**CONNEX INSURANCE QUOTE APPLICATION - DOCUMENTATION**

**Requirements and Installation:**

Before deploying the application, please make sure that the following toolkits and dependencies have been installed on the system.

i) Make sure that latest JRE is installed. link: 'https://www.java.com/en/download/manual.jsp'

ii) Download and install 'Liberica Java JDK (JDK 17)' from 'https://bell-sw.com/pages/downloads/'

iii) Download and install NodeJS version-18.16.1 LTS from 'https://nodejs.org/en'

iv) Download and install required npm packages

* Install Bootstrap from npm by using the command 'npm install bootstrap bootstrap-icons'

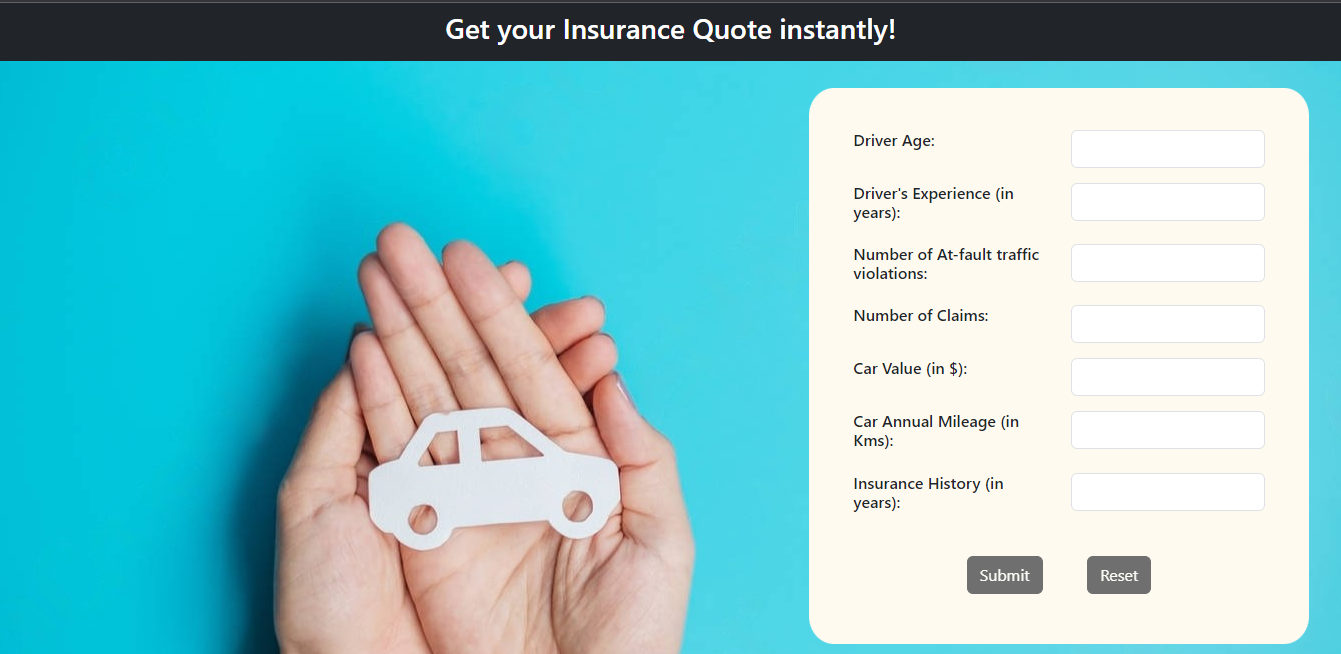
Deployment:

i) Start the backend application server by navigating to 'SpringBoot/project/' and run the command 'mvnw spring-boot:run'

ii) Start the frontend application by navigating to 'Angular/project/' and run the command 'ng serve'

View the application:

Go to your browser and type the URL 'localhost:4200'. you should be able to view the UI of the application as shown below



Application – Overview:

The objective of the application is to collect user information and factors involved in insurance premium calculation and notify the user of the insurance premium that is calculated. The application has been developed by using Angular in the front-end and SpringBoot in the backend. The backend contains an API service using which we can request for insurance quote by providing the necessary parameters.

API details:

Headers are not required since the Accept and Content-Type are both json. No authorization, since insurance quote application doesn’t need one.

