file deployable in any servlet containers like Tomcat.

* UI

Run npm run build

It builds the app for production to the build folder.

It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.

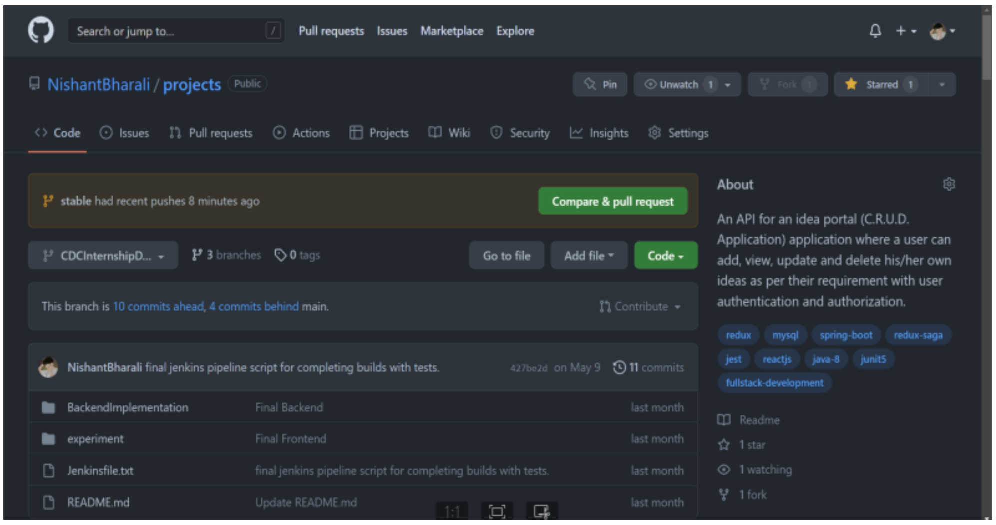
The application will be successfully deployed after following the provided steps.

To initiate the frontend: npm start

It starts the development server in the browser.

4.3. Research Result, Demonstration and Output

4.3.1. GitHub Repository



NishantBharali / projects Public

stable had recent pushes 8 minutes ago

CDCInternshipD... 3 branches 0 tags

This branch is 10 commits ahead, 4 commits behind main.

NishantBharali final jenkins pipeline script for completing builds with tests. 427be2d on May 9 11 commits

BackendImplementation	Final Backend	last month
experiment	Final Frontend	last month
Jenkinsfile.txt	final jenkins pipeline script for completing builds with tests.	last month
README.md	Update README.md	last month

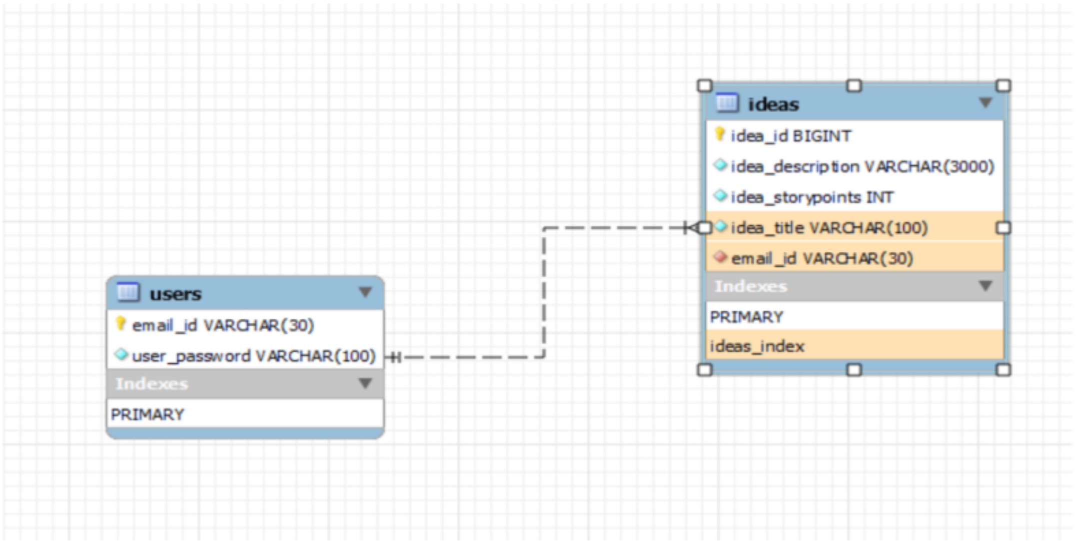
About

An API for an idea portal (C.R.U.D. Application) application where a user can add, view, update and delete his/her own ideas as per their requirement with user authentication and authorization.

redux mysql spring-boot redux-saga jest reactjs java-8 junit5 fullstack-development

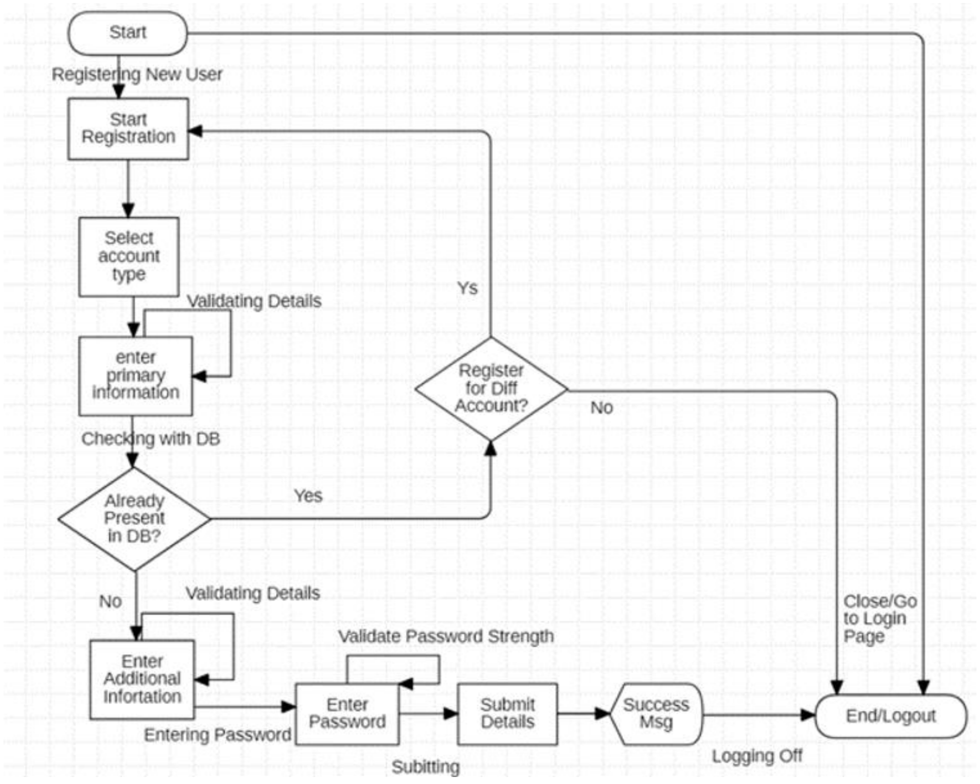
Readme
1 star
1 watching
1 fork

4.1.2. Database Schema



e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

4.1.3. User Login Module Flow



4.1.4. Creating the request body for the endpoints.

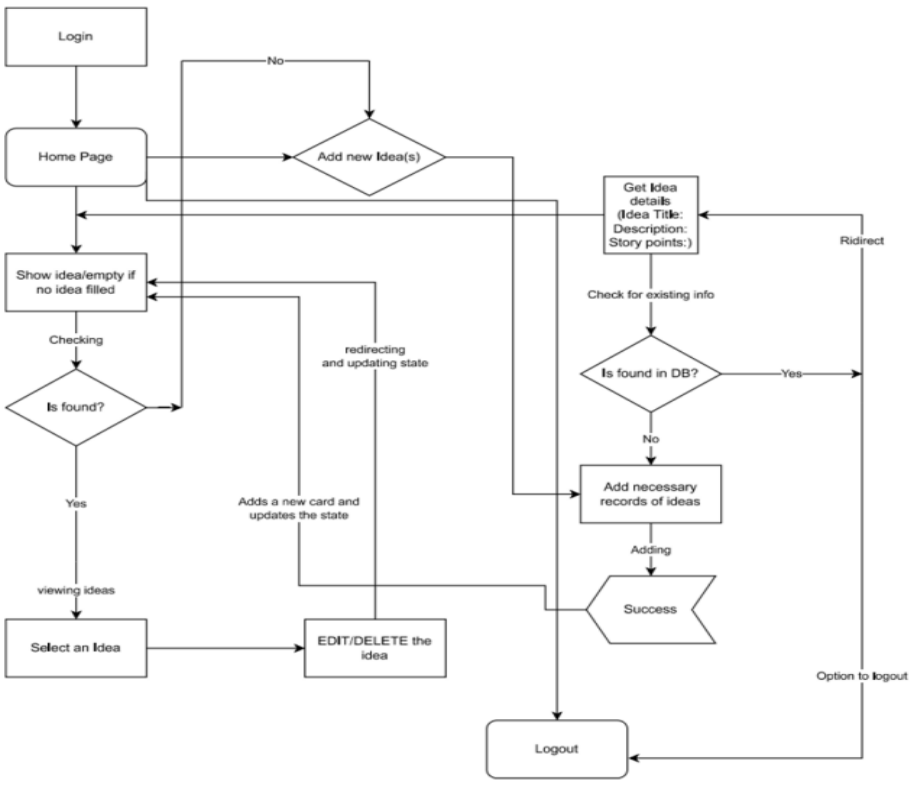
The request bodies don't have any manual value.

