

Web based Applications for Insurance Services

FINAL REVIEW REPORT

Submitted by

Ekansh Gupta (18BCI0175)

Mihir Srivastava (18BCI0214)

Laksh Gupta (18BCI0190)

Sumeet Roy Kurian (18BCI0188)

Prepared For

DATABASE SYSTEMS (CSE2004) – PROJECT COMPONENT

Submitted To

Dr. Ramanathan L

Assistant Professor (SG)

School of Computer Science and Engineering



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

INDEX

- Abstract
- Introduction
- Requirement
- Literature survey
- ER diagram
- List of tables
- Work Breakdown
- Output screenshots
- Code snippets

Abstract

Insurance is a means of protection from financial loss. It is a form of risk management, primarily used to hedge against the risk of a contingent or uncertain loss. In these days it is a necessity for everyone to own insurances against different kinds of threats.

So we came up with an idea of providing insurance services to people by using a website.

This website caters people of different of sects of society with a variety of insurances. One can select the desired insurance types and premium according to his income and demand.

This project uses concepts of database and web development. In this project we are making an insurance website which will take information from user. Our website has a user friendly user interface and it is being created using HTML-CSS and Bootstrap. Our project uses mongo database to store and retrieve data from database. Node JS is used as a tool for connectivity between databases and front-end. This project is a great learning experience especially for the concepts of database and its practical application.

Introduction

Insurance website utilizes concept of web development along with database organization. For frontend web development we have utilized HTML-CSS and Bootstrap. We have created multiple linked webpages. Our homepage contains multiple insurance options for user to choose along with a user login and user sign up options.

Multiple insurance options provided in this webpages utilize concept of hyperlinks to link different web pages. Each webpage connected to home page consist different forms to collect data from user.

The data which is collected here is stored to mongo database. Connecting our mongo database with our web page required utilization of back-end development through Node js.

Requirements

1. Software Requirements

- MONGO DB SHELL
- NODE JS
- VS CODE
- EXPRESS JS

2. Hardware Requirements

- Processor-ryzen3 or above/Intel- i3 3rd gen or above
- Graphic card- UHD 520
- RAM 2gb or above

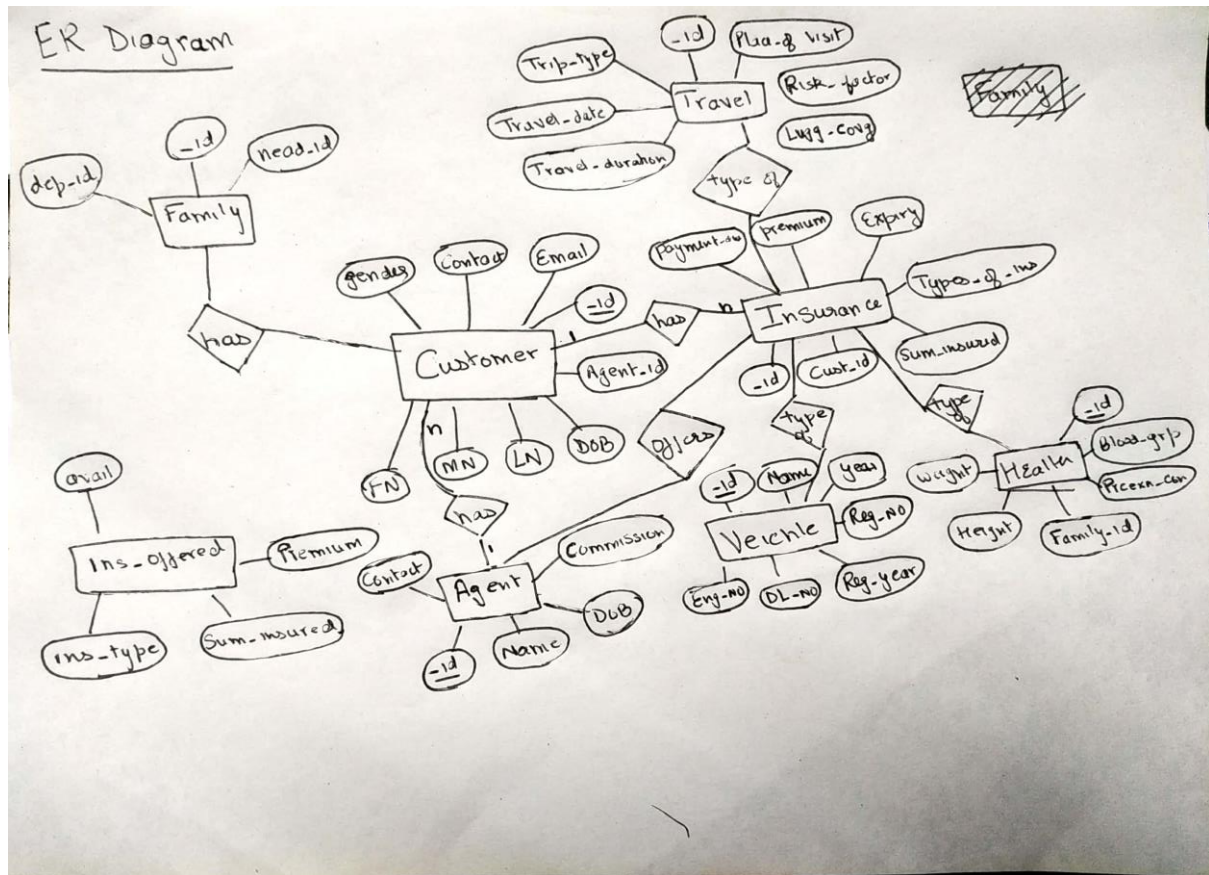
Literature Survey

In this section, literature survey is given. Accordingly, research papers are reviewed and analyzed based on the prediction methods used.

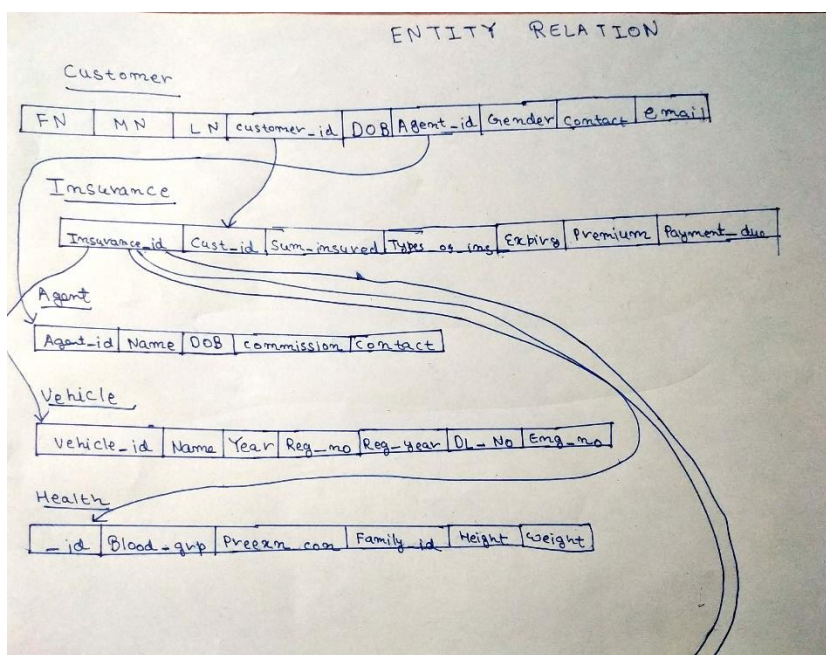
Authors	Method	Purpose	Advantages	Disadvantages
Momenul Ahmad	How to do web development	Giving general idea about web development and internet.	It provides essential knowledge to beginner.	Knowledge provided is limited and it is not helpful to a more experienced developer.
<u>Joko Santoso</u>	Building node applications with mongodb and backbone	Provides knowledge about mongo database	It is vast and contains wide range of examples.	It require a pre requisite knowledge and not recommended for beginner.
Kavya.S	1.1.1.1 A Study on Mongodb Database	Gives an overview on mongo database	Explains general functioning of mongo data base.	Doesn't contain practical examples.
<u>Mazhar Ahmed</u>	1.1.1.2 Kickstart with Node.js and MongoDB	Provides general idea about node js and mongo database	Contains a nice overview of node js and mongo database	It would have been better if it contain more examples and if was better organised.
Me Me Khaing, Kyi Pyar	1.1.1.3 Why Undergraduates Should Learn Web Development and Design Foundations	Provides basic knowledge of HTML and web	Provides HTML syntax and some examples regarding it.	It is nice document but it lacks some or the other syntax and contains less

