Criterion C: Development

This project was developed using the Node JS JavaScript framework with help of the Visual Studio Code IDE.

Techniques that were used in the development process:

1. GUI Elements created using Headers, CSS stylesheets, and DIV classes
2. Creating responsive pop-up forms using JavaScript functions
3. Dynamic creation of HTML elements using EJS (embedded JavaScript)
4. Asynchronous Programming
5. Serialization/Deserialization of Objects
6. Password Authentication and Storage
7. Role-Based Access Control System (RBAC)
8. External Libraries
9. Relational Databases
10. SQL commands used
11. User-defined Tree Data Structures for Predictive Search (Trie)
12. Sending Mails through SMTP and Node-Mailer

GUI Elements created using Headers, CSS stylesheets, and DIV classes

In order to design an appealing UI for my client, I employed CSS Stylesheets to better organize my code. Each of the 'div' classes such as '.sidebar' in the stylesheet reduced code redundancy of my ejs/html files. The stylesheet enables easier development as only one document needs to be changed to alter GUI elements.

```css
/* Initializing sidebar view */
.sidebar {
  height: 100%;
  min-width: 15%;
  width: 20%;
  max-width: 50%;
  position: fixed;
  z-index: 1;
  top: 0;
  left: 0;
  background-color: #164357;
  overflow-x: hidden;
  padding-top: 20px;
}

.sidebar a {
  padding: 6px 8px 6px 16px;
  text-decoration: none;
  font-size: 25px;
  text-align: center;
  color: #f1f1f1;
  display: block;
}

.sidebar a:hover {
  color: yellow;
}

/* Style page content */
.main {
  margin-left: 160px; /* Same as the width of the sidebar */
  padding: 0px 10px;
  align-items: center;
```

CSS classes used to structure code for improved extensibility and modularity

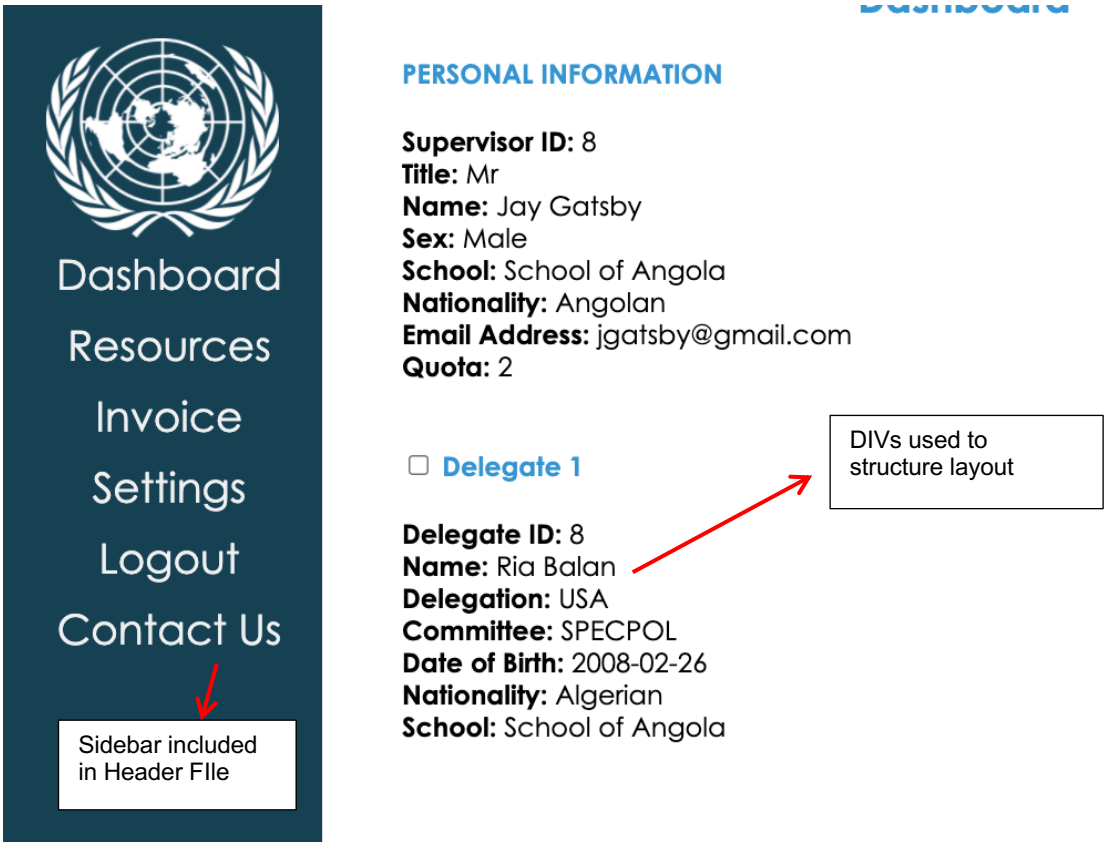
```html
<link rel="stylesheet" type= "text/css" href="../../css/style2.css"/>
```

```
<%- include('./layouts/header') %>
```