All the variables are dynamic.

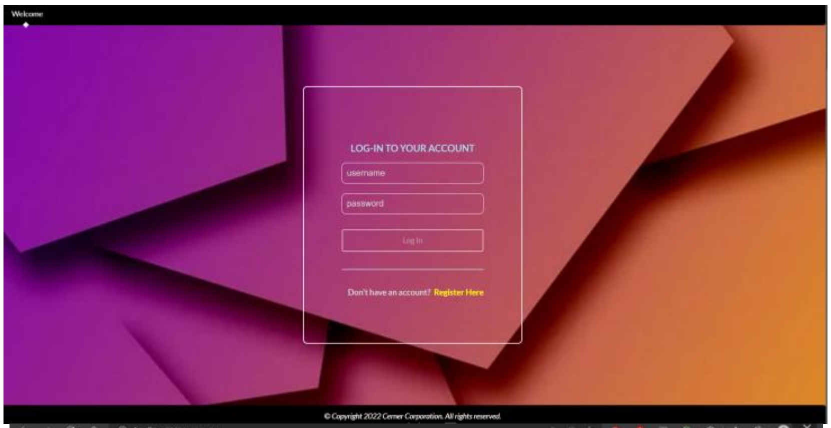
The variables are accessing their values from the environment variables.

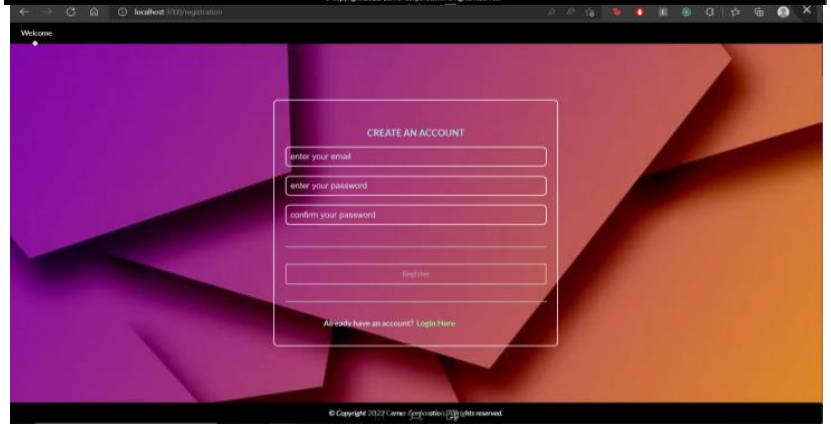
For every individual service, individual and dedicated enviroment varibles are created.

4.1.5. Main Page Workflow

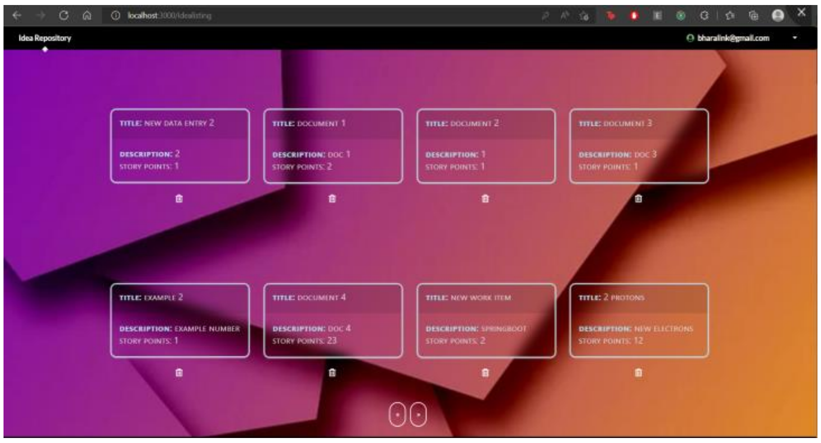

Login
No
Home Page
Add new Idea(s)
Get Idea details (Idea Title: Description: Story points:)
Ridirect
Show idea/empty if no idea filled
Check for existing info
Checking
redirecting and updating state
Is found?
Is found in DB?
Yes
No
Add necessary records of ideas
Yes
Adds a new card and updates the state
Adding
viewing ideas
Success
Select an Idea
EDIT/DELETE the idea
Option to logout
Logout

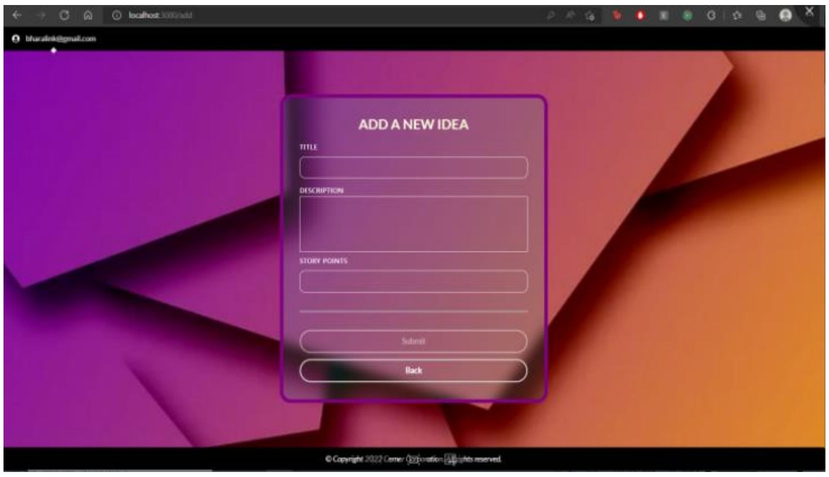
4.1.6. Login Page and Registration Page


Welcome
LOG-IN TO YOUR ACCOUNT
username
password
Log In
Don't have an account? Register Here
© Copyright 2022 Cerner Corporation. All rights reserved.

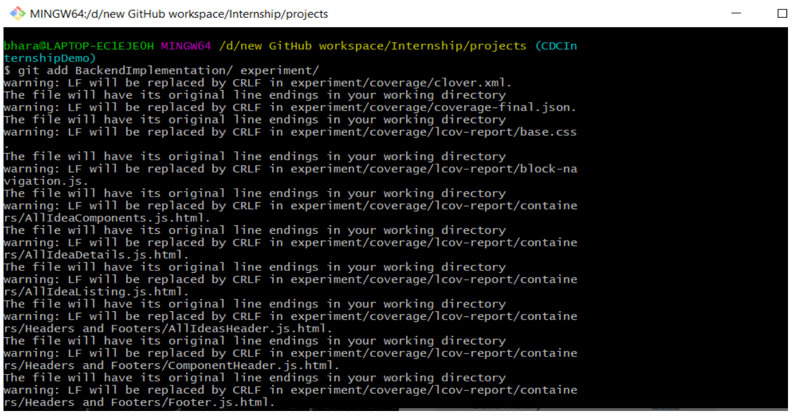

localhost:3000/registration
Welcome
CREATE AN ACCOUNT
enter your email
enter your password
confirm your password
Register
Already have an account? Login Here
© Copyright 2022 Cerner Corporation. All rights reserved.

4.1.7. Main Page (idea repository) and C.R.U.D. Example

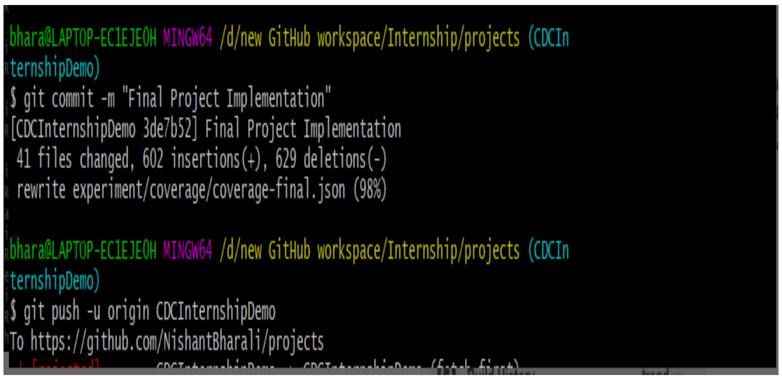




4.1.8. Pushing the updated folders of all the successfully built application, both backend and frontend to my GIT Branch.

