SAFEINSURE

Online insurance made easy

Semester finals project

Course:

Fullstack Application Development with Node.js + Express.js + React.js - 2024

Project authors:

Vasil Egov
FN: 9MI3400190

Genadi Kolev
FN: 9MI0700211

GitHub link: https://github.com/genasim/SafeInsure

Project Description

This project involves creating a comprehensive online web platform for an insurance company, aimed at improving customer experience, streamlining operations, and enhancing accessibility to insurance services, serving both customers and company workers.

The Front-end user interface is built with React.js in the form of a single-page application. React Router is utilized for efficient routing without the need for page reloads

The back-end is built on top of the Express.js framework to create a scalable and secure REST JSON API. The API exposes endpoints necessary for the platform's operation, including user authentication, policy management, claims processing, and administrative functions.

Aggregates

1. *User* - can be a client of the platform, an expert working for the organization or an admin with higher rights
2. *Policy Templates* - templates for policies that a client can choose to purchase
3. *Policies* - the actual policy instance that a client can purchase from the organization. The policy can have multiple types, from which the client selects one. The template defines the needed data for the policy and when the claim is submitted.
4. *Policy Packages* - the packages with clauses associated with a policy that a user can decide to purchase from the organization
5. *Premium Payment* - the payment of the premium for the policy
6. *Calculations and Formulas* - The formulas and coefficients associated with the calculation of the premium for the policy
7. *Claim* - the claim or claim that the user can submit to the organization when an event happens

8. *Payment for Claim* - the payment for the claim, if approved
9. *Claim Documents* - the documents associated to the claim, for example images of the car
10. *Notifications* - notifications sent to the client when his claim changes status

Main Use Cases / Scenarios

Use Case	Brief Descriptions	Actors Involved
Buy a Policy	Client buys a policy with specific clauses as per their needs	Client
Submit a claim	Client submits claim so that they can be reimbursed	Client
View user policies and claims	The client can view his policies, clauses, claims and associated payments	Client
View user notifications	View notifications about the claim	Client
Policy payments	Client can view the payments made for his policies	Client
Expert view of policies and claims	The expert view of all claims and policies	Expert
Approve or disapprove claim	The organization goes through all claims and approves or disapproves them	Expert
Manage experts	Site admins can add or remove experts to the organization.	Admin
Manage sensitive data	Can delete incorrect documents.	Admin
Claim payments	Upon approval, submitted claims by clients they are reimbursed	Expert, Client

Use Case	Brief Descriptions	Actors Involved
Buy a Policy	Client buys a policy with specific clauses as per their needs	Client
Submit a claim	Client submits claim so that they can be reimbursed	Client
View user policies and claims	The client can view his policies, clauses, claims and associated payments	Client
View user notifications	View notifications about the claim	Client
Policy payments	Client can view the payments made for his policies	Client
Expert view of policies and claims	The expert view of all claims and policies	Expert
Client notification	The client receives a real time notification of the result of his claim	Client
Document uploading	Client uploads documents related to a certain damage claim	Client
Manage calculations	Actuary sets certain calculations and formulae that are used in calculating premium and claim payments	Actuary

Frontend Web Client

Overview

The platform web UI frontend client is built with React.js, using functional components exclusively, written in Typescript, along with Bootstrap for styling and React Router for seamless navigation, without reloading the page. Upon logging into the platform the backend server sends a JWT token to the client that is then stored into the browser's Session Storage.

Most routes are role-specific and are thus protected. The role is extracted from the token and if the protected route allows it to pass the page is then rendered; otherwise the user is redirected to the **/** route. If a token is missing altogether, the user is redirected to the **/login** route to authenticate.

Routes

Unprotected routes		
View name	**Brief Description**	**URI**
Landing page	The landing page of the application. Shows the options for policies that a client can purchase. If the client is not logged in, then they are redirected to the login page when they select a policy to purchase.	/
Login	Users can login into an existing account. Upon successful login, the sent JWT is saved in Session Storage	/login
Register	Users can create a fresh account with *only* Client privileges. Upon successful register, the sent JWT is saved in Session Storage	/register

Client routes		
View name	**Brief Description**	**URI**
Policy purchase	Clients can buy a new Policy that will be added to their account. When buying they fill the needed information for the template and select their policy clauses	/client/policies
Claims dashboard	Clients get a summary of their pending claims and policies. They can select a policy and create a new claim for it	/client/claims
Submit claim	Clients can submit a claim on one of their policies by filling out the form	/client/claims/:policyId
Notifications	Notifications relevant to clients' current policies and claims and profile (if any)	/client/notifications