URL - localhost:8080/getQuote

The request schema for the back-end API is as follows:

{

 "age":26, //Driver’s Age : min-16, max-120

 "experience": 6, //Driver’s Experience : min-0, max-100

 "record": 0, //No. of violations : min-0, max-1000

 "claims": 0, //No. of previous claims : min-0, max-1000

 "car\_value": 20000, //Car’s value : min-0, max-9999999

 "car\_annual\_mileage": 15000, //Car’s mileage a year : min-0, max-999999

 "insurance\_history": 6 //History of insurance : min-0, max-100

}

The response schema for the back-end API is as follows:

{

    "premium": 1414.29, //premium for 12 months

    "reference": "983BC10", //quote reference

    "message": "Your premium has been calculated!", //message-description

    "available": true //is quote calculated

}

Input validations have been implemented at both the front-end and the API at the back-end.

Also refer to the postman collections in the github repo, as needed.

**Challenges faced during development:**

1. CORS:

It is always a challenge to deploy an angular application, especially since it runs as a separate process on itself in a separate port. Due to this, the origin of the deployed server and the deployed front-end are different. During my previous tenure as a software engineer, since I wasn’t involved much with configuration of end points, I found it to be something new and had to search a lot of resources to arrive at the solution.

In order to overcome this problem, I added ‘proxy.conf.json’ to bypass the existing api target.

{

    "/api": {

      "target": "http://localhost:8080",

      "secure": true,

      "changeOrigin": true,

      "pathRewrite": {

        "^/api": ""

      }

    }

  }

1. Calculation of insurance quote:

The first step that was took was to arrive at the logic for the calculation of the insurance quote. The calculation should also be most efficient, so that the application would be scalable if this function was to be used. Hence the premium was calculated by determining factor value for each variable, adding them all up and then dividing them by 7 (number of variables) so that an average factor value could be found. This value was then multiplied with the base premium amount to arrive at the quote.

The code is present in DriverDetailsService.java:

**public** Double getFactoredPremium() **throws** UnknownPremiumException

{

**return** (**double**) Math.*round*(getOverallFactor() \* getBasePremium() \* 100) / 100; //To round up to 2 decimal places

}

**public** Float getOverallFactor() **throws** UnknownPremiumException

{

**return** (getAgeFactor() + getExperienceFactor() + getRecordFactor() + getClaimsFactor() + getValueFactor() + getMileageFactor() + getHistoryFactor()) / 7;

}

1. Validations in the application:

One of the most significant, but easy feature to add to an application is the validation of inputs to the service as well as the front-end. In SpringBoot, validations are implemented using the hibernate-validator framework in-built in Spring. Made use of Jakarta validations since the javax validations had become deprecated and is no longer supported in the SpringBoot framework. Handled the messages through an ExceptionHandler and the BindingResult object to include the customized message. i18n is also enabled for future use, by implementing the validation messages in resources/messages.properties file.

@Override

**protected** ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException exception,

HttpHeaders httpHeaders, HttpStatusCode httpStatus, WebRequest webRequest) {

Map<String, Object> objectBody = **new** LinkedHashMap<>();

objectBody.put("Status", httpStatus.value());

List<String> exceptionalErrors = exception.getBindingResult()

.getFieldErrors()

.stream()

.map(field -> field.getDefaultMessage())

.collect(Collectors.*toList*());

objectBody.put("Errors", exceptionalErrors);

**return** **new** ResponseEntity<>(objectBody, httpStatus);

**}**

**//Exception handler file**

**Significant features in the application:**

* Form validations in the UI – limited characters, range and empty field checks, using Angular’s Validator framework. Also, the form submission is forbidden until the form is valid
* Reset form feature, where all validations and field values are cleared, when reset button is clicked
* Copy to clipboard feature is included for the insurance quote reference in UI
* Efficient calculation of quote for better back-end service
* Custom exception called “UnknownPremiumException” implemented in java to distinguish the feasibility of premium calculation
* i18n – future extension support provided in SpringBoot and for the front-end angular has its own framework for i18n support
* Responsive and bootstrap loaded UI for better user experience
* The API calls is isolated from component in Angular to avoid coupling of functions
* Validations in the back-end using Spring’s Jakarta API and ExceptionHandler