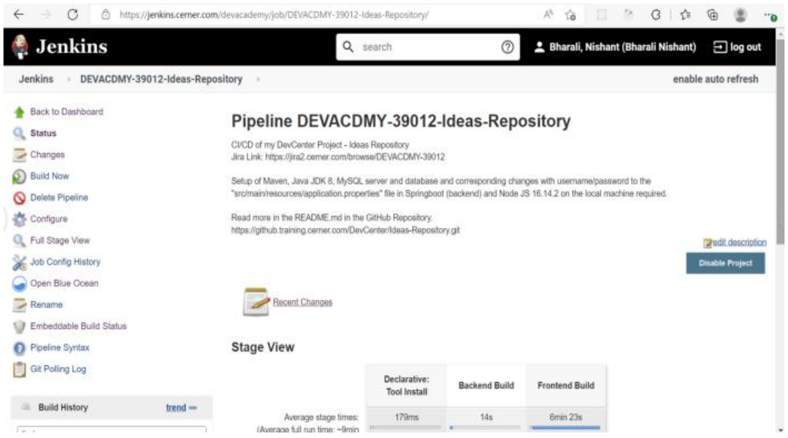


```
MINGW64:/d/new GitHub workspace/Internship/projects
bhara@LAPTOP-EC1EJEOH MINGW64 /d/new GitHub workspace/Internship/projects (CDCIn
ternshipDemo)
$ git add BackendImplementation/ experiment/
warning: LF will be replaced by CRLF in experiment/coverage/clover.xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/coverage-final.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/base.css

The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/block-na
vigation.js.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/AllIdeaComponents.js.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/AllIdeaDetails.js.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/AllIdeaListing.js.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/Headers and Footers/AllIdeasHeader.js.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/Headers and Footers/ComponentHeader.js.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in experiment/coverage/lcov-report/containe
rs/Headers and Footers/Footer.js.html.
```



```
bhara@LAPTOP-EC1EJEOH MINGW64 /d/new GitHub workspace/Internship/projects (CDCIn
ternshipDemo)
$ git commit -m "Final Project Implementation"
[CDCInternshipDemo 3de7b52] Final Project Implementation
 41 files changed, 602 insertions(+), 629 deletions(-)
 rewrite experiment/coverage/coverage-final.json (98%)

bhara@LAPTOP-EC1EJEOH MINGW64 /d/new GitHub workspace/Internship/projects (CDCIn
ternshipDemo)
$ git push -u origin CDCInternshipDemo
To https://github.com/NishantBharali/projects
```

4.1.9. Running the application as backend and frontend components in the Jenkins.



https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/

Jenkins · Bharali, Nishant (Bharali Nishant) · log out

Jenkins › DEVACDMY-39012-Ideas-Repository · enable auto refresh

Pipeline DEVACDMY-39012-Ideas-Repository

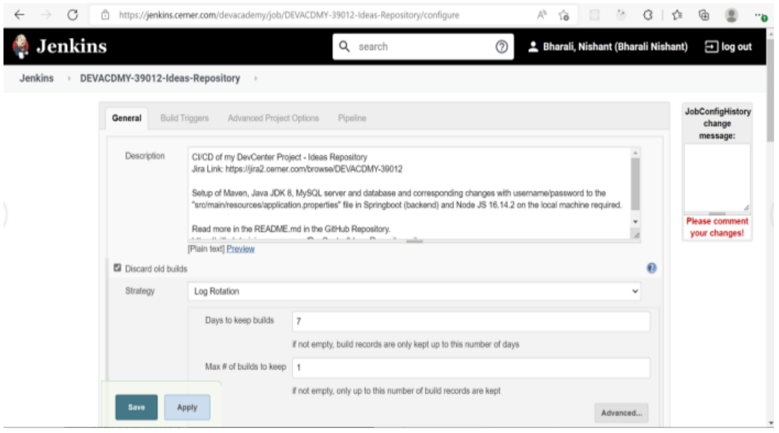
CI/CD of my DevCenter Project - Ideas Repository
Jira Link: https://jira2.cerner.com/browse/DEVACDMY-39012

Setup of Maven, Java JDK 8, MySQL server and database and corresponding changes with username/password to the "src/main/resources/application.properties" file in Springboot (backend) and Node JS 16.14.2 on the local machine required.

Read more in the README.md in the GitHub Repository.
https://github.training.cerner.com/DevCenter/Ideas-Repository.git

Stage View

	Declarative: Tool Install	Backend Build	Frontend Build
Average stage times: (Average full run time: ~9min	179ms	14s	6min 23s



https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/configure

Jenkins · Bharali, Nishant (Bharali Nishant) · log out

Jenkins › DEVACDMY-39012-Ideas-Repository

Description: CI/CD of my DevCenter Project - Ideas Repository
Jira Link: https://jira2.cerner.com/browse/DEVACDMY-39012

Setup of Maven, Java JDK 8, MySQL server and database and corresponding changes with username/password to the "src/main/resources/application.properties" file in Springboot (backend) and Node JS 16.14.2 on the local machine required.

Read more in the README.md in the GitHub Repository.

[Plain text] Preview

☑ Discard old builds

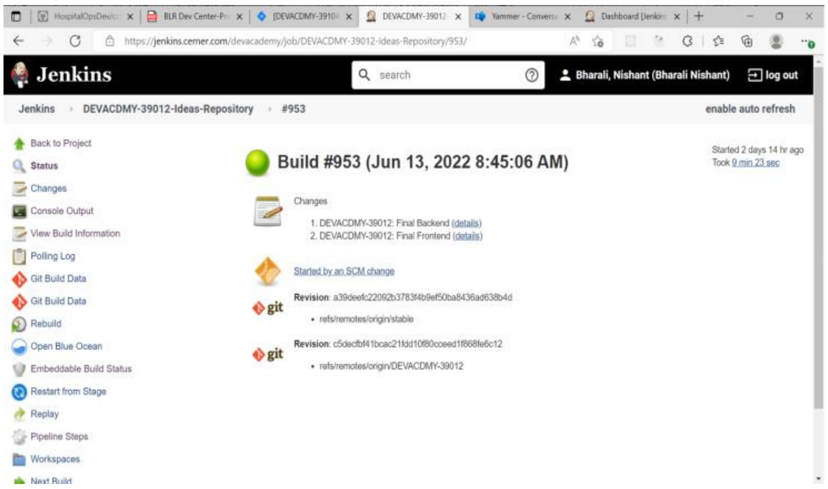
Strategy: Log Rotation

Days to keep builds: 7
If not empty, build records are only kept up to this number of days

Max # of builds to keep: 1
If not empty, only up to this number of build records are kept

JobConfigHistory change message:
Please comment your changes!

4.1.10. Jenkins Build



https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/953/

Jenkins › DEVACDMY-39012-Ideas-Repository › #953 enable auto refresh

Build #953 (Jun 13, 2022 8:45:06 AM)

Started 2 days 14 hr ago
Took 9 min 23 sec

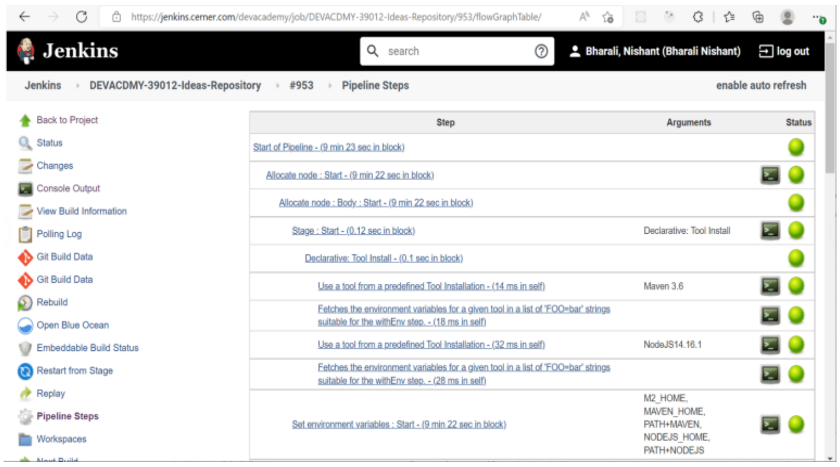
Changes
1. DEVACDMY-39012: Final Backend (details)
2. DEVACDMY-39012: Final Frontend (details)

Started by an SCM change

Revision: a39deefc22092b3783f4b9ef50ba8436ad638b4d
- refs/remotes/origin/stable

Revision: c5decfbf41bcac21fdd10f80cceed1f868fe6c12
- refs/remotes/origin/DEVACDMY-39012

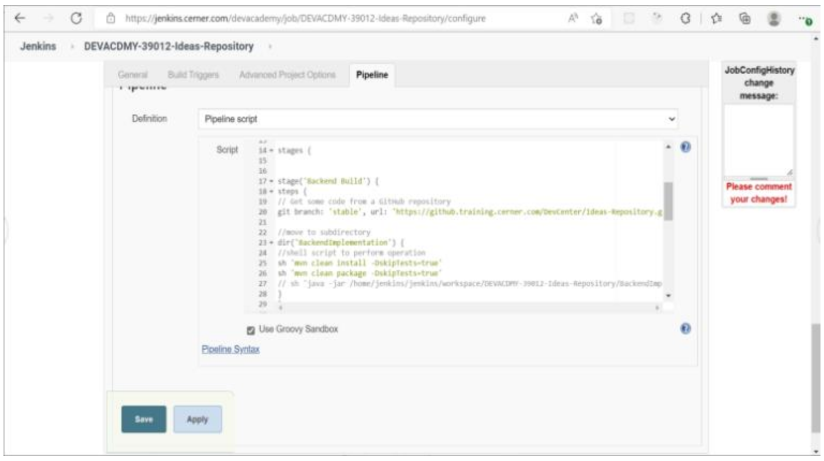
4.1.11. Jenkins Pipeline status and groovy scripts



https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/953/flowGraphTable/

Jenkins › DEVACDMY-39012-Ideas-Repository › #953 › Pipeline Steps enable auto refresh