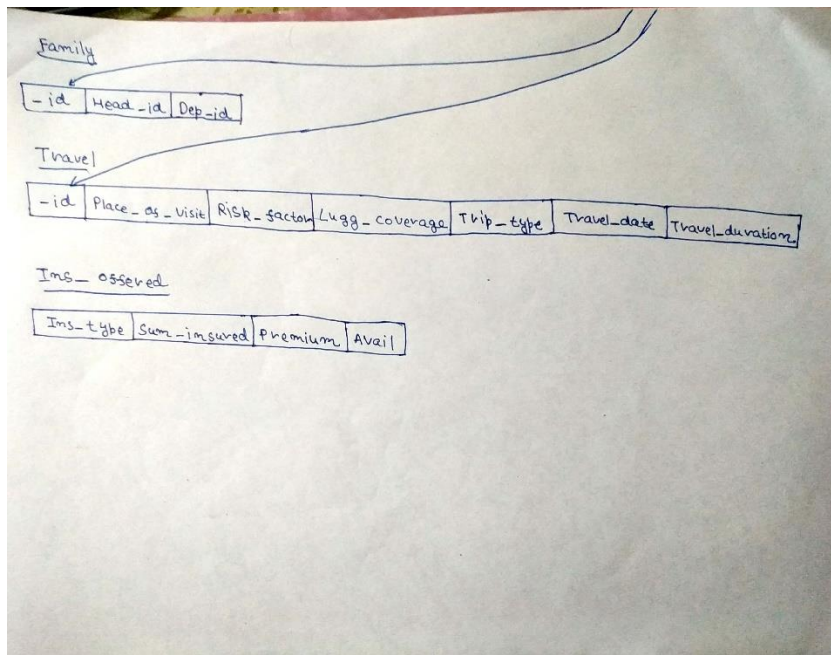
	with HTML5	development		examples.
--	------------	-------------	--	-----------

ER Diagram:



Entity Relationship Model





List of Tables

Table: customer

Attribute	Datatype	Constraint
FN	Varchar2	Not null
MN	Varchar2	
LN	Varchar2	Not null
_id	Varchar2	Primary key
DOB	Date	
Agent_id	Varchar2	Foreign key to agent table
Gender	Varchar2	Sex in(M/F)
Contact	Number	Not null, Unique
Email	Varchar2	Not null, Unique

Table: Insurance

Attribute	Datatype	Constraints
_id	Varchar2	Primary key
Cust_id	Varchar2	Foreign key
Sum_insured	Number	Not null
Types_of_ins	Varchar2	Not null

Expiry	Date	Not null
Premium	Number	>0
Payment_due	Char	Y/N

Table: Agent

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary key
Name	Varchar2	Not null
DOB	Date	Not null
Commission	Number	
Contact	Number	Not null, unique

Table: Vehicle

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary Key
Name	Varchar2	Not null
Year	Varchar2	
Reg_no	Varchar2	Unique, Not null
Reg_year	Varchar2	
DL_No	Varchar2	Unique, Not null
Eng_no	Varchar2	Unique, Not null

Table: Health

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary key
Blood_grp	varchar2	Not Null
Preexn_con	Varchar2	
Family_id	Varchar2	
Height	Number	Not Null, >0
Weight	Number	Not Null, >0

Table: Family

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary key
Head_id	Varchar2	Not null
Dep_id	Varchar2	

Table: travel

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary key
Place_of_visit	Varchar2	Not null
Risk_factor	Number	
Lugg_coverage	Char	Not null
Trip_type	Varchar2	

Travel_date	Date	
Travel_duration	Number	

Travel: Ins_offered

<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
Ins_type	Varchar2	
Sum_insured	Number	>0
Premium	Number	>0
Avail	Char	Y/N

Table: life_ins

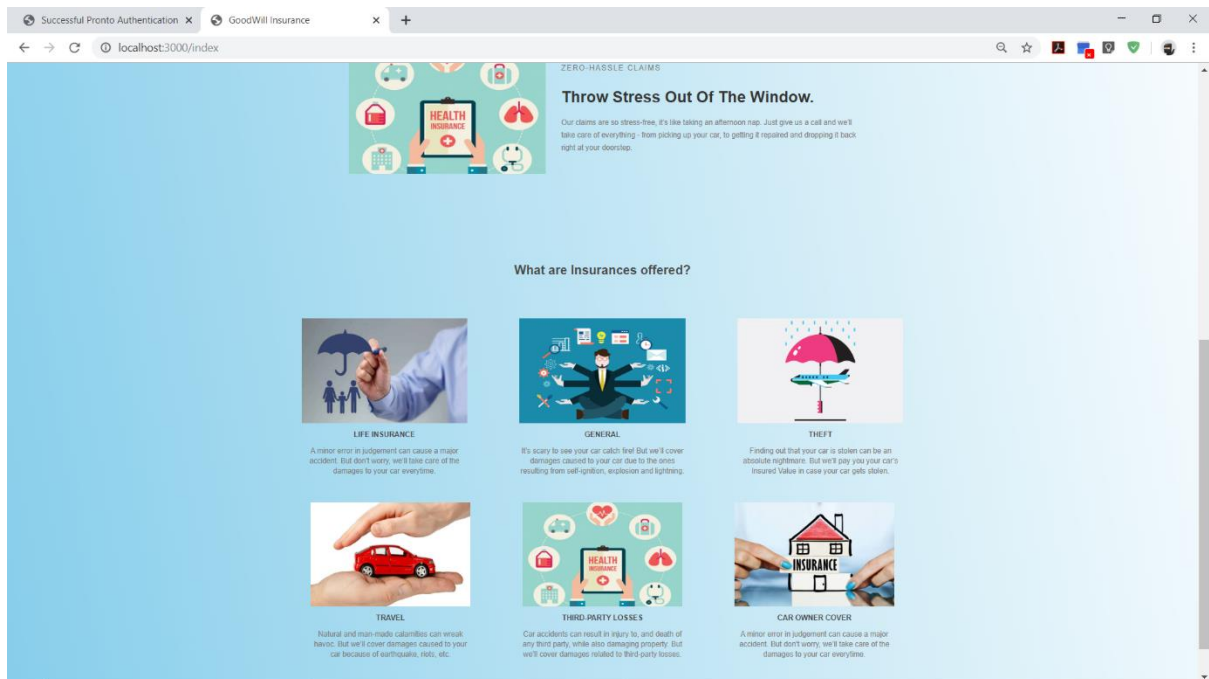
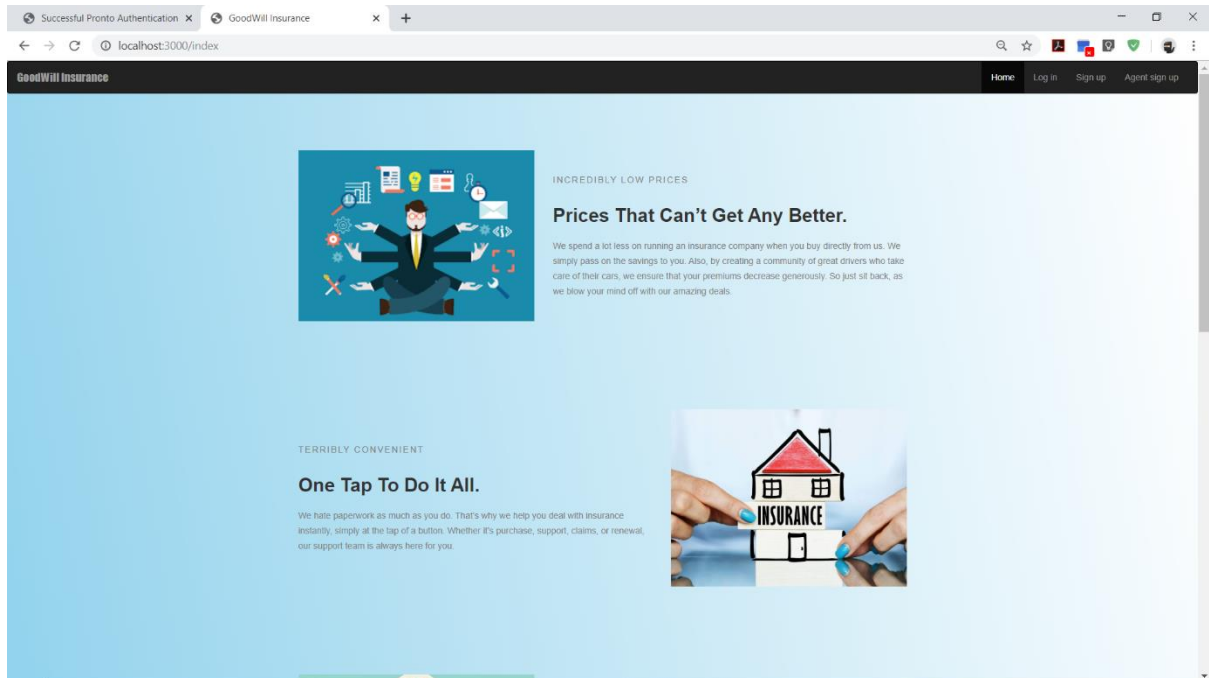
<i>Attribute</i>	<i>Datatype</i>	<i>Constraints</i>
_id	Varchar2	Primary key
Cover	Number	Not Null
Terms	Varchar2	
Nominee	Varchar2	Not Null
Nominee_rel	Varchar2	
Health_history	Varchar2	