Reduced redundancy as only stylesheet and HTML header file are linked rather than repeating code.

```html
<div class="sidebar">
    <img class="centerimg" src = "../../img/logo.svg" alt="CISMUN logo" width="120" height="120" />
    <a href="/">Dashboard</a>
    <% if (user.role == 'CLIENT') { %>
        <a href="/resources">Resources</a>
        <a href="/settings">Settings</a>
        <a href="/generate-invoice">Create Invoice</a>
    <% } %>
    <% if (user.role == 'ADMIN') { %>
        <a href="/admin">Manage</a>
        <a href="/delegates">Delegates</a>
    <% } %>
    <a href="#" onclick="logout()">Logout</a>
</div>
```

CSS classes used to reduce code redudancy



Creating responsive pop-up forms using JavaScript functions

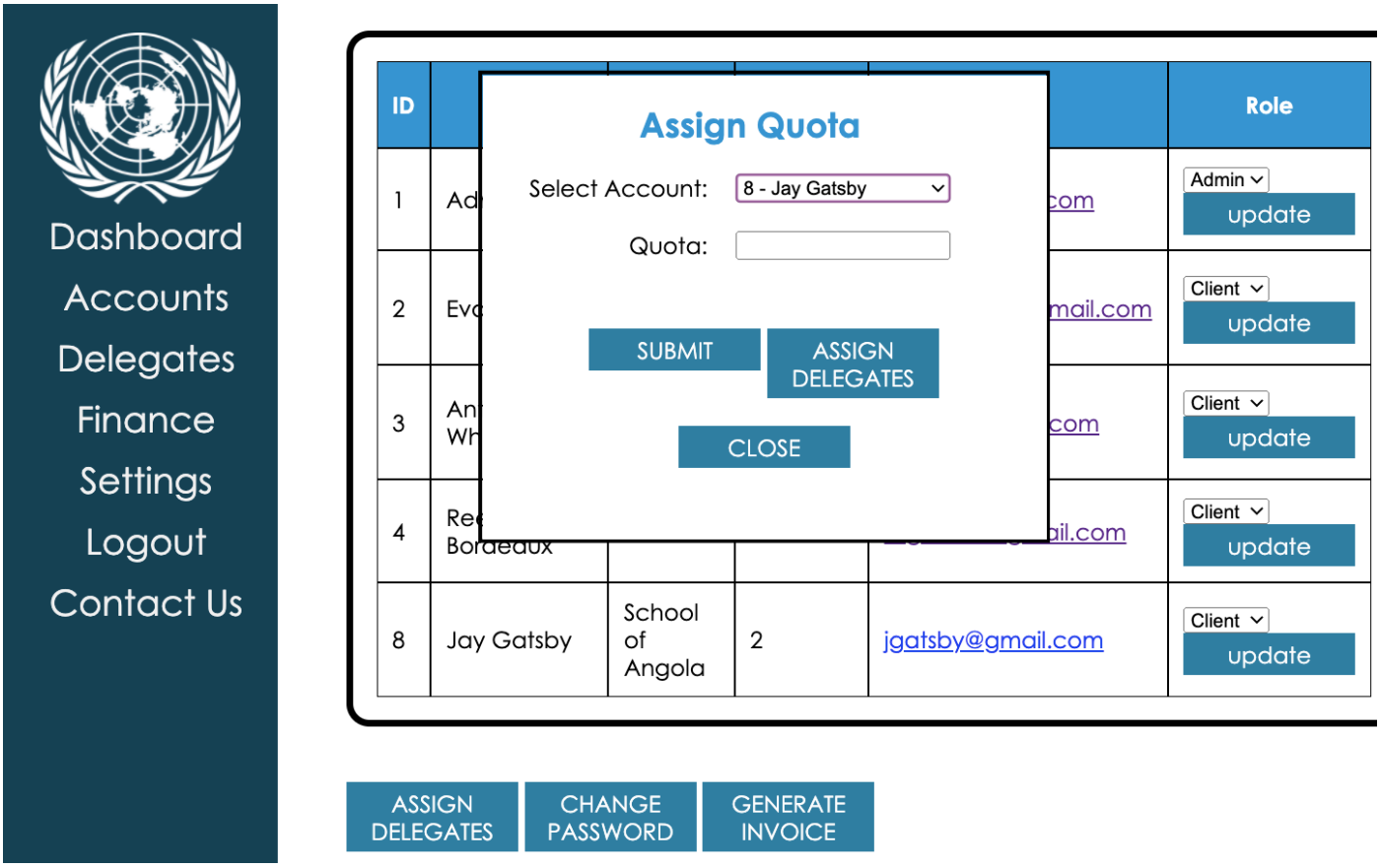
Since there is a need for many forms in the program for the client and users to access and modify the database, it's vital that there is a balance of functionality and accessibility. Hence, rather than creating separate pages for each form, I created JavaScript functions such as 'openForm()' and 'closeForm()' to display a pop up form that is shown when the user presses on the specified button. These functions displayed and hid a 'div', which contained the form details, with the help of the display style attribute when called.