Step	Arguments	Status
Start of Pipeline - (9 min 23 sec in block)		●
Allocate node : Start - (9 min 22 sec in block)		●
Allocate node : Body : Start - (9 min 22 sec in block)		●
Stage : Start - (0.12 sec in block)	Declarative: Tool Install	●
Declarative: Tool Install - (0.1 sec in block)		●
Use a tool from a predefined Tool Installation - (14 ms in self)	Maven 3.6	●
Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. - (18 ms in self)		●
Use a tool from a predefined Tool Installation - (32 ms in self)	NodeJS14.16.1	●
Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step. - (28 ms in self)		●
Set environment variables : Start - (9 min 22 sec in block)	M2_HOME, MAVEN_HOME, PATH+MAVEN, NODEJS_HOME, PATH+NODEJS	●



https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/configure

Jenkins › DEVACDMY-39012-Ideas-Repository

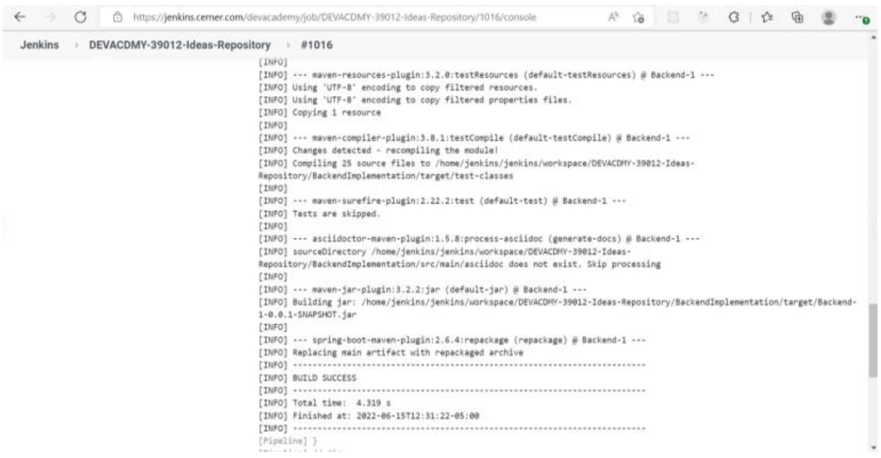
Definition: Pipeline script

Script

```
stages {


stage('Backend Build') {
steps {
// Get some code from a GitHub repository
git branch: 'stable', url: 'https://github.training.cerner.com/DevCenter/Ideas-Repository.g
 
//move to subdirectory
dir('BackendImplementation') {
//shell script to perform operation
sh 'mvn clean install -DskipTests=true'
sh 'mvn clean package -DskipTests=true'
// sh 'java -jar /home/jenkins/jenkins/workspace/DEVACDMY-39012-Ideas-Repository/BackendImp
}
```

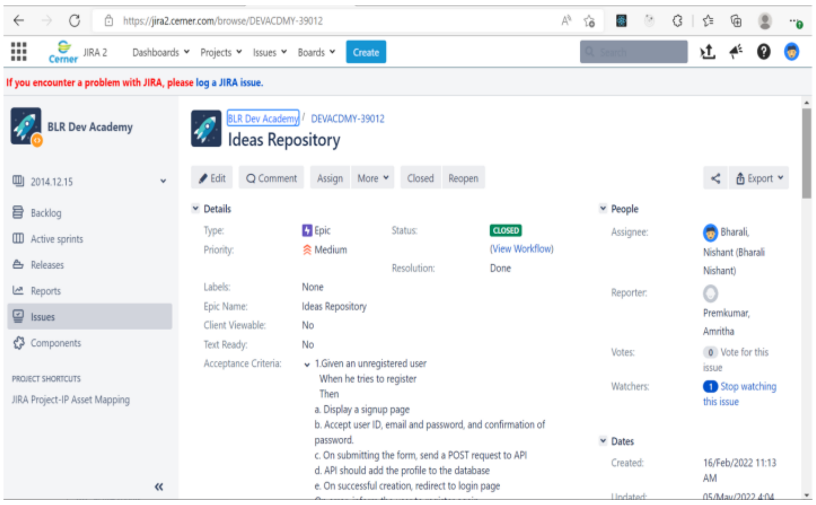
☑ Use Groovy Sandbox

Pipeline Syntax


https://jenkins.cerner.com/devacademy/job/DEVACDMY-39012-Ideas-Repository/1016/console
Jenkins
DEVACDMY-39012-Ideas-Repository
#1016

```
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ Backend-1 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ Backend-1 ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 25 source files to /home/jenkins/jenkins/workspace/DEVACDMY-39012-Ideas-
Repository/BackendImplementation/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ Backend-1 ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- asciidoctor-maven-plugin:1.5.8:process-asciidoc (generate-docs) @ Backend-1 ---
[INFO] sourceDirectory /home/jenkins/jenkins/workspace/DEVACDMY-39012-Ideas-
Repository/BackendImplementation/src/main/asciidoc does not exist. Skip processing
[INFO]
[INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ Backend-1 ---
[INFO] Building jar: /home/jenkins/jenkins/workspace/DEVACDMY-39012-Ideas-Repository/BackendImplementation/target/Backend-
1-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.6.4:repackage (repackage) @ Backend-1 ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.319 s
[INFO] Finished at: 2022-06-15T12:31:22-05:00
[INFO] ------------------------------------------------------------------------
[Pipeline] }
```

4.1.12. Jira for the research (AGILE/ Maintenance of the data about the project and tracking)



If you encounter a problem with JIRA, please log a JIRA issue.

BLR Dev Academy / DEVACDMY-39012

Ideas Repository

Details

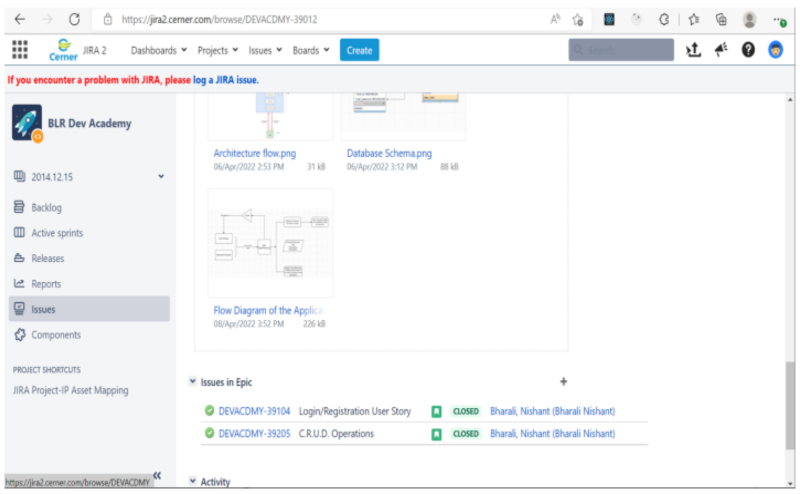
Type: Epic
Status: CLOSED (View Workflow)
Priority: Medium
Resolution: Done
Labels: None
Epic Name: Ideas Repository
Client Viewable: No
Text Ready: No
Acceptance Criteria: 1.Given an unregistered user
When he tries to register
Then
a. Display a signup page
b. Accept user ID, email and password, and confirmation of password.
c. On submitting the form, send a POST request to API
d. API should add the profile to the database
e. On successful creation, redirect to login page

People

Assignee: Bharali, Nishant (Bharali Nishant)
Reporter: Premkumar, Amritha
Votes: 0 Vote for this issue
Watchers: 1 Stop watching this issue

Dates

Created: 16/Feb/2022 11:13 AM
Updated: 05/May/2022 4:04



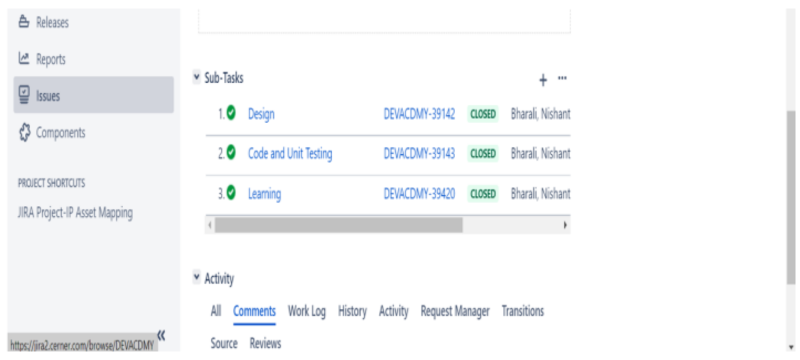
If you encounter a problem with JIRA, please log a JIRA issue.

