In this assignment you will create the database and connect it to your web application.

Description:
Same as assignment 1.

Additional Details:

Database must include following tables:
- UserCredentials (ID & password), password should be encrypted.
- ClientInformation
- FuelQuote
- Any additional tables you feel, like States.

Important deliverables:
- You should have validations in place for required fields, field types, and field lengths.
- Backend should retrieve data from DB and display it to front end.
- Form data should be populated from the backend. Backend should receive data from front end, validate, and persist to DB.
- Any new code added should be covered by unit tests. Keep code coverage above 80%.

- **NOTE:** Only provide a word / pdf doc. You should use GitHub for your group collaboration and code.

Answer these questions:
1. Provide link to GitHub repository for TAs to view the code.(5 points)
2. Provide SQL statements to create database.(3 points)
3. Rerun the code coverage report and provide it. (2 points)
4. IMPORTANT: list who did what within the group. TAs should be able to validate in GitHub, otherwise team members who didn't contribute will receive a ZERO.

1. ***Provide link to GitHub repository for TAs to view the code. (5 points)***

- *https://github.com/Rawbethy/COSC4353Group42*

2. ***Provide SQL statements to create database. (3 points)***
- We decided to use MongoDB for our database so we'll provide the schemas implemented. Using MongoDB allowed us to use the

mongoose library to define all of the schemas for our database. With mongoose, we were able to put the validations in place for all the data; it allowed use of middleware functions to work on the validations. Mongoose also works well with Node.js which we used for the backend development of our application.

- user.js for User Credentials:

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema( {
    username: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        minLength: 3
    },
    email: {
        type: String,
        required: true,
        unique: true,
        trim: true
    },
    password: {
        type: String,
        required: true,
        unique: false,
        trim: true,
        minLength: 8
    },
    userInfo: {
        fullName: {
            type: String
        },
        address1: {
            type: String
        },
        address2: {
            type: String
        },
        city: {
            type: String
        },
        state: {
            type: String,
            minLength: 2
        },
        zipcode: {
            type: String,
            minLength: 5,
            maxlength: 9
        },
        phoneNum: {
            type: String,
            minLength: 9
        },
        email: {
            type: String,
            minLength: 9
        }
    }
}, {
    timestamps: true
});

userSchema.pre('save', async function(next) {
    try {
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(this.password, salt);
        this.password = hashedPassword;
        next();
    }
    catch(err) {
        next(err);
    }
})
```

```
const User = mongoose.model('User', userSchema);

module.exports = User;
```

- profileInfo.js for Client Information:

```
const mongoose = require('mongoose');

const profileSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        unique: true
    },
    fullName: {
        type: String,
        required: true
    },
    address1: {
        type: String,
        required: true
    },
    address2: {
        type: String,
        required: true
    },
    city: {
        type: String,
        required: true
    },
    state: {
        type: String,
        required: true,
        minLength: 2
    },
    zip: {
        type: String,
        required: true,
        minLength: 5,
        maxlength: 9
    },
    phone: {
        type: String,
        required: true,
        minLength: 9
    },
    email: {
        type: String,
        required: true,
        minLength: 9
    }
}, {
    timestamps: true
});

const profileInfo = mongoose.model('Profile', profileSchema);
module.exports = profileInfo;
```

- quotes.js for Fuel Quote:

```
const mongoose = require('mongoose');
const quoteSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true
    },
    quotes: [{
        address: {
            type: String,
            required: true
        },
        deliveryDate: {
            type: Date,
            required: true
        },
        gallonsReq: {
            type: Number,
            required: true
        },
        pricePerGallon: {
            type: Number,
            required: true
        },
        total: {
            type: Number,
            required: true
        }
    }]
}, {
    timestamps: true
});