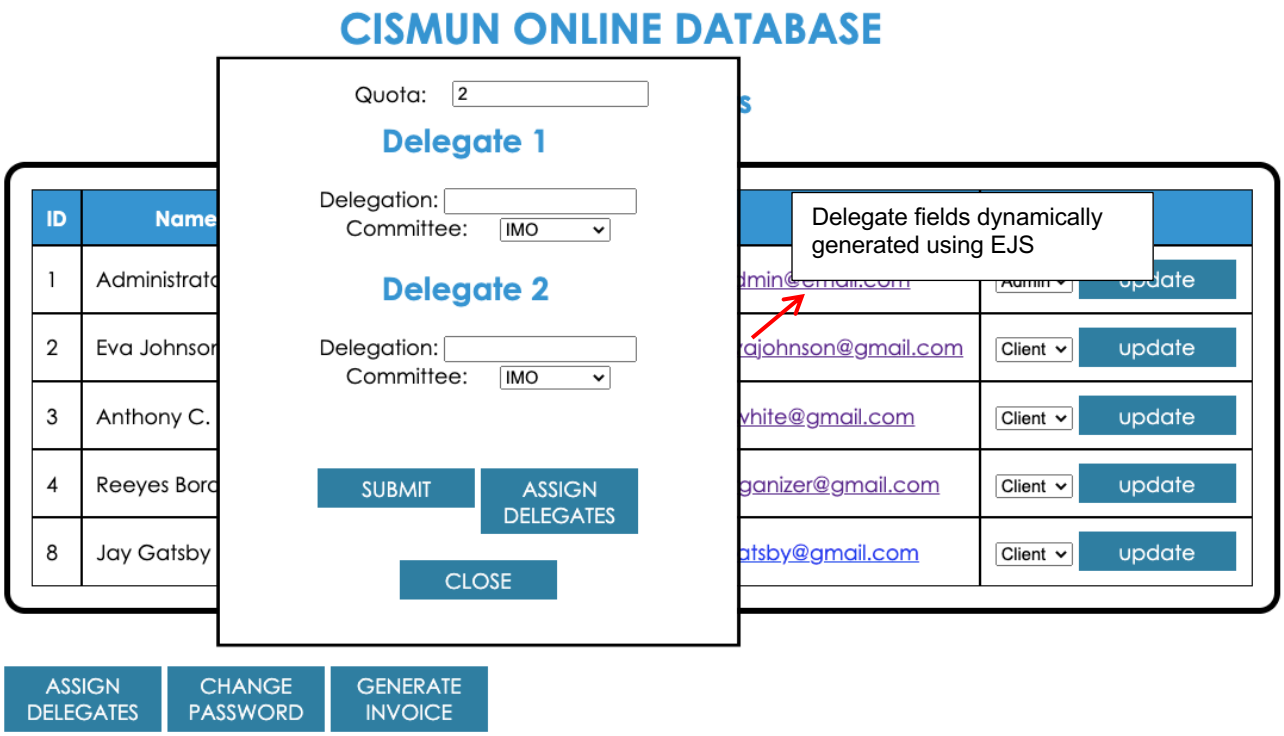
```
<script>
    function openForm(n) {
        document.getElementById("popupForm"+ n).style.display = "block";
    }

  function closeForm(n) {
    document.getElementById("popupForm" +n).style.display = "none";
  }
```

Parameterised functions to improve extensibility as one function is used to open and close all popup forms

Form is opened and closed by toggling display style attribute of a DIV

```html
137     <button class="button" onclick="openForm()" style="max-width: 100%, ">ASSIGN DELEGATES</button>
138     <button class="button" onclick="openForm('2')" style="max-width: 100%, ">CHANGE PASSWORD</button>
139     <button class="button" onclick="openForm('3')" style="max-width: 100%, ">GENERATE INVOICE</button>
140
141 <div class="loginPopup">…
169 </div>
170 <div class="loginPopup">
171     <div class="formPopup" id="popupForm2">
172       <form action="/settings/change-password" method="POST">
173           <h2 style="text-align:center;">Change Password</h2>
174           <div class="container" id="fields" >
175                 <label for="id">Select Account: </label>
176                 <select class="input" style="height:20px;" name="id" required>…
180                 </select>
181                 <label for="password">New Password: </label>
182                 <input class="input" type="password" id="password" name="password" required>
183                 <label for="confirmPassword">Confirm Password: </label>
184                 <input class="input" type="password" id="confirmPassword" name="confirmPassword" required>
185                 <br><br>
186
187               </div>
188           <br><br>
189           <br><br>
190           <button class="button" type="submit">CHANGE</button>
191           <button type="button" class="button" onclick="closeForm('2')">CLOSE</button>
192           <br><br>
193           <br><br>
194
195       </form>
196     </div>
197 </div>
```