Architecture flow.png 06/Apr/2022 2:53 PM 31 kB
Database Schema.png 06/Apr/2022 3:12 PM 88 kB
Flow Diagram of the Applica 08/Apr/2022 3:52 PM 226 kB

Issues in Epic

DEVACDMY-39104	Login/Registration User Story		CLOSED	Bharali, Nishant (Bharali Nishant)
DEVACDMY-39205	C.R.U.D. Operations		CLOSED	Bharali, Nishant (Bharali Nishant)

Activity

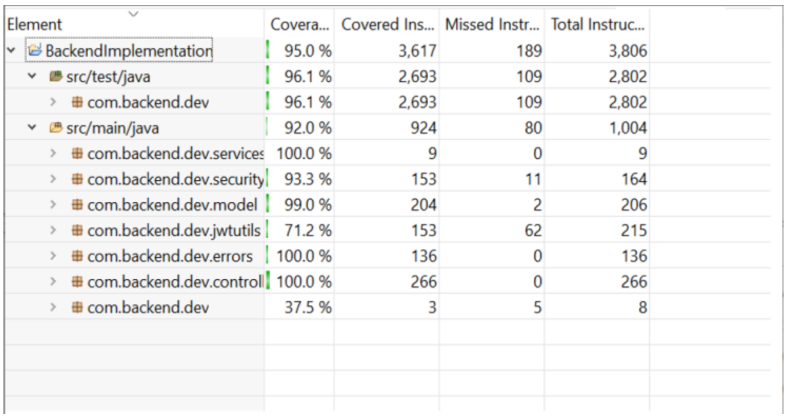


Sub-Tasks

1.	Design	DEVACDMY-39142	CLOSED	Bharali, Nishant
2.	Code and Unit Testing	DEVACDMY-39143	CLOSED	Bharali, Nishant
3.	Learning	DEVACDMY-39420	CLOSED	Bharali, Nishant

Activity

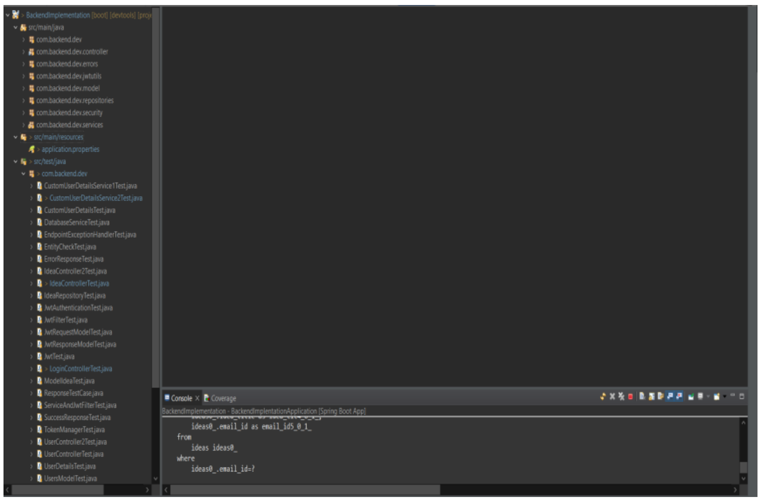
4.1.13. CODE COVERAGE (Test coverage):



Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
BackendImplementation	95.0 %	3,617	189	3,806
src/test/java	96.1 %	2,693	109	2,802
com.backend.dev	96.1 %	2,693	109	2,802
src/main/java	92.0 %	924	80	1,004
com.backend.dev.services	100.0 %	9	0	9
com.backend.dev.security	93.3 %	153	11	164
com.backend.dev.model	99.0 %	204	2	206
com.backend.dev.jwtutils	71.2 %	153	62	215
com.backend.dev.errors	100.0 %	136	0	136
com.backend.dev.controll	100.0 %	266	0	266
com.backend.dev	37.5 %	3	5	8

3.1.14. Code Sample(s)

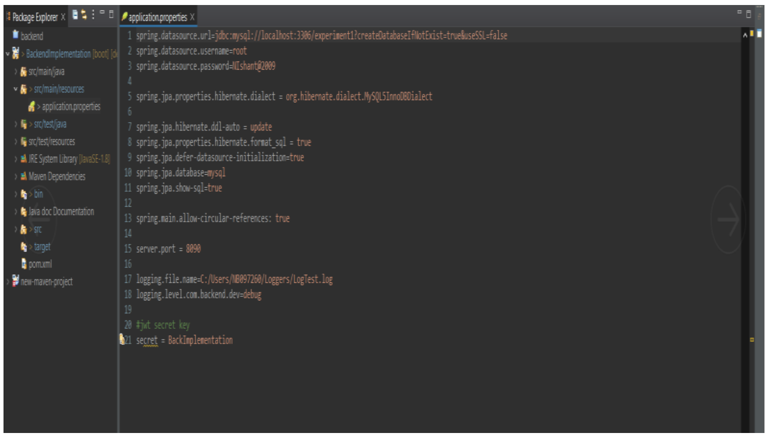
Folder and File Structure for the Backend



```
BackendImplementation [boot] [devtools] [proje
  src/main/java
    com.backend.dev
    com.backend.dev.controller
    com.backend.dev.errors
    com.backend.dev.jwtutils
    com.backend.dev.model
    com.backend.dev.repositories
    com.backend.dev.security
    com.backend.dev.services
  src/main/resources
    application.properties
  src/test/java
    com.backend.dev
      CustomUserDetailsService1Test.java
      CustomUserDetailsService2Test.java
      CustomUserDetailsTest.java
      DatabaseServiceTest.java
      EndpointExceptionHandlerTest.java
      EntityCheckTest.java
      ErrorResponseTest.java
      IdeaController2Test.java
      IdeaControllerTest.java
      IdeaRepositoryTest.java
      JwtAuthenticationTest.java
      JwtFilterTest.java
      JwtRequestModelTest.java
      JwtResponseModelTest.java
      JwtTest.java
      LoginControllerTest.java
      ModelIdeaTest.java
      ResponseTestCase.java
      ServiceAndJwtFilterTest.java
      SuccessResponseTest.java
      TokenManagerTest.java
      UserController2Test.java
      UserControllerTest.java
      UserDetailsTest.java
      UsersModelTest.java
```

```
BackendImplementation - BackendImplentationApplication [Spring Boot App]
        ideas0_.email_id as email_id5_0_1_
    from
        ideas ideas0_
    where
        ideas0_.email_id=?
```

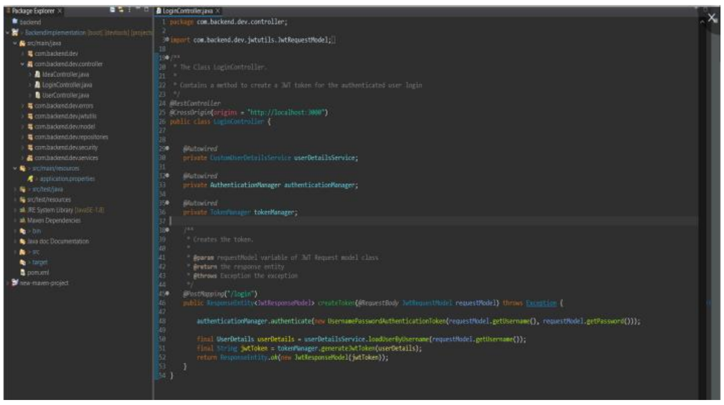
Code to integrate MySQL database with hibernate and jpa repository in Spring boot



```
backend
BackendImplementation [boot] [de
  src/main/java
  src/main/resources
    application.properties
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  bin
  Java doc Documentation
  src
  target
  pom.xml
new-maven-project
```

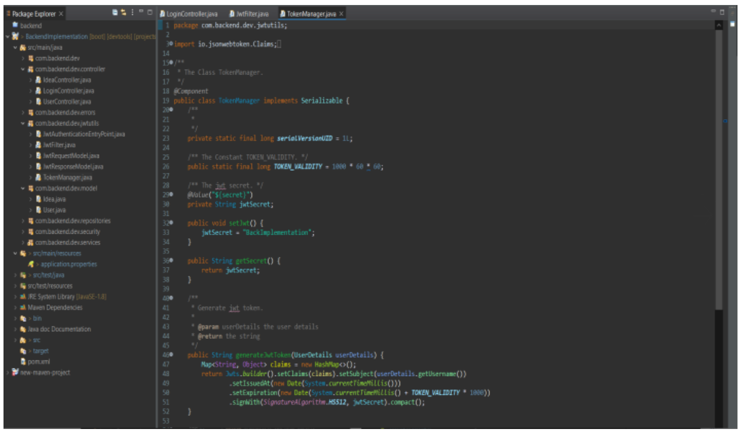
```
spring.datasource.url=jdbc:mysql://localhost:3306/experiment1?createDatabaseIfNotExist=true&useSSL=false
spring.datasource.username=root
spring.datasource.password=NIshant@2009

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.format_sql = true
spring.jpa.defer-datasource-initialization=true
spring.jpa.database=mysql
spring.jpa.show-sql=true

spring.main.allow-circular-references: true

server.port = 8090

logging.file.name=C:/Users/NB097260/Loggers/LogTest.log
logging.level.com.backend.dev=debug

#jwt secret key
secret = BackImplementation
```