Work break down:

<i>Team Member Registration Number</i>	<i>Name</i>	<i>Work Assigned</i>
18BCI0190	Laksh Gupta	Frontend web dev, Mongo backend
18BCI0214	Mihir Srivastava	Frontend web dev, Mongo backend
18BCI0175	Ekansh Gupta	Mongo DB, Node JS connectivity
18BCI0188	Sumeet Roy Kurian	Node JS connectivity, Mongo DB

Screenshots

Home screen



Sign in page

Goodwill Insurance

Home Login Sign up Agent Sign up

SIGN IN

1001

Log In

Sign up page-customer

Goodwill Insurance

Home Login Sign up Agent Sign up

Sign Up

Customer id

First Name

Middle Name

Last Name

DOB :

dd-mm-yyyy

Gender: M/F

Address

Contact

Email

1001

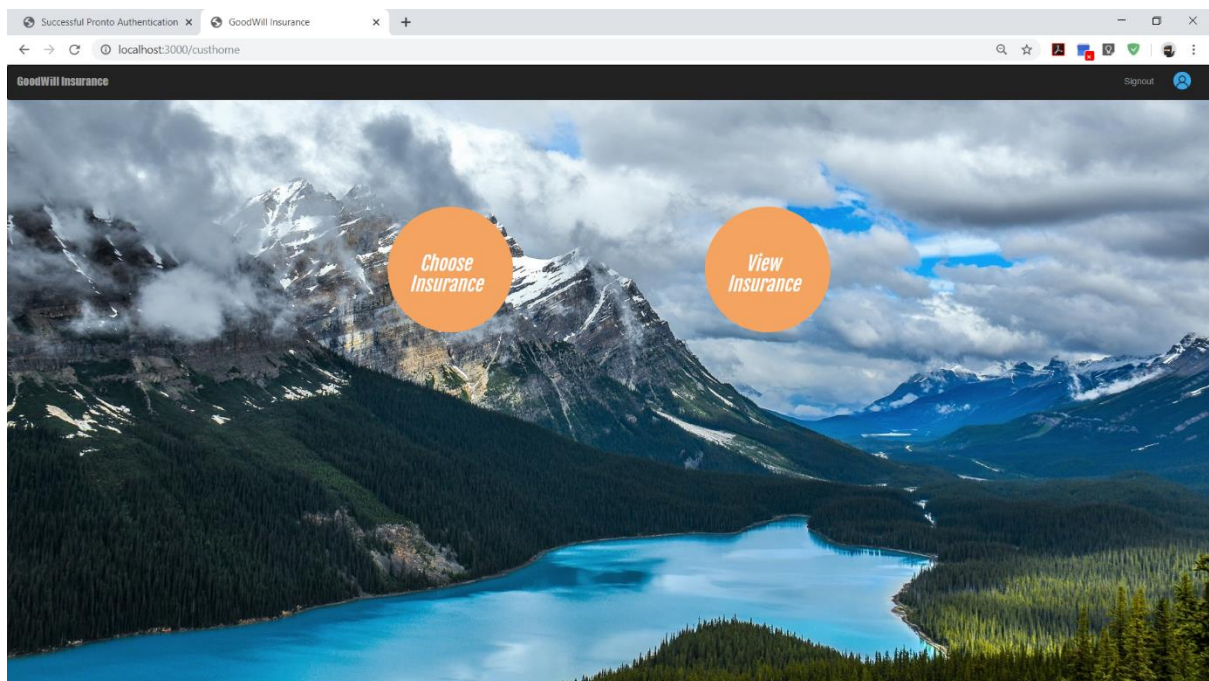
Sign Up

Agent sign up

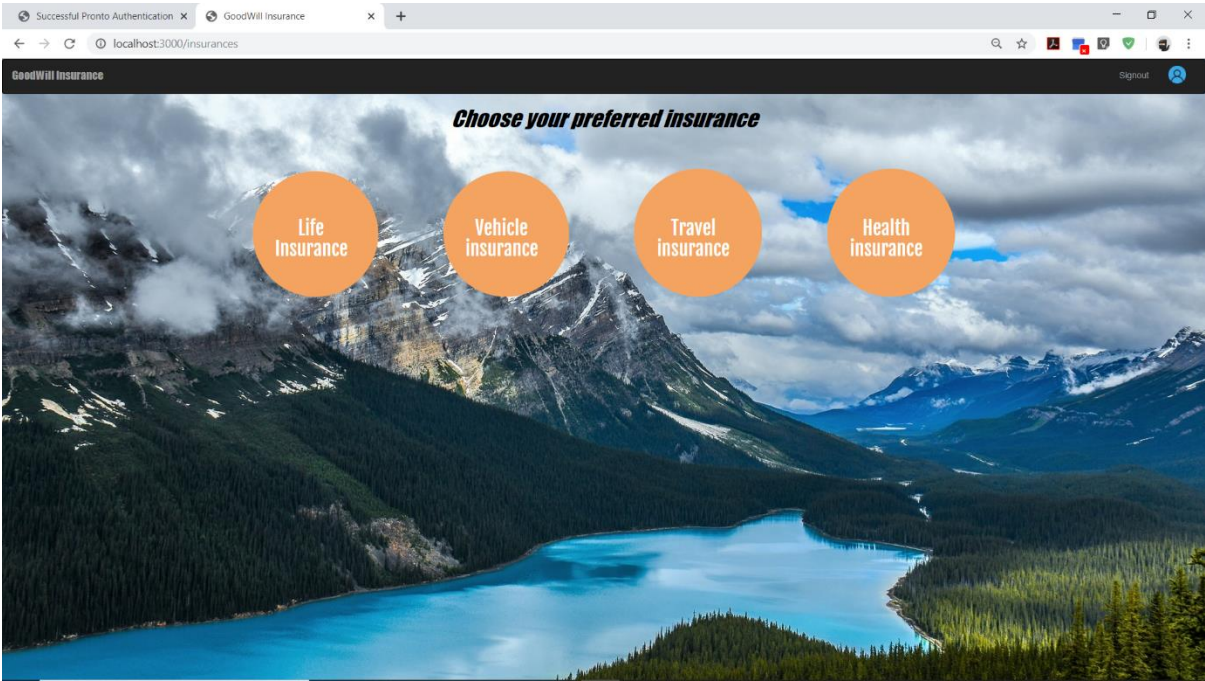
The screenshot shows a web browser window with two tabs: 'Successful Pronto Authentication' and 'GoodWill Insurance'. The address bar shows 'localhost:3000/Agentlog'. The page header includes 'Goodwill Insurance' and navigation links for 'Home', 'Login', 'Sign up', and 'Agent Sign up'. The main content area has a blue background with a central black sign-up form. The form is titled 'SIGN UP' and contains the following fields and buttons:

- USER ID**: A text input field containing '1001'.
- NAME**: A text input field containing '*****'.
- DOB**: A date input field with a dropdown arrow, showing 'dd-mm-yyyy'.
- Contact**: A text input field.
- Commission**: A text input field.
- Sign Up**: A green button.