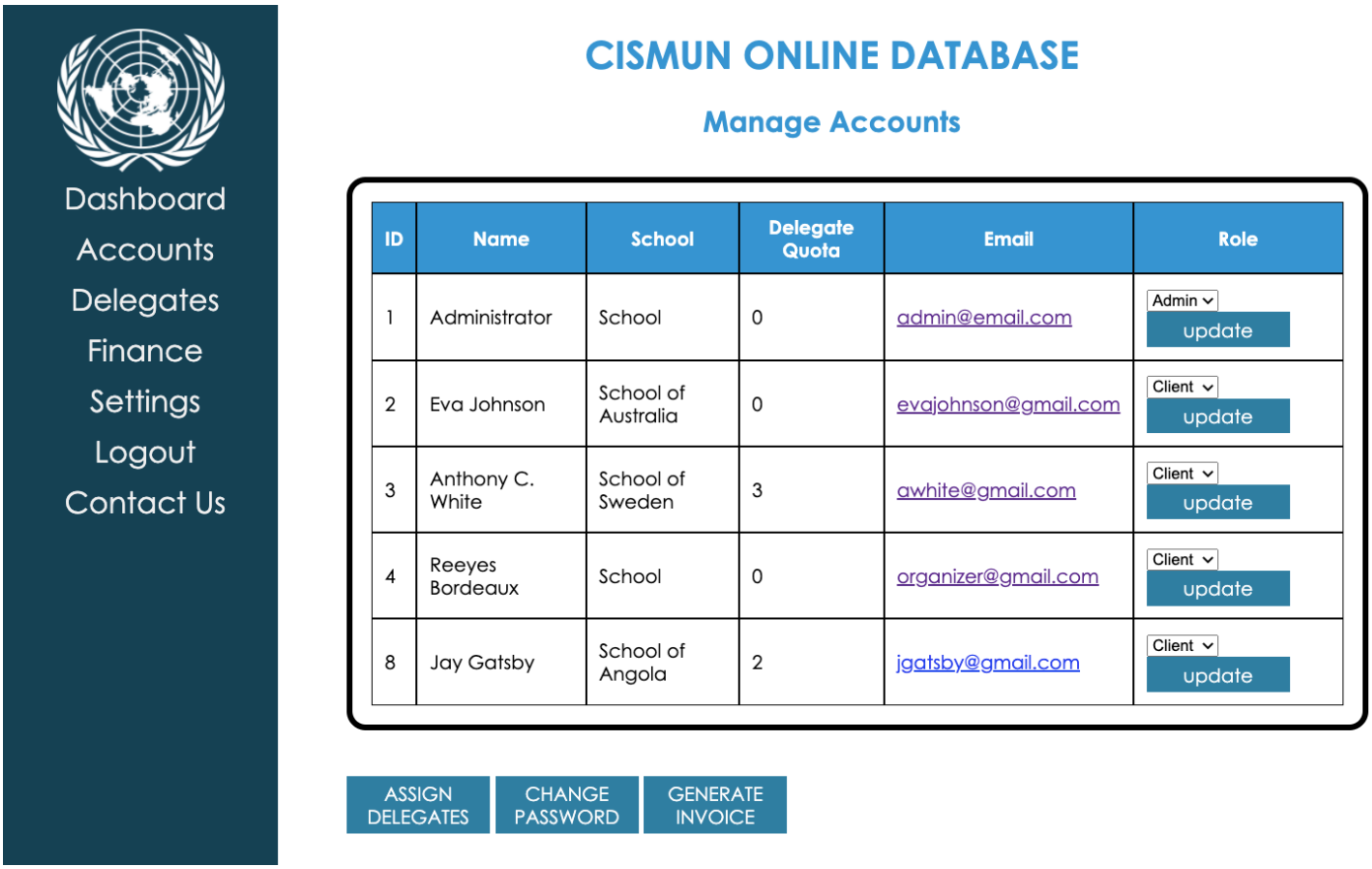


Dynamic creation of HTML elements using ejs (embedded Java Script)

Embedded JavaScript (EJS) is a templating language which enables one to generate HTML markup using JavaScript. ("What is EJS?") As EJS is processed before run-time, this feature was used to display data sent from the server to the user's webpage.

```
<table class="table">
<thead>
    <th>ID</th>
    <th>Name</th>
    <th>School</th>
    <th>Delegate Quota</th>
    <th>Email</th>
    <th>Role</th>
</thead>
<tbody>
    <% users.forEach(user => { %>
    <tr>
    <td><%= user.id %></td>
    <td><%= user.name %></td>
    <td><%= user.school %></td>
    <td><%= user.quota %></td>
    <td><a href="/user/<%= user.id %>"><%= user.email %></a></td>
    <td>
        <form action="/admin/update-role" method="post">
        <input type="hidden" name="id" value="<%= user.id %>" />
        <select name="role" id="role">
            <option value="ADMIN" <%= user.role === 'ADMIN' ? 'selected' : '' %> >Admin</option>
            <option value="CLIENT" <%= user.role === 'CLIENT' ? 'selected' : '' %>>Client</option>
        </select>
        <input type="submit" class="button" value="update">
        </form>
    </td>
    </tr>
    <% }) %>
</tbody>
</table>
```

EJS For-Each loop used to generate HTML Table rows, iterating over delegates array