Code for Login REST Controller and API with error handling



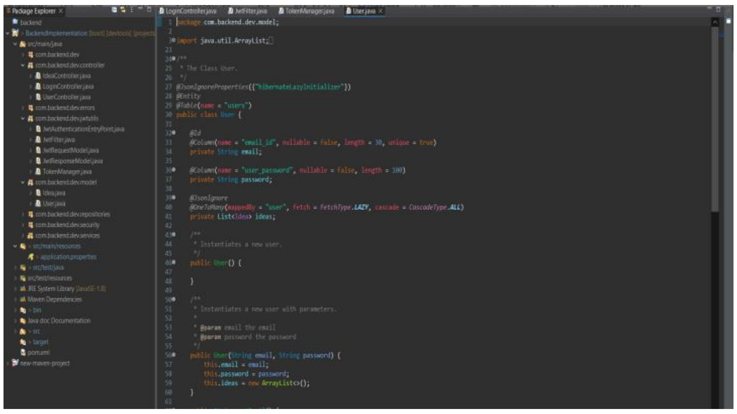
```
package com.backend.dev.controller;

import com.backend.dev.jwtutils.JwtRequestModel;

/**
 * The Class LoginController.
 *
 * Contains a method to create a JWT token for the authenticated user login
 */
@RestController
@CrossOrigin(origins = "http://localhost:3000")
public class LoginController {


    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private TokenManager tokenManager;

    /**
     * Creates the token.
     *
     * @param requestModel variable of JWT Request model class
     * @return the response entity
     * @throws Exception the exception
     */
    @PostMapping("/login")
    public ResponseEntity<JwtResponseModel> createToken(@RequestBody JwtRequestModel requestModel) throws Exception {

        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(requestModel.getUsername(), requestModel.getPassword()));

        final UserDetails userDetails = userDetailsService.loadUserByUsername(requestModel.getUsername());
        final String jwtToken = tokenManager.generateJwtToken(userDetails);
        return ResponseEntity.ok(new JwtResponseModel(jwtToken));
    }
}
```

Code to implement a JWT token for user authentication and managing Token using JWT filter



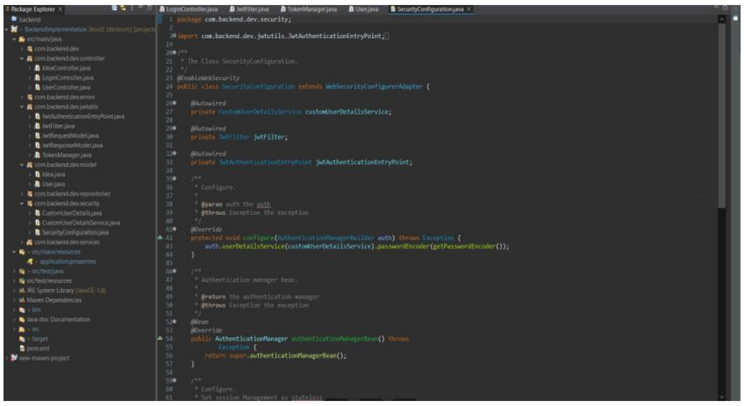
```
package com.backend.dev.jwtutils;

import io.jsonwebtoken.Claims;

/**
 * The Class TokenManager.
 */
@Component
public class TokenManager implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    /** The Constant TOKEN_VALIDITY. */
    public static final long TOKEN_VALIDITY = 1000 * 60 * 60;

    /** The jwt secret. */
    @Value("${secret}")
    private String jwtSecret;

    public void setJwt() {
        jwtSecret = "BackImplementation";
    }

    public String getSecret() {
        return jwtSecret;
    }

    /**
     * Generate jwt token.
     *
     * @param userDetails the user details
     * @return the string
     */
    public String generateJwtToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return Jwts.builder().setClaims(claims).setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + TOKEN_VALIDITY * 1000))
                .signWith(SignatureAlgorithm.HS512, jwtSecret).compact();
    }

```

Code to create User model class



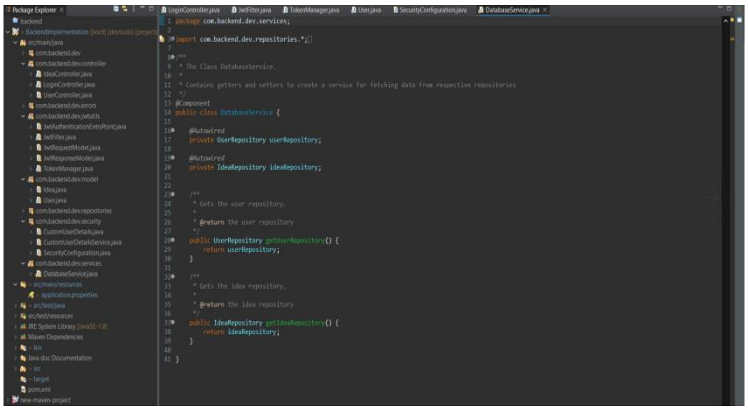
```
package com.backend.dev.model;

import java.util.ArrayList;

/**
 * The Class User.
 */
@JsonIgnoreProperties({"hibernateLazyInitializer"})
@Entity
@Table(name = "users")
public class User {

    @Id
    @Column(name = "email_id", nullable = false, length = 30, unique = true)
    private String email;

    @Column(name = "user_password", nullable = false, length = 100)
    private String password;

    @JsonIgnore
    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<Idea> ideas;

    /**
     * Instantiates a new user.
     */
    public User() {

    }

    /**
     * Instantiates a new user with parameters.
     *
     * @param email the email
     * @param password the password
     */
    public User(String email, String password) {
        this.email = email;
        this.password = password;
        this.ideas = new ArrayList<>();
    }

```

Altering the default security configuration for Spring security



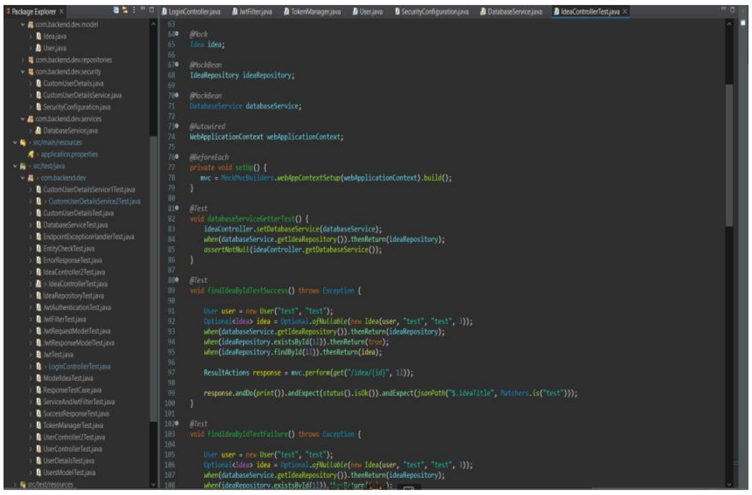
```
package com.backend.dev.security;

import com.backend.dev.jwtutils.JwtAuthenticationEntryPoint;

/**
 * The Class SecurityConfiguration.
 */
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Autowired
    private JwtFilter jwtFilter;

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    /**
     * Configure.
     *
     * @param auth the auth
     * @throws Exception the exception
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService).passwordEncoder(getPasswordEncoder());
    }

    /**
     * Authentication manager bean.
     *
     * @return the authentication manager
     * @throws Exception the exception
     */
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws
            Exception {
        return super.authenticationManagerBean();
    }

    /**
     * Configure.
     * Set session Management as stateless
```

Creating a database Service



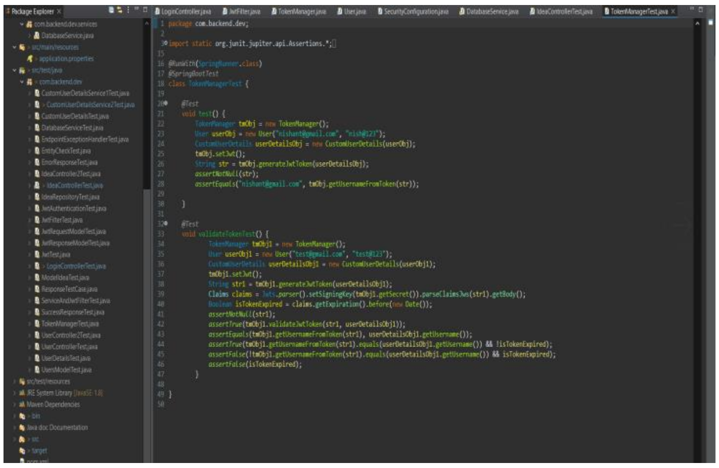
```java
package com.backend.dev.services;

import com.backend.dev.repositories.*;

/**
 * The Class DatabaseService.
 *
 * Contains getters and setters to create a service for fetching data from respective repositories
 */
@Component
public class DatabaseService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private IdeaRepository ideaRepository;


    /**
     * Gets the user repository.
     *
     * @return the user repository
     */
    public UserRepository getUserRepository() {
        return userRepository;
    }

    /**
     * Gets the idea repository.
     *
     * @return the idea repository
     */
    public IdeaRepository getIdeaRepository() {
        return ideaRepository;
    }

}
```