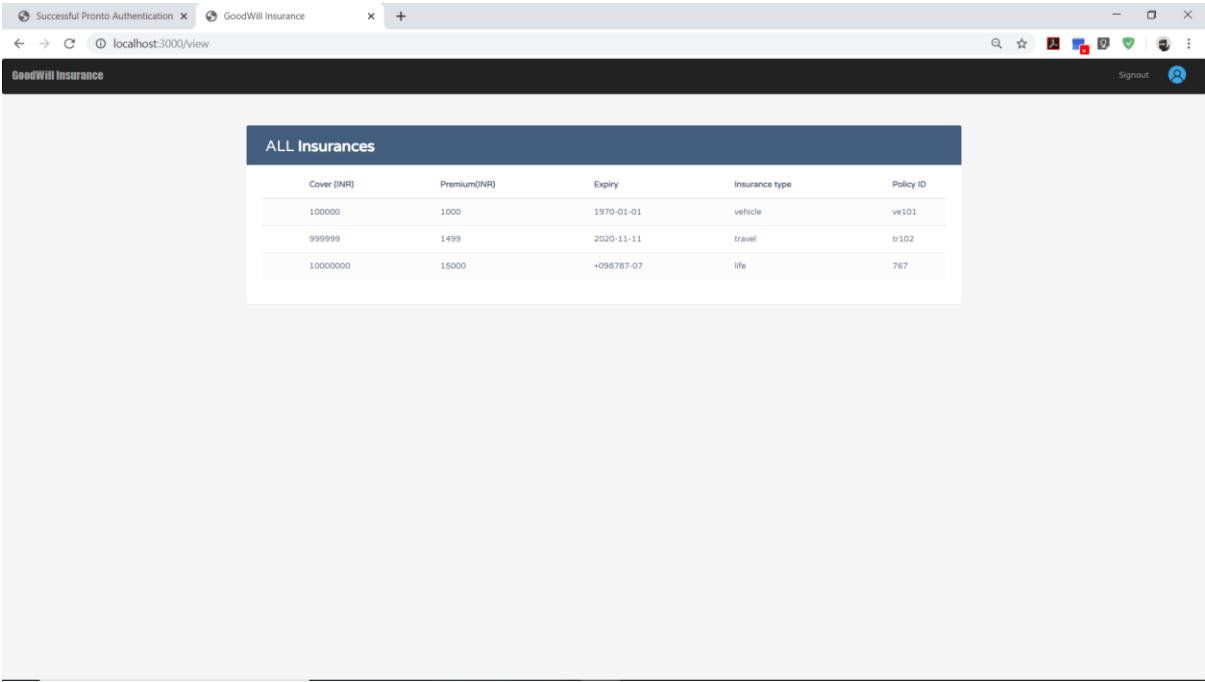
Home page for customer



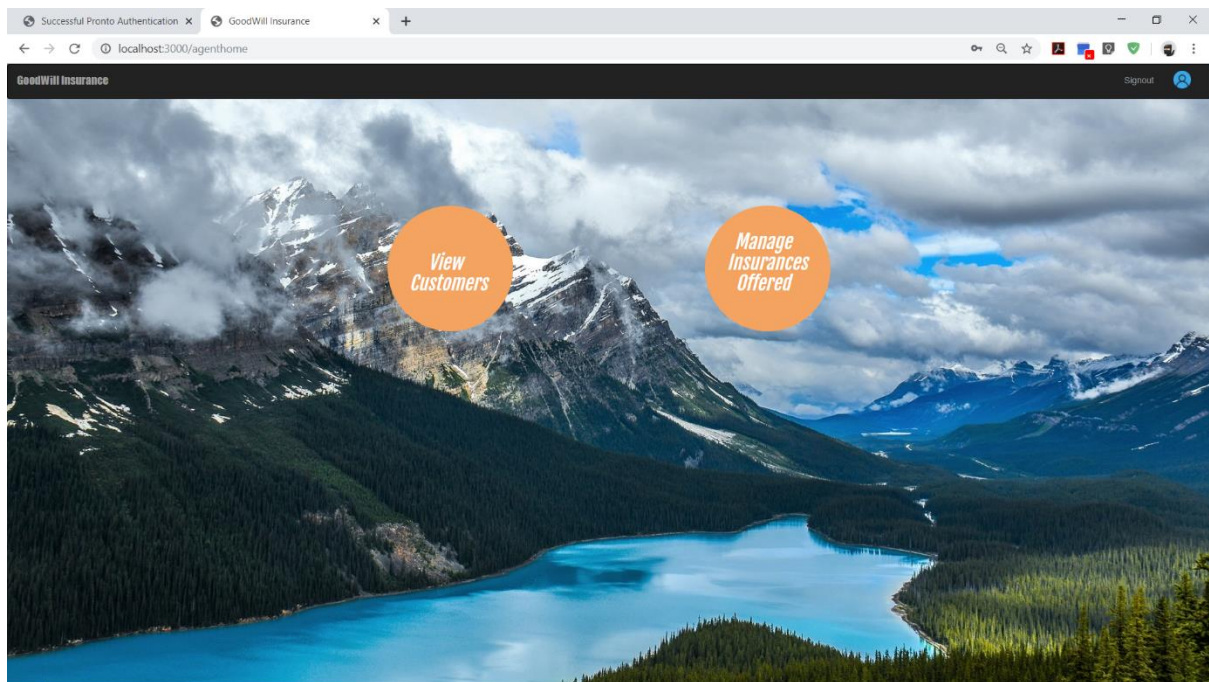
Choosing new insurance for customer



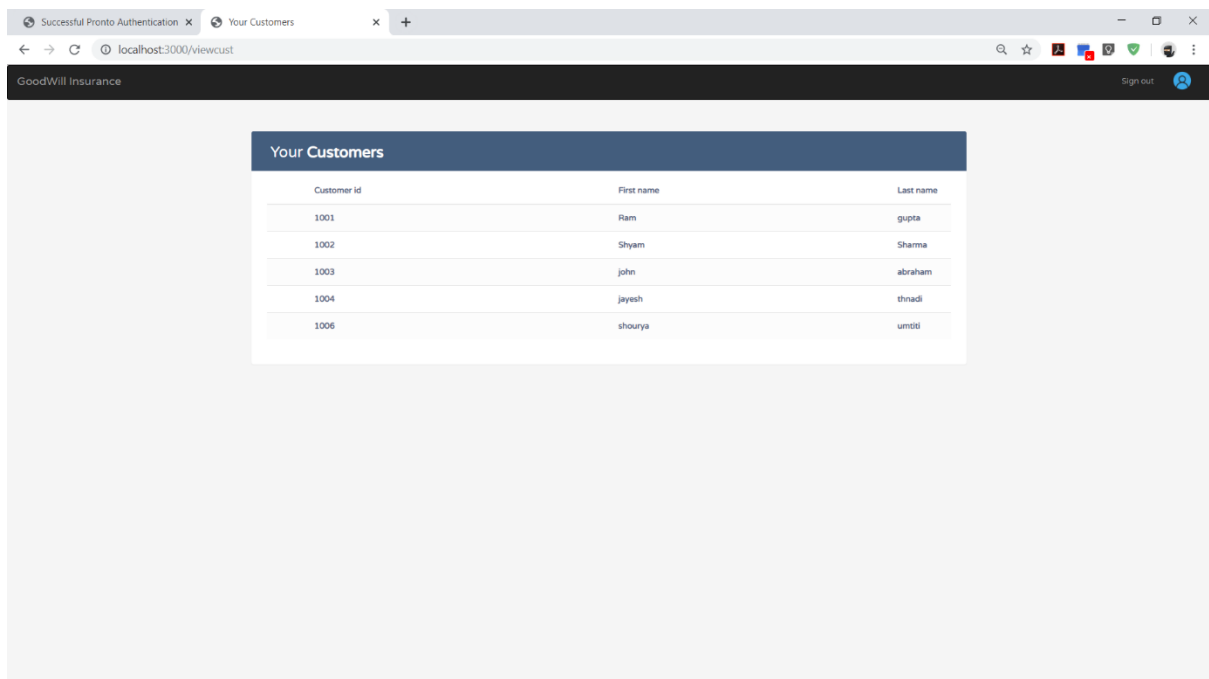
Viewing current insurances-customer



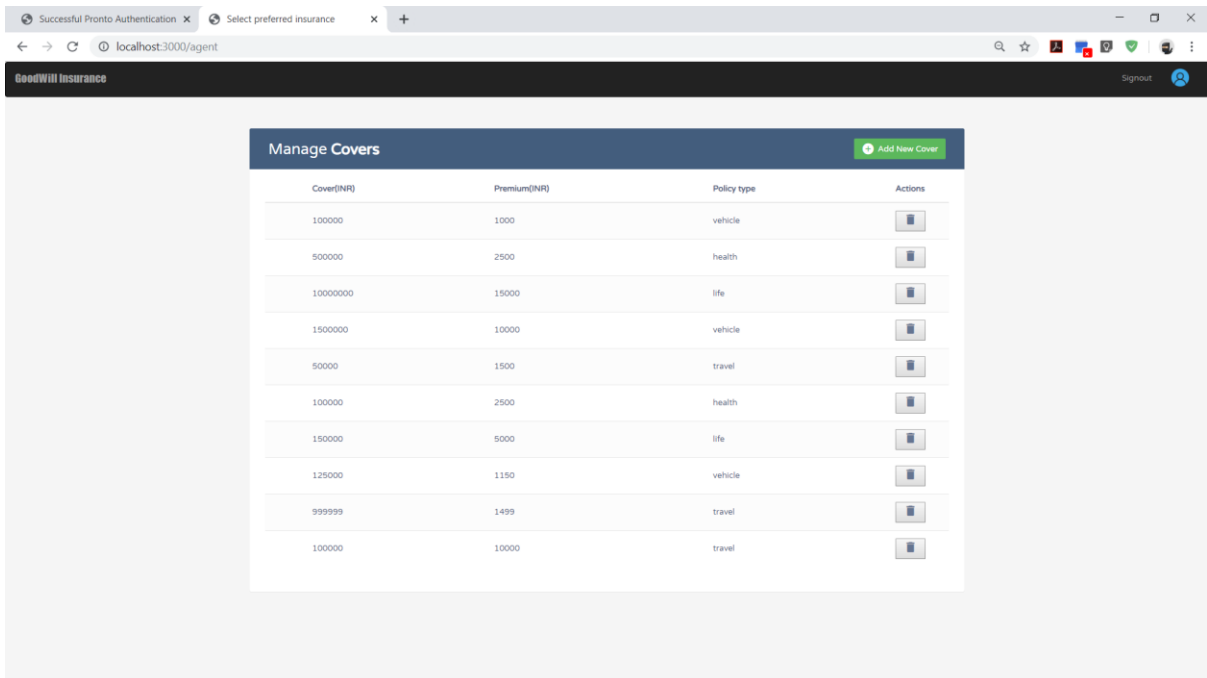
Agent home page- customer



Viewing current insurances-agent



Manage insurances page-agent

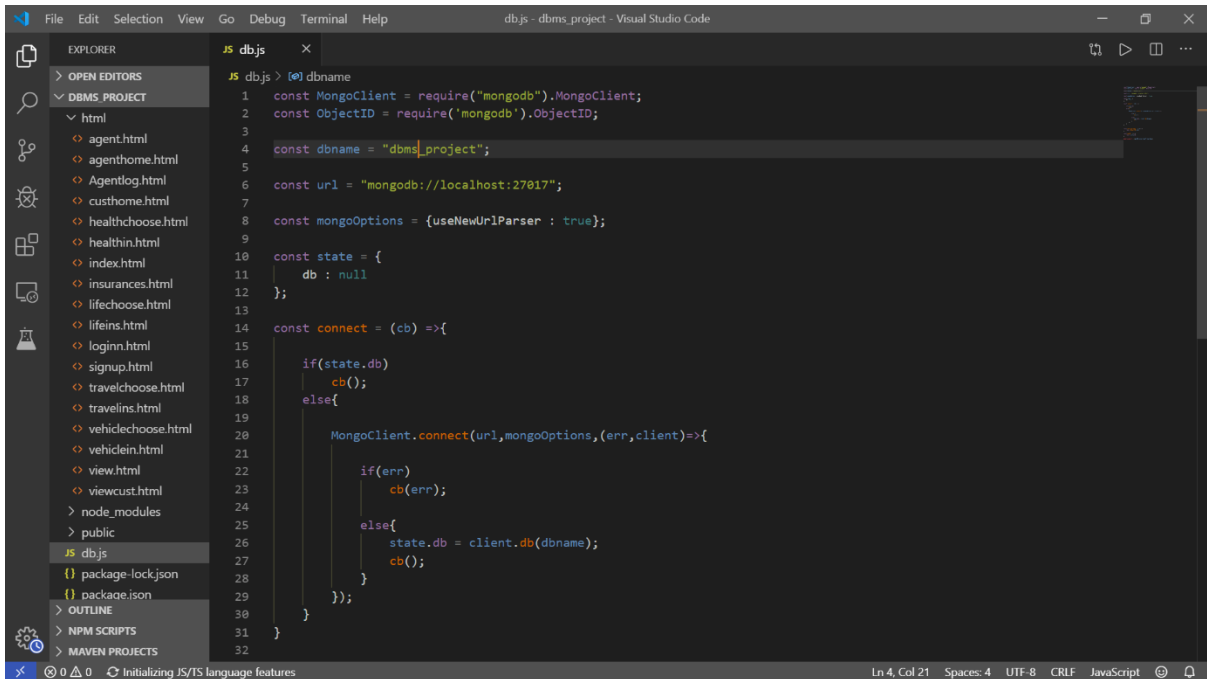


The screenshot displays the 'Manage Covers' interface of the GoodWill Insurance application. The page features a table with the following data:

Cover(INR)	Premium(INR)	Policy type	Actions
100000	1000	vehicle	[Delete Icon]
500000	2500	health	[Delete Icon]
10000000	15000	life	[Delete Icon]
1500000	10000	vehicle	[Delete Icon]
50000	1500	travel	[Delete Icon]
100000	2500	health	[Delete Icon]
150000	5000	life	[Delete Icon]
125000	1150	vehicle	[Delete Icon]
999999	1499	travel	[Delete Icon]
100000	10000	travel	[Delete Icon]

A green button labeled 'Add New Cover' is located in the top right corner of the table area. The browser's address bar shows 'localhost:3000/agent'.