Asynchronous Programming

JavaScript is synchronous by nature. A synchronous program requires each task is performed sequentially. The disadvantage is that there is high latency in high-traffic environments. To counter this, I have implemented asynchronous techniques to handle requests simultaneously, reducing any delays a user may encounter. This ensured that there is minimal 'lag' experienced by users logging into the database.

```
function initialize(passport, getUserByEmail) {
  const authenticateUser = async (email, password, done) => {
    const user = getUserByEmail(email)
    if (user == null) {
      return done(null, false, { message: 'An account with this email does not exist' })
    }
    try {
      if (await bcrypt.compare(password, user.password)) {
        return done(null, user)
      } else {
        return done(null, false, { message: 'The password you entered is incorrect.' })
      }
    } catch (e) {
      return done(e)
    }
  }
```

Declaring asynchronous authenticateUser() function

Serialization/Deserialization

Serialization in javascript refers to translating an object to a stream of bytes for transmission. Serialization is used to store the ID of the user object to initialize the session between the user and server. Deserialization converts the byte stream back into the user object. This creates a secure session between the user and server, eliminating the need for re-authentication after a fixed timeout.

```
passport.use(new LocalStrategy({ usernameField: 'email' }, authenticateUser))
passport.serializeUser((user, done) => done(null, user))
passport.deserializeUser((user, done) => {
  let sql = "SELECT * FROM accounts WHERE id =" + user.id
  connection.query(sql, (error, results, fields) => {
    if (error) {
      return console.error(error.message)
    }
    done(error, results)
  })
})
```

Password Hashing and Storage

To securely store passwords on my database, I used password hashing so that any intruder who accesses the database wouldn't have access to the user's login credentials. I enabled salting which adds a random string of characters at the end to prevent 'rainbow table' attacks. I decided to use the external library 'bcrypt' instead of coding it myself to reduce the vulnerability risks. The produced hash is stored as a BLOB on the MYSQL table.

```
app.post('/register', checkNotAuthenticated, async (req, res) => {
  try {
    /* adding the new registration as a record in the database*/
    // asynchronous hashing of the password with 10 rounds of salting to ensure data security
    const hashedPassword = await bcrypt.hash(req.body.password, 10)
    var salutation = req.body.salutation
    var name = req.body.name
    var sex = req.body.sex
    var school = req.body.school
    var nationality = req.body.nationality
    var email = req.body.email
```

```
12  const authenticateUser = async (email, password, done) => {
13    const user = getUserByEmail(email)
14    if (user == null) {
15      return done(null, false, { message: 'An account with this email does not exist' })
16    }
17    try {
18      if (await bcrypt.compare(password, user.password)) {
19        return done(null, user)
20      } else {
21        return done(null, false, { message: 'The password you entered is incorrect.' })
22      }
23    } catch (e) {
24      return done(e)
25    }
```

During Authentication, password hash is retrieved and compared with the inputted value.