JUNIT Mockito and MockMvc Testing



```java

    @Mock
    Idea idea;

    @MockBean
    IdeaRepository ideaRepository;

    @MockBean
    DatabaseService databaseService;

    @Autowired
    WebApplicationContext webApplicationContext;

    @BeforeEach
    private void setUp() {
       mvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();
    }

    @Test
    void databaseServiceGetterTest() {
        ideaController.setDatabaseService(databaseService);
        when(databaseService.getIdeaRepository()).thenReturn(ideaRepository);
        assertNotNull(ideaController.getDatabaseService());
    }

    @Test
    void findIdeaByIdTestSuccess() throws Exception {

        User user = new User("test", "test");
        Optional<Idea> idea = Optional.ofNullable(new Idea(user, "test", "test", 3));
        when(databaseService.getIdeaRepository()).thenReturn(ideaRepository);
        when(ideaRepository.existsById(1l)).thenReturn(true);
        when(ideaRepository.findById(1l)).thenReturn(idea);

        ResultActions response = mvc.perform(get("/idea/{id}", 1l));

        response.andDo(print()).andExpect(status().isOk()).andExpect(jsonPath("$.ideaTitle", Matchers.is("test")));
    }

    @Test
    void findIdeaByIdTestFailure() throws Exception {

        User user = new User("test", "test");
        Optional<Idea> idea = Optional.ofNullable(new Idea(user, "test", "test", 3));
        when(databaseService.getIdeaRepository()).thenReturn(ideaRepository);
        when(ideaRepository.existsById(1l)).thenReturn([illegible]);
```

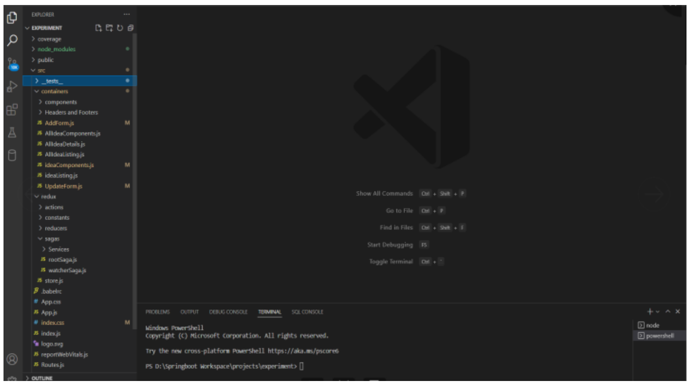
Testing JWT token reliability



```java
package com.backend.dev;

import static org.junit.jupiter.api.Assertions.*;

@RunWith(SpringRunner.class)
@SpringBootTest
class TokenManagerTest {

    @Test
    void test() {
        TokenManager tmObj = new TokenManager();
        User userObj = new User("nishant@gmail.com", "nish@123");
        CustomUserDetails userDetailsObj = new CustomUserDetails(userObj);
        tmObj.setJwt();
        String str = tmObj.generateJwtToken(userDetailsObj);
        assertNotNull(str);
        assertEquals("nishant@gmail.com", tmObj.getUsernameFromToken(str));

    }

    @Test
    void validateTokenTest() {
            TokenManager tmObj1 = new TokenManager();
            User userObj1 = new User("test@gmail.com", "test@123");
            CustomUserDetails userDetailsObj1 = new CustomUserDetails(userObj1);
            tmObj1.setJwt();
            String str1 = tmObj1.generateJwtToken(userDetailsObj1);
            Claims claims = Jwts.parser().setSigningKey(tmObj1.getSecret()).parseClaimsJws(str1).getBody();
            Boolean isTokenExpired = claims.getExpiration().before(new Date());
            assertNotNull(str1);
            assertTrue(tmObj1.validateJwtToken(str1, userDetailsObj1));
            assertEquals(tmObj1.getUsernameFromToken(str1), userDetailsObj1.getUsername());
            assertTrue(tmObj1.getUsernameFromToken(str1).equals(userDetailsObj1.getUsername()) && !isTokenExpired);
            assertFalse(!tmObj1.getUsernameFromToken(str1).equals(userDetailsObj1.getUsername()) && isTokenExpired);
            assertFalse(isTokenExpired);
        }

}
```

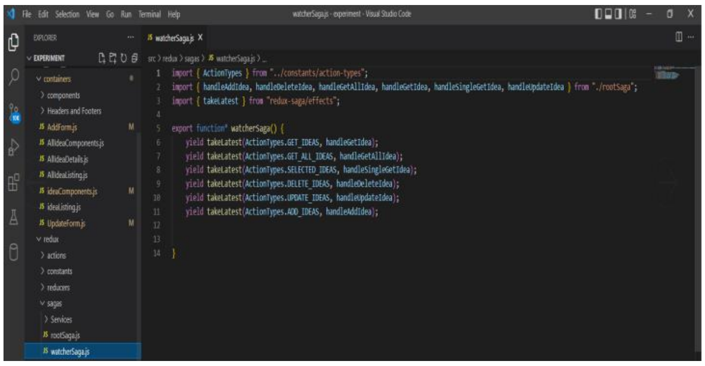
e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:01/January-2023 Impact Factor- 6.752 www.irjmets.com

Folder and File Structure for the Frontend



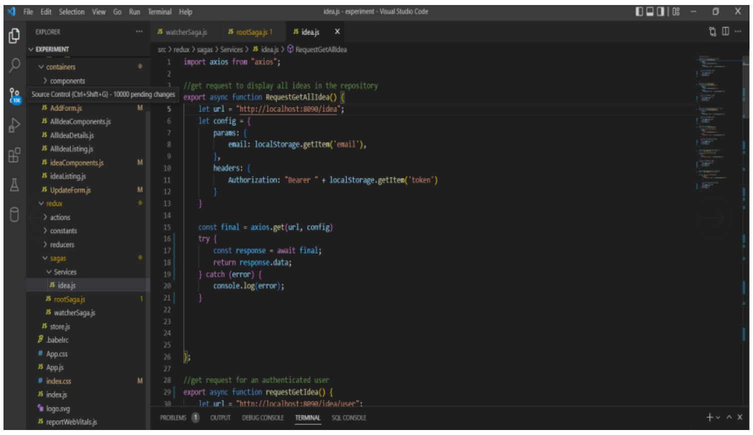
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Springboot Workspace\projects\experiment> 
```

Redux-Saga implementation:


watcherSaga.js - experiment - Visual Studio Code
src > redux > sagas > watcherSaga.js

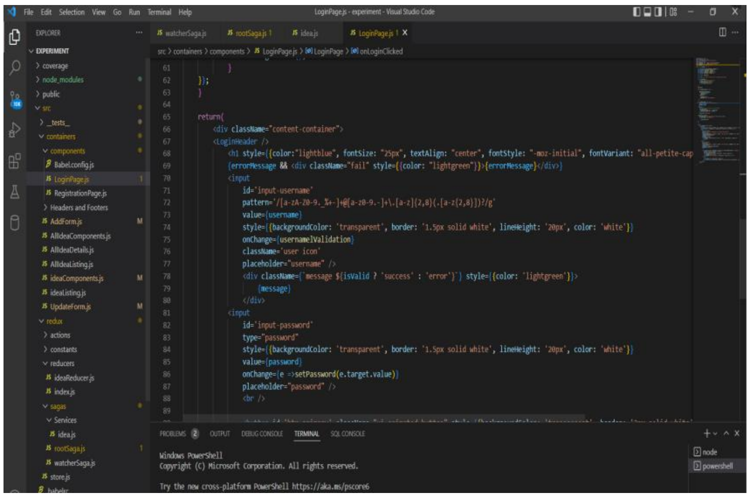
```
import { ActionTypes } from "../constants/action-types";
import { handleAddIdea, handleDeleteIdea, handleGetAllIdea, handleGetIdea, handleSingleGetIdea, handleUpdateIdea } from "./rootSaga";
import { takeLatest } from "redux-saga/effects";

export function* watcherSaga() {
    yield takeLatest(ActionTypes.GET_IDEAS, handleGetIdea);
    yield takeLatest(ActionTypes.GET_ALL_IDEAS, handleGetAllIdea);
    yield takeLatest(ActionTypes.SELECTED_IDEAS, handleSingleGetIdea);
    yield takeLatest(ActionTypes.DELETE_IDEAS, handleDeleteIdea);
    yield takeLatest(ActionTypes.UPDATE_IDEAS, handleUpdateIdea);
    yield takeLatest(ActionTypes.ADD_IDEAS, handleAddIdea);


}
```

Service/AJAX call to the backend to retrieve information using axios HTTP library:


idea.js - experiment - Visual Studio Code
src > redux > sagas > Services > idea.js > RequestGetAllIdea

```
import axios from "axios";

//get request to display all ideas in the repository
export async function RequestGetAllIdea() {
    let url = "http://localhost:8090/idea";
    let config = {
        params: {
            email: localStorage.getItem('email'),
        },
        headers: {
            Authorization: "Bearer " + localStorage.getItem('token')
        }
    }

    const final = axios.get(url, config)
    try {
        const response = await final;
        return response.data;
    } catch (error) {
        console.log(error);
    }