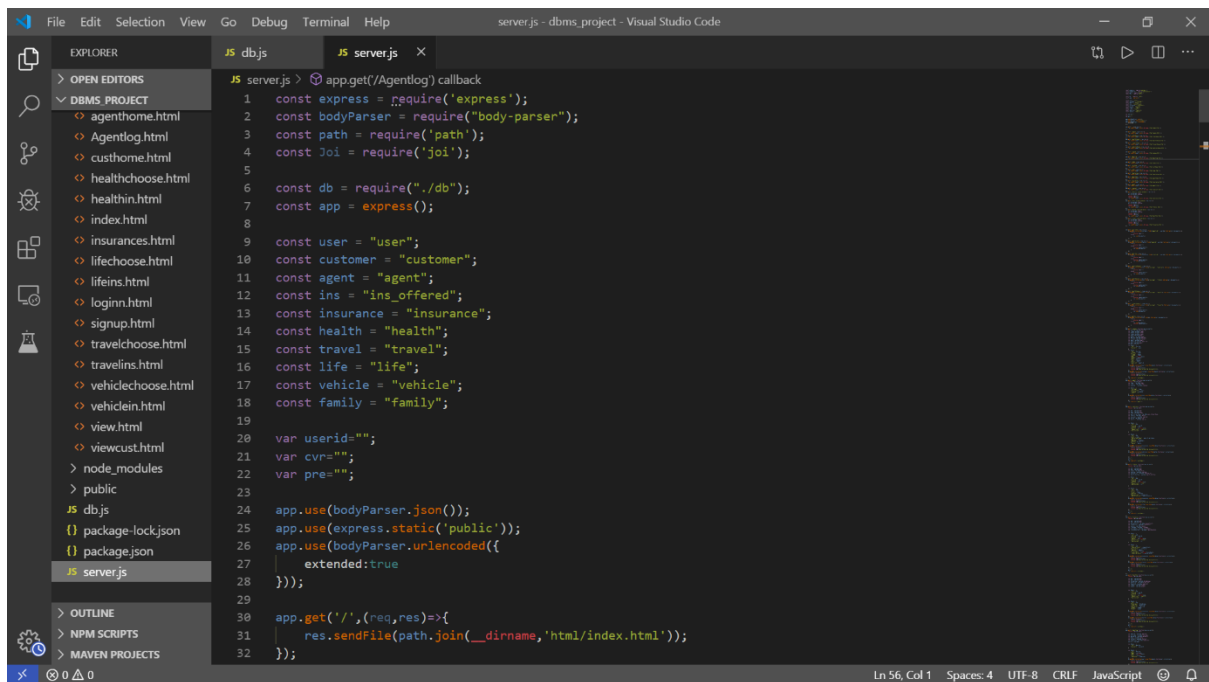
Server connectivity JavaScript file



```
1 const MongoClient = require("mongodb").MongoClient;
2 const ObjectId = require('mongodb').ObjectId;
3
4 const dbname = "dbms_project";
5
6 const url = "mongodb://localhost:27017";
7
8 const mongoOptions = {useNewUrlParser : true};
9
10 const state = {
11   db : null
12 };
13
14 const connect = (cb) =>{
15
16   if(state.db)
17     cb();
18   else{
19     MongoClient.connect(url,mongoOptions,(err,client)=>{
20
21       if(err)
22         cb(err);
23
24       else{
25         state.db = client.db(dbname);
26         cb();
27       }
28     });
29   }
30 }
31
32
```

The screenshot shows the Visual Studio Code editor with the 'db.js' file open. The Explorer sidebar on the left displays the project structure, including a 'public' folder containing 'db.js'. The status bar at the bottom indicates 'Ln 4, Col 21' and 'Spaces: 4'.

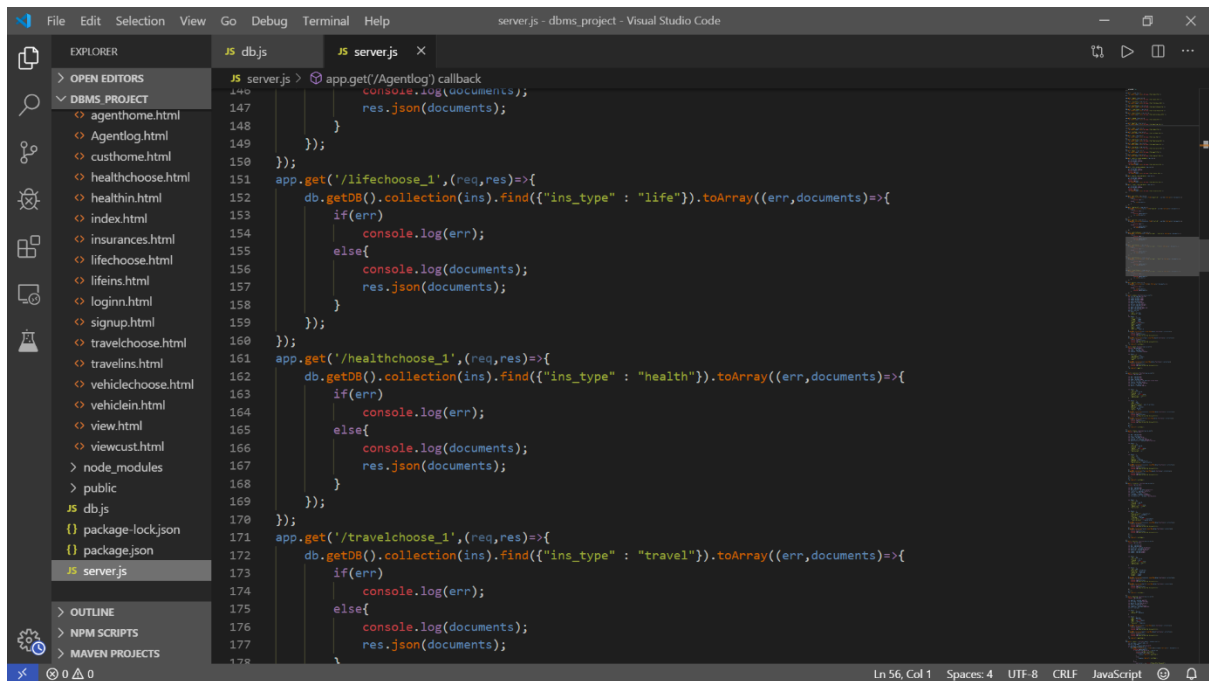
Server JavaScript file



The screenshot shows the Visual Studio Code interface with the 'server.js' file open in the editor. The Explorer sidebar on the left shows the project structure, including 'dbms_project' and 'node_modules'. The editor displays the following JavaScript code:

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const path = require('path');
4 const Joi = require('joi');
5
6 const db = require('./db');
7 const app = express();
8
9 const user = "user";
10 const customer = "customer";
11 const agent = "agent";
12 const ins = "ins_offered";
13 const insurance = "insurance";
14 const health = "health";
15 const travel = "travel";
16 const life = "life";
17 const vehicle = "vehicle";
18 const family = "family";
19
20 var userid="";
21 var cvr="";
22 var pre="";
23
24 app.use(bodyParser.json());
25 app.use(express.static('public'));
26 app.use(bodyParser.urlencoded({
27   extended:true
28 }));
29
30 app.get('/',(req,res)=>{
31   res.sendFile(path.join(__dirname,'html/index.html'));
32 });
```

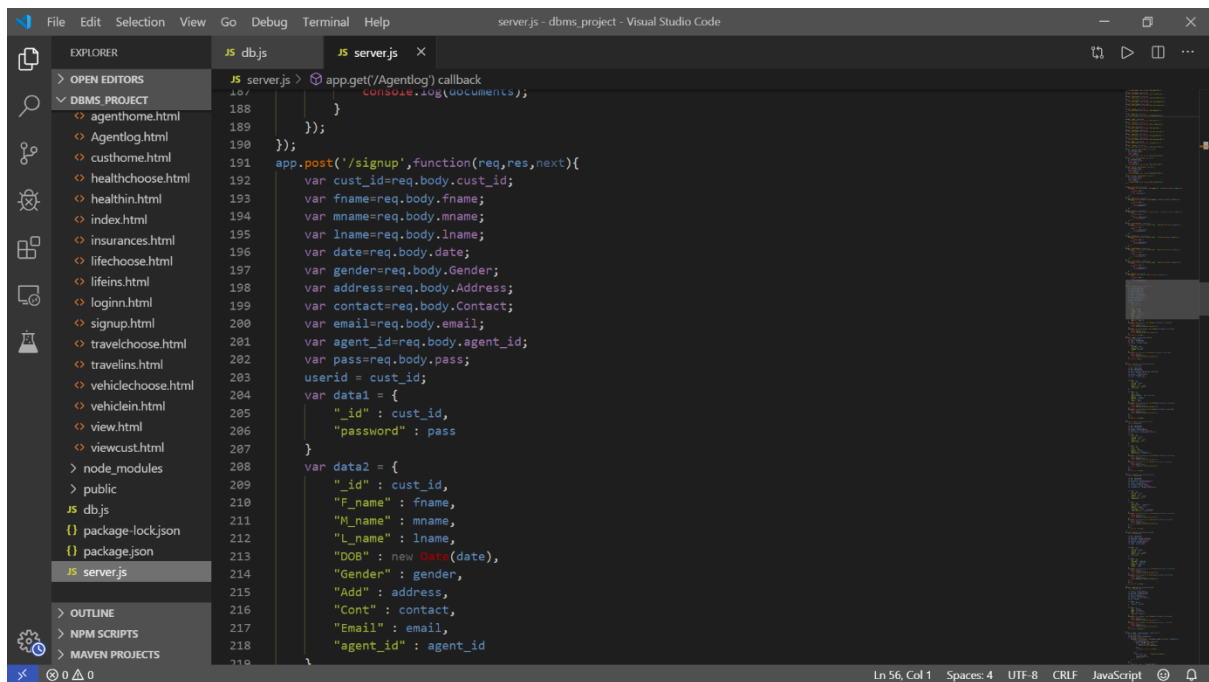
Inserting data into database -JavaScript



The screenshot shows the Visual Studio Code interface with the 'server.js' file open in the editor. The editor displays the following JavaScript code, which includes database insertion logic:

```
146 console.log(documents);
147 res.json(documents);
148 }
149 });
150
151 app.get('/lifechoose_1',(req,res)=>{
152   db.getDB().collection(ins).find({"ins_type" : "life"}).toArray((err,documents)=>{
153     if(err)
154       console.log(err);
155     else{
156       console.log(documents);
157       res.json(documents);
158     }
159   });
160 });
161
162 app.get('/healthchoose_1',(req,res)=>{
163   db.getDB().collection(ins).find({"ins_type" : "health"}).toArray((err,documents)=>{
164     if(err)
165       console.log(err);
166     else{
167       console.log(documents);
168       res.json(documents);
169     }
170   });
171 });
172
173 app.get('/travelchoose_1',(req,res)=>{
174   db.getDB().collection(ins).find({"ins_type" : "travel"}).toArray((err,documents)=>{
175     if(err)
176       console.log(err);
177     else{
178       console.log(documents);
179       res.json(documents);
180     }
181   });
182 });
```

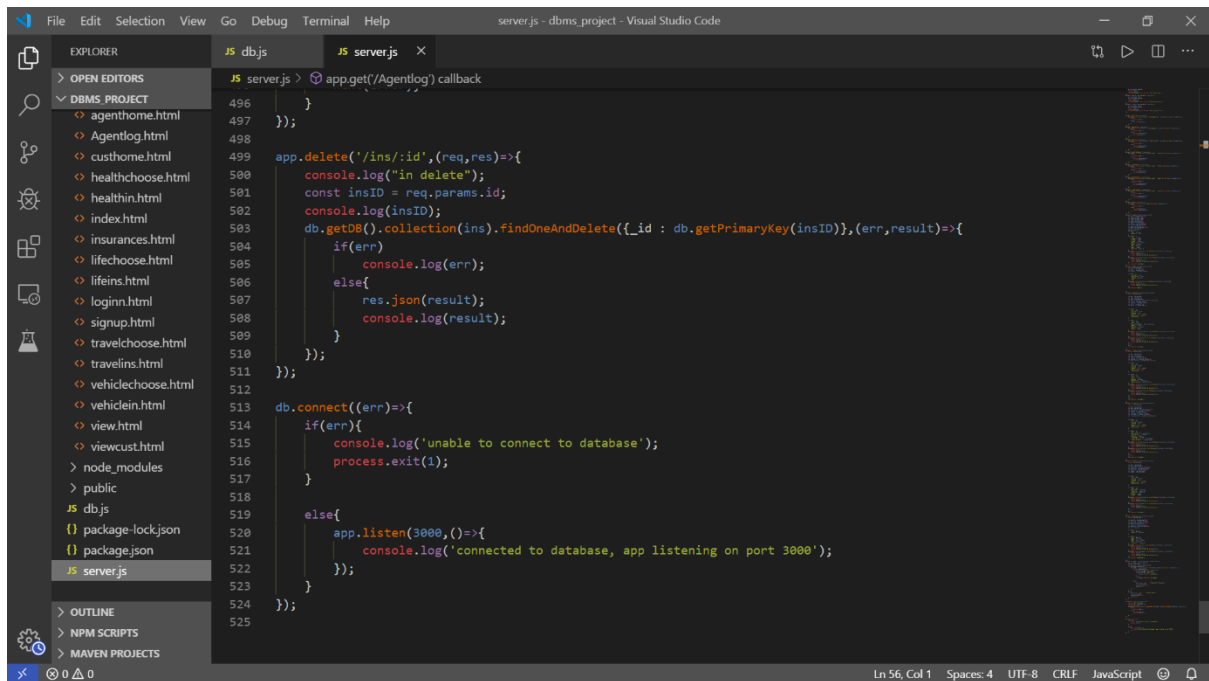
Posting data into database-JavaScript



The screenshot shows the Visual Studio Code editor with a project named 'server.js - dbms_project'. The Explorer sidebar on the left shows a file tree with 'db.js' selected. The main editor window displays the code for 'db.js', which includes a POST endpoint for user registration. The code uses the 'mongoose' library to interact with a MongoDB database. It defines a schema for a user with fields like 'cust_id', 'fname', 'lname', 'date', 'gender', 'address', 'contact', 'email', and 'agent_id'. The 'signup' endpoint takes a request and response object and inserts a new user into the 'users' collection.

```
188 app.get('/Agentlog') callback
189 console.log(documents);
190 });
191 app.post('/signup',function(req,res,next){
192   var cust_id=req.body.cust_id;
193   var fname=req.body.fname;
194   var lname=req.body.lname;
195   var date=req.body.date;
196   var gender=req.body.Gender;
197   var address=req.body.Address;
198   var contact=req.body.Contact;
199   var email=req.body.email;
200   var agent_id=req.body.agent_id;
201   var pass=req.body.pass;
202   userid = cust_id;
203   var data1 = {
204     "_id" : cust_id,
205     "password" : pass
206   }
207   var data2 = {
208     "_id" : cust_id,
209     "F_name" : fname,
210     "M_name" : lname,
211     "L_name" : lname,
212     "DOB" : new Date(date),
213     "Gender" : gender,
214     "Add" : address,
215     "Cont" : contact,
216     "Email" : email,
217     "agent_id" : agent_id
218   }
219 }
```

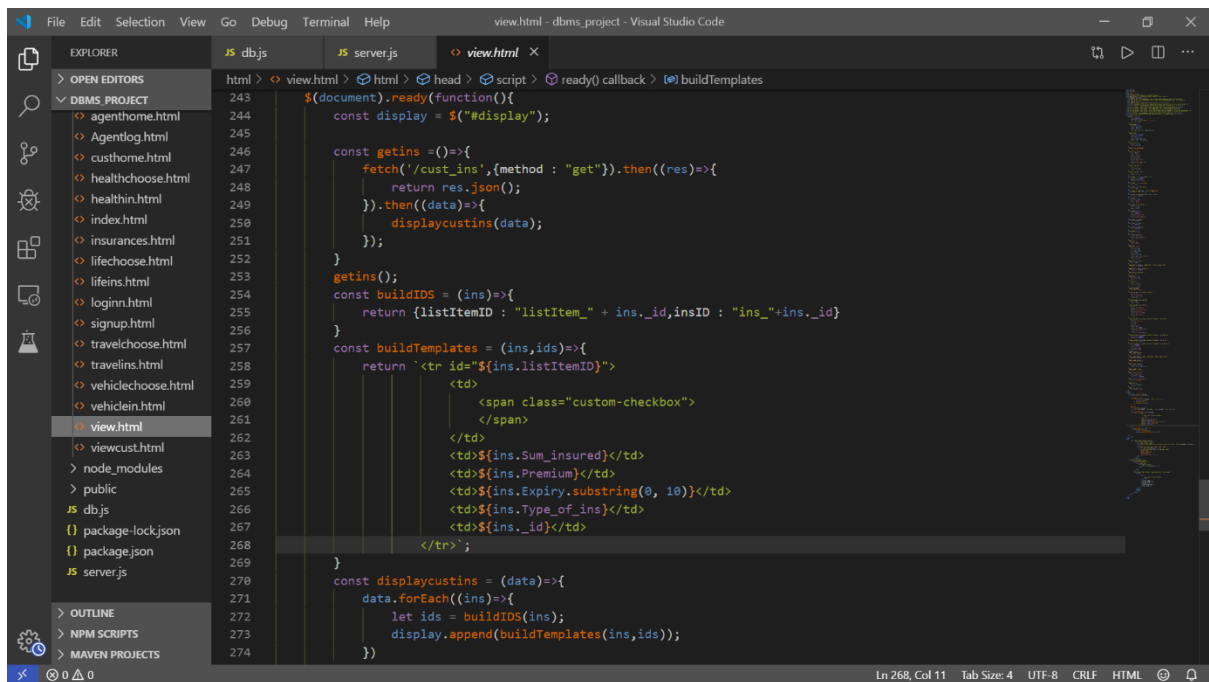
Connection to database -JavaScript



The screenshot shows the Visual Studio Code editor with the same project. The Explorer sidebar shows 'db.js' selected. The main editor window displays the code for 'db.js', which includes a DELETE endpoint for deleting a user and a database connection setup. The code uses the 'mongoose' library to connect to a MongoDB database. It defines a schema for a user and a 'delete' endpoint that takes a request and response object and deletes a user from the 'users' collection. The 'connect' function is used to establish a connection to the database, and the 'listen' method is used to start the server on port 3000.