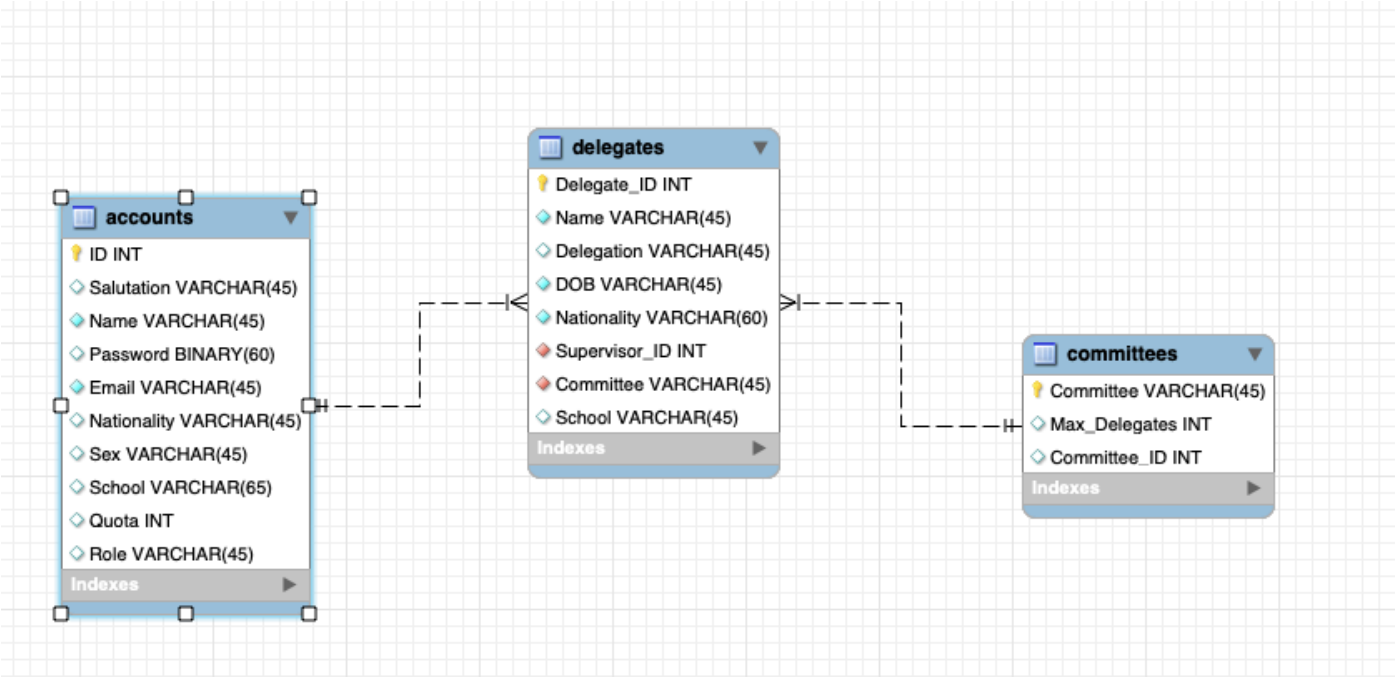
External Libraries

A number of external libraries were used in development to ensure that the program is easy to debug, and easy for an experienced network administrator to develop the code. Libraries such as ‘mysql’ and ‘express’ provide features of creating database connections and the web interface. ‘passport’ and ‘bcrypt’ allows for secure connections to the server.

```
6  const express = require('express') //Build web app, handle post and get requests
7  const flash = require('express-flash')
8  const session = require('express-session')
9  const bcrypt = require('bcrypt') //password hashing
10 const passport = require('passport') //Session Authentication
11 const createHttpError = require('http-errors'); // error-handling library
12 const methodOverride = require('method-override')
13 const bodyParser = require('body-parser');
14 let mysql = require('mysql') //connecting to MYSQL database
15 if (process.env.NODE_ENV !== 'production') {
16   require('dotenv').config() //Creates environment variables
17 }
18 const app = express()
19 var path = require('path');
```

Use of Relational Database Model with 1 to n Foreign Keys

A relational database model was opted to reduce the manual computation required when modifying/deleting records in the databases. The many-to-one link between delegates and committee database provides the client with the function to modify committees.



SQL Statements

```
sql = 'DELETE FROM delegates WHERE (DELEGATE_ID =' + delegate + ');'
var index = delegates.findIndex(x => x.id == delegate);
if (index > -1) {
  delegates.splice(index, 1); // 2nd parameter means remove one item only
}
```

```
if (password != confirmPassword){
  res.render('settings.ejs', {user: req.session.passport.user, messages: {error: "Passwords Do
  not Match"} })
}

let sql = 'UPDATE accounts SET Password = "' + hashedPassword + '" WHERE (ID =' + id + ');'
connection.query(sql, (error)=>{
  if(error){
    return console.error(error.message)
  }
})
var i = users.findIndex(x => x.id == id);
users[i].password = hashedPassword;
```

```
sql = '';
if (del_list){
  if (Array.isArray(del_list)){
    for( let index = 0; index < del_list.length; index++ ) {
      //parallel arrays
      sql += 'INSERT INTO delegates(Delegate_ID, Delegation, Committee, Supervisor_ID) VALUES(NULL,"' + del_list
      [index] +'","' + committee_list[index] + '","' + id+'");'
    }
    connection.query(sql, (error)=>{
      if(error){
        return console.error(error.message)
      }
    })
```

```
const hashedPassword = await bcrypt.hash(req.body.password, 10)
var salutation = req.body.salutation
var name = req.body.name
var sex = req.body.sex
var school = req.body.school
var nationality = req.body.nationality
var email = req.body.email
if (email == "admin@email.com"){
  var role = roles.admin
}
else{
  var role = roles.client
}
let insertStatement = 'INSERT INTO accounts(ID, Salutation, Name, Sex, School, Nationality, Email, Password,
Role) VALUES(NULL,"'+ salutation +'","' + name +'","'+ sex +'","' + school +'","' + nationality +'","' + email
+'","' + hashedPassword +'","' + role +'")'

connection.query(insertStatement, (error)=>{
  if(error){
    return console.error(error.message)
  }
})
```

```
let insertStatement = 'INSERT INTO committees(Committee_ID, Committee,
Max_Delegates) VALUES(NULL,"'+ name +'","' + max_delegates +'")'
connection.query(insertStatement, (error)=>{
  if(error){
    return console.error(error.message)
  }
})
```

```
// Extracting data from the database to the 'users' array
let sql = 'SELECT * FROM accounts'
const users = []
connection.query(sql, (error, results, fields) => {
  if (error) {
    // Displays the error (if any) on the console
    return console.error(error.message)
  }

  for (var i = 0; i < results.length; i++) {
    users.push({
      id: results[i].ID,
      salutation: results[i].Salutation,
      name: results[i].Name,
      sex: results[i].Sex,
      school: results[i].School,
      nationality: results[i].Nationality,
      email: results[i].Email,
      password: results[i].Password.toString(),
      role: results[i].Role,
      quota: results[i].Quota
    })
  }
});
```

User-defined Tree Data Structures for Predictive Search (Trie)