};

//get request for an authenticated user
export async function requestGetIdea() {
    let url = "http://localhost:8090/idea/user";
```

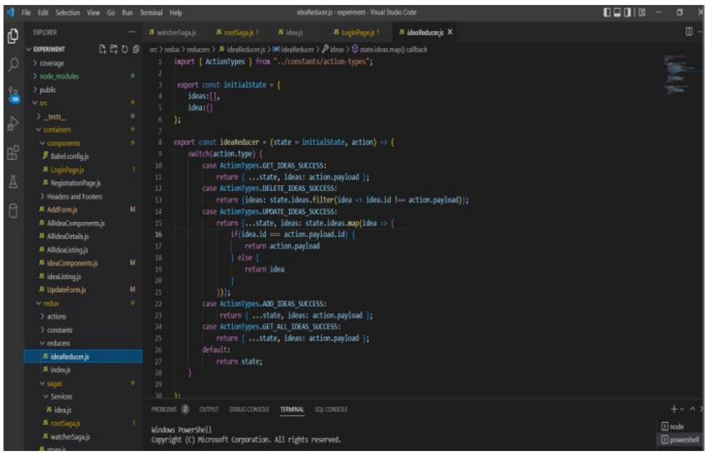
Login Page code implementation:


LoginPage.js - experiment - Visual Studio Code
src > containers > components > LoginPage.js > LoginPage > onLoginClicked

```
            }
    });
    }

    return(
        <div className="content-container">
        <LoginHeader />
            <h1 style={{color:"lightblue", fontSize: "25px", textAlign: "center", fontStyle: "-moz-initial", fontVariant: "all-petite-cap
            {errorMessage && <div className="fail" style={{color: "lightgreen"}}>{errorMessage}</div>}
            <input
                id='input-username'
                pattern='/[a-zA-Z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,8}(.[a-z{2,8}])?/g'
                value={username}
                style={{backgroundColor: 'transparent', border: '1.5px solid white', lineHeight: '20px', color: 'white'}}
                onChange={usernamelValidation}
                className='user icon'
                placeholder="username" />
                <div className={`message ${isValid ? 'success' : 'error'}`} style={{color: 'lightgreen'}}>
                    {message}
                </div>
            <input
                id='input-password'
                type="password"
                style={{backgroundColor: 'transparent', border: '1.5px solid white', lineHeight: '20px', color: 'white'}}
                value={password}
                onChange={e =>setPassword(e.target.value)}
                placeholder="password" />
                <br />
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6
```

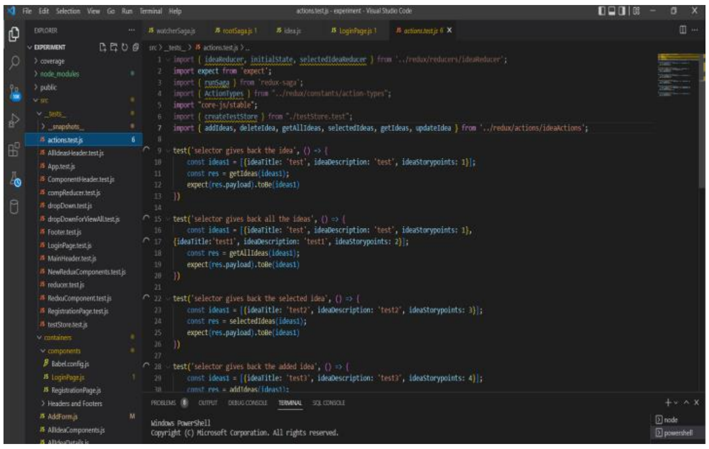
Reducer example for redux-store updation for new ideas C.R.U.D. operations:



```
import { ActionTypes } from "../constants/action-types";

export const initialState = {
    ideas:[],
    idea:{}
};

export const ideaReducer = (state = initialState, action) => {
    switch(action.type) {
        case ActionTypes.GET_IDEAS_SUCCESS:
            return { ...state, ideas: action.payload };
        case ActionTypes.DELETE_IDEAS_SUCCESS:
            return {ideas: state.ideas.filter(idea => idea.id !== action.payload)};
        case ActionTypes.UPDATE_IDEAS_SUCCESS:
            return {...state, ideas: state.ideas.map(idea => {
                if(idea.id === action.payload.id) {
                    return action.payload
                } else {
                    return idea
                }
            })};
        case ActionTypes.ADD_IDEAS_SUCCESS:
             return { ...state, ideas: action.payload };
        case ActionTypes.GET_ALL_IDEAS_SUCCESS:
            return { ...state, ideas: action.payload };
        default:
            return state;
    }

};
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

JEST enzyme Testing of the frontend components:



```
import { ideaReducer, initialState, selectedIdeaReducer } from '../redux/reducers/ideaReducer';
import expect from 'expect';
import { runSaga } from 'redux-saga';
import { ActionTypes } from "../redux/constants/action-types";
import "core-js/stable";
import { createTestStore } from "./testStore.test";
import { addIdeas, deleteIdea, getAllIdeas, selectedIdeas, getIdeas, updateIdea } from '../redux/actions/ideaActions';

test('selector gives back the idea', () => {
    const ideas1 = [{ideaTitle: 'test', ideaDescription: 'test', ideaStorypoints: 1}];
    const res = getIdeas(ideas1);
    expect(res.payload).toBe(ideas1)
})

test('selector gives back all the ideas', () => {
    const ideas1 = [{ideaTitle: 'test', ideaDescription: 'test', ideaStorypoints: 1},
{ideaTitle:'test1', ideaDescription: 'test1', ideaStorypoints: 2}];
    const res = getAllIdeas(ideas1);
    expect(res.payload).toBe(ideas1)
})

test('selector gives back the selected idea', () => {
    const ideas1 = [{ideaTitle: 'test2', ideaDescription: 'test2', ideaStorypoints: 3}];
    const res = selectedIdeas(ideas1);
    expect(res.payload).toBe(ideas1)
})

test('selector gives back the added idea', () => {
    const ideas1 = [{ideaTitle: 'test3', ideaDescription: 'test3', ideaStorypoints: 4}];
    const res = addIdeas(ideas1);
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

V. CONCLUSION

For local API testing and UI maintenance, full-stack development is a crucial development skill. At the end of the testing, we are able to confirm that the system is working as expected in terms of performance, functionality, security, and reliability. We are able to validate API endpoints' functionality in order to evaluate their performance and response on the basis of security, functional correctness, or just a status check. Not only are automated API tests faster to run, but they also guarantee ongoing API testing, which makes maintenance and debugging more effective. Because automated execution of test cases is quicker than manual execution, automated testing increases testing coverage. Additionally, it reduces reliance of testing on the test engineers' availability. Because automated tests can be run at any time in a 24x7 environment, C.R.U.D. operations testing provides coverage all day, every day. When compared to manual testing, the execution of a full-stack application that provides C.R.U.D. consumes significantly fewer resources. By creating a repository of various tests, it aids in training the test engineers and expands their knowledge. Reliability, stress, load, and performance testing are just a few examples of the kinds of testing it facilitates that would not be possible without automation.

ACKNOWLEDGEMENTS

It is my pleasure to express with deep sense of gratitude to Prof. Rajesh Kumar M, Associate Professor, School of Electronics Engineering (SENSE), Vellore Institute of Technology, for his constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavour. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of NLP.

I would like to express my gratitude to G Viswanathan (Chancellor), G.V.Selvam (Vice presidents), Dr. Anand A. Samuel (Vice Chancellor), Dr. S. Narayanan (Pro-Vice Chancellor) and Dr. Sivanantham S (Dean), School of Electronics Engineering (SENSE) for providing with an environment to work in and for his inspiration during the tenure of the course. In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Prakasam P, professor grade 1, HoD, School of Information Technology and Engineering (SITE), VIT, Vellore, all teaching staffs and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully.

VI. REFERENCES

[1] Sunil L. Bangare, Seema Borse, Pallavi S Bangare, Shital Nandedka (2012). Automated api testing approach. Sinhgad Technical Education Society Sinhgad Academy of Engineering, International Journal of Engineering Science and Technology

[2] Adeel Ehsan, Mohammed Ahmad M. E. Abuhaliqa, Cagatay Catal and Deepti Mishra, (2022). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. Department of Computer Science & Engineering, Qatar University, Doha 2713, Qatar

[3] S M Sohan, Frank Maurer, S M Sohan, Frank Maurer, Martin P. Robillard (2020). A Study of the Effectiveness of Usage Examples in REST API Documentation.

[4] Isha , Abhinav Sharma , M. Revathi. (2020). Automated API Testing. Department of Computer Science and Engineering SRM Institute of Science and Technology, Chennai, India.

[5] Venkatraj S, Vengatesan K, REST API composition for effectively testing the Cloud.

[6] Rajesh M, Rajiv Vincent, Development of Test Automation Framework for REST API Testing.

[7] REST API composition for effectively testing the Cloud. (2022).

[8] "DevOps," Mar. 2020, page Version ID: 947885950. [Online].Available:
https://en.wikipedia.org/w/index.php?title=De vOps&oldid=947885950