const Quote = mongoose.model('Quote', quoteSchema);
module.exports = Quote;
```

- Current pricing.js for the pricing module:

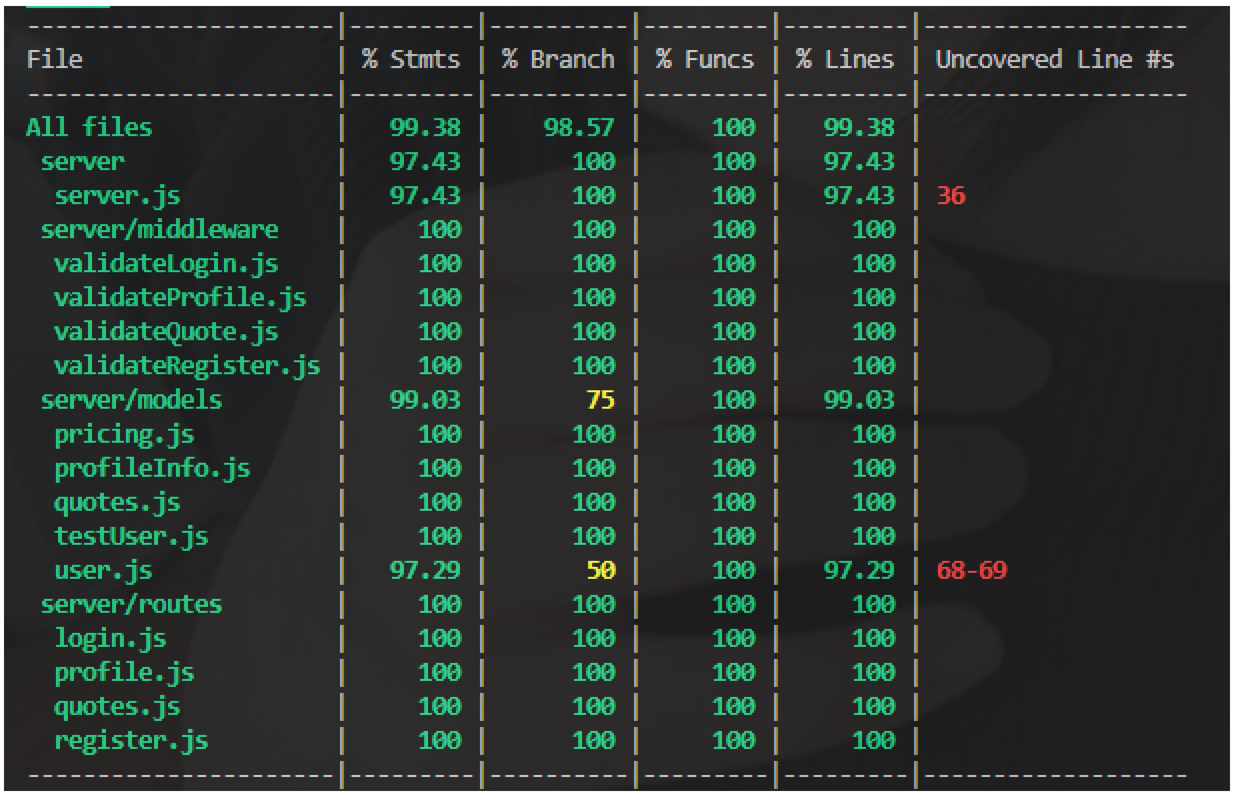
```
    class PricingModule {
        constructor(address, deliveryDate, gallonsReq, pricePerGallon, total){
            this.address = address;
            this.deliveryDate = deliveryDate;
            this.gallonsReq = gallonsReq;
            this.pricePerGallon = pricePerGallon;
            this.total = total;
        }

        totalPrice() {
            return this.total;
        }
        // calculatePrice(){
            //do calculation for pricing here
        // }

    }

    module.exports = PricingModule;
```

3. Rerun the code coverage report and provide it. (2 points)



```
----------------------|---------|----------|---------|---------|-------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------
All files             |   99.38 |    98.57 |     100 |   99.38 |
 server               |   97.43 |      100 |     100 |   97.43 |
  server.js           |   97.43 |      100 |     100 |   97.43 | 36
 server/middleware    |     100 |      100 |     100 |     100 |
  validateLogin.js    |     100 |      100 |     100 |     100 |
  validateProfile.js  |     100 |      100 |     100 |     100 |
  validateQuote.js    |     100 |      100 |     100 |     100 |
  validateRegister.js |     100 |      100 |     100 |     100 |
 server/models        |   99.03 |       75 |     100 |   99.03 |
  pricing.js          |     100 |      100 |     100 |     100 |
  profileInfo.js      |     100 |      100 |     100 |     100 |
  quotes.js           |     100 |      100 |     100 |     100 |
  testUser.js         |     100 |      100 |     100 |     100 |
  user.js             |   97.29 |       50 |     100 |   97.29 | 68-69
 server/routes        |     100 |      100 |     100 |     100 |
  login.js            |     100 |      100 |     100 |     100 |
  profile.js          |     100 |      100 |     100 |     100 |
  quotes.js           |     100 |      100 |     100 |     100 |
  register.js         |     100 |      100 |     100 |     100 |
----------------------|---------|----------|---------|---------|-------------------
```

4.

Group Member Name	What is your contribution?	Discussion Notes
Saim Ali	Pricing module class, bug fixes, database population	
Robert Duque	Creating database and config, connecting frontend/backend to database, various tables from backend such as order history	
Muhaimin Badar	validator middleware for data from database and corresponding unit tests	
Vy Nguyen	Provided MongoDB schemas in report Q#2, made minor updates to validation/error messages	Provided short justification for using MongoDB for database management