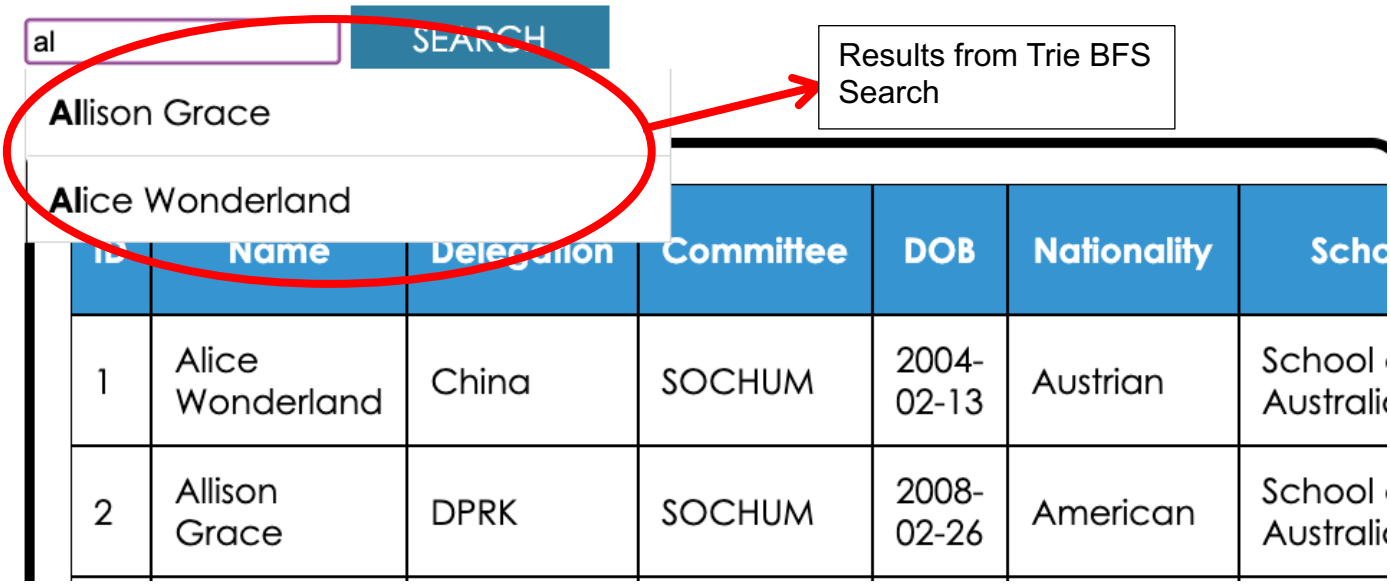
A Trie is a tree data structure which is created on the basis of String prefixes. To implement these trees, I took an object-oriented approach by creating the Node and Trie class. The organized structure allowed for more efficient and quicker problem-solving and development. Additionally, the classes allowed for an easier creation of multiple Node or Trie objects, which helped in debugging the code.

```
class Node {
    constructor(){
        this.children = {}
        this.last = false //indicates end of word
    }
}

class TrieAutocomplete{
    constructor(wordList){
        const root = new Node()
        for (const word of wordList){
            this.insertNode(root, word)
        }
        this.root = root
    }

```

```
insertNode(node, word){
    for(const char of word){
        // creates node if it doesn't exist already
        if (node.children[char] === undefined) {
            node.children[char] = new Node()
        }
        node = node.children[char]
    }
    node.last = true;
}

```

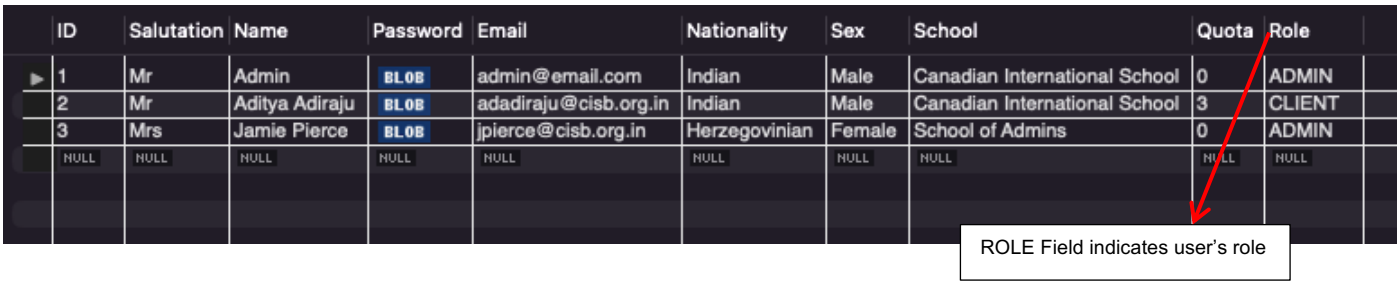
```
97      beginsWith(word){
98          let node = this.root
99          let foundFlag = true
100         let result = ''
101
102         for (const char of word){
103
104             if (node.children[char]) {
105                 node = node.children[char]
106                 result += char;
107             }
108             //Case-insenstive search
109             else if (node.children[char.toUpperCase()]){
110                 node = node.children[char.toUpperCase()]
111                 result += char.toUpperCase()
112             }
113             else{
114                 foundFlag = false;
115                 break;
116                 //exit loop if not found
117             }
118         }
119         if (foundFlag) {
120           return{
121             node: node,
122             result
123           }
124         } else{
125           return{
126             node: null,
127             result
128           }
129         }
130     }
```

```
    breadthFirstSearch(searchQuery){

        const {node, result} = this.beginsWith(searchQuery)

        if (!node){
            return []
        }

        const list = []
        // queue stores possible Trie results or traversals
        const queue = [{node, result}]
        const possibleWords = []
        //if current node is a word, add to suggestion list
        if (node.last){
            list.push(result)
        }

        while (queue.length){
            const {node, result} = queue.shift()
            // array.shift() removes and returns the element at index 0
            // queue.shift is a dequeue operation
            //
            for(const child in node.children){
                if(node.children[child].last){
                    possibleWords.push(result + child)
                }
                //queue.push() acts as enqueue operation
                queue.push({
                    node: node.children[child],
                    result: result + child
                })

            }
        }
        return list.concat(possibleWords)
    }
}
```