Expert routes		
View name	**Brief Description**	**URI**
Policies Dashboard	View for all the policies in the system and their detailed information	/backoffice/policies
Claims dashboard	Clients get a summary of their pending claims and policies. They can select a policy and create a new claim for it	/backoffice/claims
Claim Payments	View for all the claim payments in the system	/backoffice/claim-payments
Policy Payments	View for all the policy payments in the system	/backoffice/premium-payments

Actuary Views		
View name	**Brief Description**	**URI**
Manage Coefficients	Dashboard for the actuary coefficients	/actuary
Update Coefficient	Update a particular coefficient's values, description, name, etc	/actuary/:coefficientId
Create Coefficient	Create a coefficient	/actuary/create-coefficient

Admin Views		
View name	**Brief Description**	**URI**
Admin Dashboard	Admins can manage the platform's user accounts, create users with backoffice rights and manage claim documents	/admin
Update user	Admins can update a particular user and change their rights, info, etc	/admin/:userId

Backend Express server and Mongo database

Overview

For the backend of the application, we have used JavaScript/TypeScript on the Node.js runtime. The backend is implemented as a web server, using mainly the HTTP protocol. The framework of choice is Express.js. As the the choice for the backend is TypeScript, a suitable database would be a one with the document paradigm. This is why MongoDB was chosen for the persistence of the project.

The backend consists of separate HTTP methods that execute different types of business logic, ranging from login and logout, register, create a policy, submit a claim, back office operations as approval and rejection for a given claim and other. This gives a finished product for submitting and managing claims and policies by clients and back office workers.

API Resources		
	Brief Description	**URI**
Admin	GET. Get all users paginated for admin	*/api/admin/users*
	GET. Get user details by id	*/api/admin/users/:id*
	POST. Creates a user	*/api/admin/users*
	PATCH. Updates the user	*/api/admin/users/:id*
	DELETE. Deletes a user	*/api/admin/users/:id*
	GET. Get all claim documents.	*/api/admin/claim-documents*
	DELETE. Deletes a claim document	*/api/admin/claim-documents/:id*
Actuaries	GET. Gets all coefficients	*/api/actuaries/coefficients*
	GET. Gets coefficient by id	*/api/actuaries/coefficients*

		/:id
	POST. Creates a coefficient	*/api/actuaries/coefficients*
	PATCH. Updates a coefficient	*/api/actuaries/coefficients /:id*
	DELETE. Deletes a coefficient.	*/api/actuaries/coefficients /:id*
Auth	POST. Allows a user to register	*/api/auth/register*
	POST. Allows a user to login	*/api/auth/login*
	POST. Validates an email	*/api/auth/valid-email*
	GET. Gets all policy types	*/api/auth/policy-types*
Clients	GET. Gets all claims	*/api/clients/claims*
	POST. Creates a claim	*/api/clients/claims*
	GET. Gets all policies	*/api/clients/policies*
	POST. Creates a policy	*/api/clients/policies*
	GET. Gets a policy by id	*/api/clients/policies/:id*
	GET. Get policy type packages	*/api/clients/policies/:type/ packages*
	GET. Get policy type coefficients	*/api/clients/policies/:type/ coefficients*
Experts	GET. Gets the claim by id	*/api/backoffice/claims/:id*
	GET. Gets all claims	*/api/backoffice/claims*
	PUT. Updates claim by id	*/api/backoffice/claims/:id*
	GET. Gets all policies	*/api/backoffice/policies*
	GET. Gets all premium payments	*/api/backoffice/premium-payments*
	GET. GEts all claim payments	*/api/backoffice/claim-pay*

		ments
Notifications	GET. Get all notifications for the user	*/api/notifications*

Initializing the project

When starting a freshly downloaded copy of the project, users need to ensure that they have a running MongoDB instance, be it local, dockerized or in the cloud, and import the following .json seeds for these collections:

- utils/backend/seeds/users.json -> users
- utils/backend/seeds/policy-packages.json -> policy-packages

This is the bare minimum of data needed to initialize the project and correctly create the rest of the entities.

Also, do make sure to have an .env at **/backend/src** file with the following entries:

- PORT (preferably 5000)
- MONGO_URL -> connection string for your MongoDB instance
- JWT_SECRET -> random string, used for the encoding of the JWT tokens