```
496 app.get('/Agentlog') callback
497 console.log(documents);
498 });
499 app.delete('/ins/:id',(req,res)=>{
500   console.log("in delete");
501   const insID = req.params.id;
502   console.log(insID);
503   db.getDB().collection(ins).findOneAndDelete({_id : db.getPrimaryKey(insID)}),(err,result)=>{
504     if(err){
505       console.log(err);
506     }else{
507       res.json(result);
508       console.log(result);
509     }
510   });
511 });
512 db.connect((err)=>{
513   if(err){
514     console.log('Unable to connect to database');
515     process.exit(1);
516   }
517   else{
518     app.listen(3000,()=>{
519       console.log('connected to database, app listening on port 3000');
520     });
521   }
522 });
523
524
525 }
```


Getting input-HTML

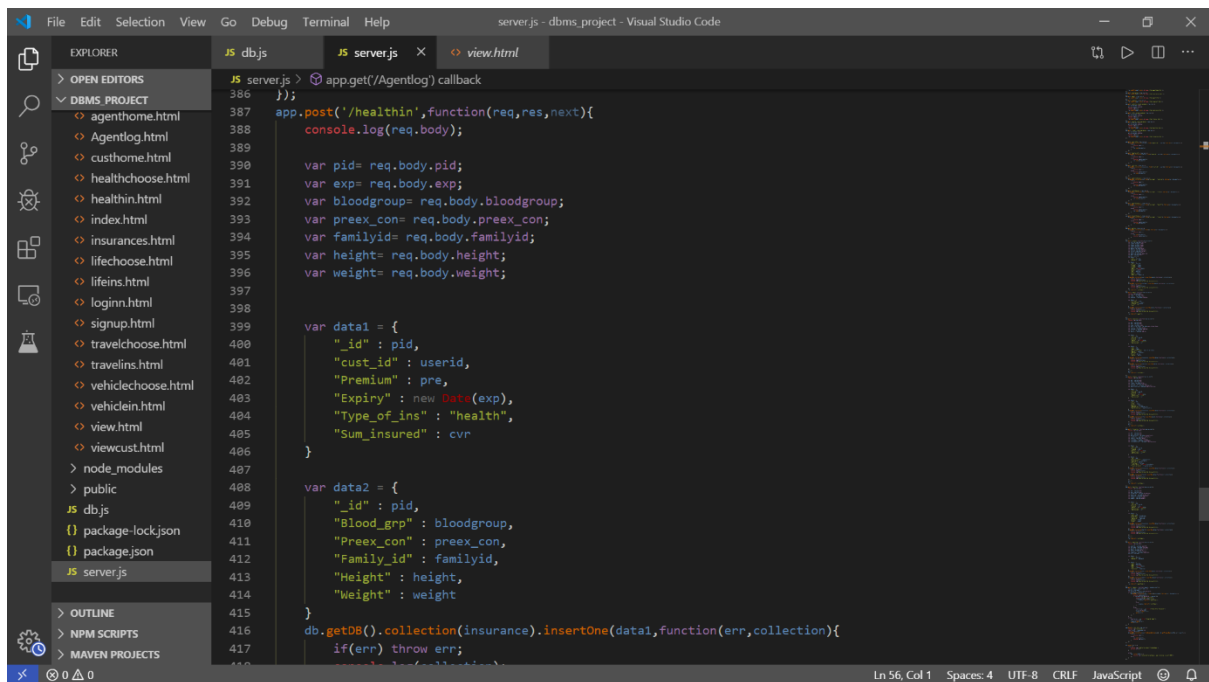


```
view.html - dbms_project - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER
> OPEN EDITORS
  DBMS_PROJECT
    agenthome.html
    Agentlog.html
    custhome.html
    healthchoose.html
    healthin.html
    index.html
    insurances.html
    lifechoose.html
    lifeins.html
    login.html
    signup.html
    travelchoose.html
    travelins.html
    vehiclechoose.html
    vehiclein.html
  view.html
  viewcust.html
  node_modules
  public
  JS db.js
  package-lock.json
  package.json
  JS server.js
  OUTLINE
  NPM SCRIPTS
  MAVEN PROJECTS

view.html
243 $(document).ready(function){
244   const display = $("#display");
245
246   const getins = ()=>{
247     fetch('/cust_ins',{method : "get"}).then((res)=>{
248       return res.json();
249     }).then((data)=>{
250       displaycustins(data);
251     });
252   }
253   getins();
254   const buildIDS = (ins)=>{
255     return {listItemID : "listItem_" + ins._id,insID : "ins_"+ins._id}
256   }
257   const buildTemplates = (ins,ids)=>{
258     return `<tr id="${ins.listItemID}">
259       <td>
260         <span class="custom-checkbox">
261           </span>
262         </td>
263         <td>${ins.Sum_insured}</td>
264         <td>${ins.Premium}</td>
265         <td>${ins.Expiry.substring(0, 10)}</td>
266         <td>${ins.Type_of_ins}</td>
267         <td>${ins._id}</td>
268       </tr>`;
269   }
270   const displaycustins = (data)=>{
271     data.forEach((ins)=>{
272       let ids = buildIDS(ins);
273       display.append(buildTemplates(ins,ids));
274     })
  }
```



```
server.js - dbms_project - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER
> OPEN EDITORS
  DBMS_PROJECT
    agenthome.html
    Agentlog.html
    custhome.html
    healthchoose.html
    healthin.html
    index.html
    insurances.html
    lifechoose.html
    lifeins.html
    login.html
    signup.html
    travelchoose.html
    travelins.html
    vehiclechoose.html
    vehiclein.html
  view.html
  viewcust.html
  node_modules
  public
  JS db.js
  package-lock.json
  package.json
  JS server.js
  OUTLINE
  NPM SCRIPTS
  MAVEN PROJECTS

server.js
386 });
387 app.get('/Agentlog',function(req,res,next){
388   console.log(req.body);
389
390   var pid= req.body.pid;
391   var exp= req.body.exp;
392   var bloodgroup= req.body.bloodgroup;
393   var preex_con= req.body.preex_con;
394   var familyid= req.body.familyid;
395   var height= req.body.height;
396   var weight= req.body.weight;
397
398   var data1 = {
399     "_id" : pid,
400     "cust_id" : userid,
401     "Premium" : pre,
402     "Expiry" : new Date(exp),
403     "Type_of_ins" : "health",
404     "Sum_insured" : cvr
405   }
406
407   var data2 = {
408     "_id" : pid,
409     "Blood_grp" : bloodgroup,
410     "Preex_con" : preex_con,
411     "Family_id" : familyid,
412     "Height" : height,
413     "Weight" : weight
414   }
415
416   db.getDB().collection(insurance).insertOne(data1,function(err,collection){
417     if(err) throw err;
418     console.log(collection);
  }
```