Role-Based Access Control System (RBAC)

To implement this system, functions such as 'checkAdmin()' were created for the server which checked the user's role and returned true if they were an admin. When the user first logs in or accesses any admin-only page, the function is called before the GET request for each of such pages. If an unauthorised user tries to access an admin-only page, the function redirects them to the dashboard.



ID	Salutation	Name	Password	Email	Nationality	Sex	School	Quota	Role
1	Mr	Admin	BLOB	admin@email.com	Indian	Male	Canadian International School	0	ADMIN
2	Mr	Aditya Adiraju	BLOB	adadiraju@cisb.org.in	Indian	Male	Canadian International School	3	CLIENT
3	Mrs	Jamie Pierce	BLOB	jpierce@cisb.org.in	Herzegovinian	Female	School of Admins	0	ADMIN
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Dashboard
Accounts
Delegates
Finance
Settings
Logout
Contact Us

CISMUN ONLINE DATABASE

Admin Dashboard

PERSONAL INFORMATION

Supervisor ID: 1
Name: Administrator
Email Address: admin@email.com
Total Delegates: 5

Committee Name	Max Delegation
IMO	13
SOCHUM	23
SPECPOL	12
UNHRC	12
UNSC	15

Add Committee | Modify Committee

Admin Dashboard and menu

Different Dashboards shown to users and admin using RBAC

Dashboard
Resources
Invoice
Settings
Logout
Contact Us

PERSONAL INFORMATION

Supervisor ID: 8
Title: Mr
Name: Jay Gatsby
Sex: Male
School: School of Angola
Nationality: Angolan
Email Address: jgatsby@gmail.com
Quota: 2

☐ **Delegate 1**

Delegate ID: 8
Name: Ria Balan
Delegation: USA
Committee: SPECPOL
Date of Birth: 2008-02-26
Nationality: Algerian
School: School of Angola

User Dashboard and Menu

```
459 function checkAuthenticated(req, res, next) {
460   if (req.isAuthenticated()) {
461     return next()
462   }
463   res.redirect('/login')
464 }
465
466 function checkNotAuthenticated(req, res, next) {
467   if (req.isAuthenticated()) {
468     return res.redirect('/')
469   }
470   next()
471 }
472
473 function checkAdmin(req, res, next) {
474
475   if (req.session.passport.user.role == roles.admin) {
476     next();
477   } else {
478     res.redirect('/');
479   }
480 }
```

Verifies user's role before creating GET request for sites to prevent unauthorized access

```
118 app.get('/', checkAuthenticated, (req, res, next) => {
119   if (req.session.passport.user.role == roles.admin){
120     res.render('admin-index.ejs', {
121       delegates: delegates.filter((x) => { return x.supervisor_id == req.
        session.passport.user.id;}),
122       user: req.session.passport.user, Committees: committees}
123       )
124   } else{
125       res.render('index.ejs', {
126       delegates: delegates.filter((x) => { return x.supervisor_id == req.
        session.passport.user.id;}),
127       user: req.session.passport.user})
128   }
129 })
```

Sending Mails through SMTP and Node-Mailer

In order to send emails from the server, The node-mailer library was used to send formatted emails to the client and users via the SMTP (Simple Mail Transfer Protocol) server.

```
262   var name = req.body.name
263   var email = req.body.email
264   var school = req.body.school
265   var message = req.body.msg
266
267   let mailTransporter = nodemailer.createTransport(emailAuth);
268
269   let mailDetails = {
270     from: emailAuth.auth.user, //noreply@server.com
271     to: '[redacted]',
272     subject: 'You\'ve recieved a response from your Contact Form',
273     text: "From: "+ name + "\nSchool: "+ school + "\nEmail: "+ email +"\n\n" + message
274   };
275
276   mailTransporter.sendMail(mailDetails, function(err, data) {
277     if(err) {
278         console.error(err.message)
279     } else {
280         console.log('Email sent successfully');
281     }
282   });
283   res.redirect('/login')
284 })
285
```

Node mailer used to create and send emails

Word Count: 879

Citations

Divine, Patrick. “Serialization and Deserialization.” *Medium*, Better Programming, 17 Oct. 2019, https://betterprogramming.pub/serialization-and-deserialization-ba12fc3fbe23.

“How to - Fixed Sidebar.” *How To Create a Fixed Sidebar*, Geeks For Geeks, https://www.w3schools.com/howto/howto_css_fixed_sidebar.asp.

“How to Create a Popup Form Using JavaScript.” *W3docs*, W3 Docs, https://www.w3docs.com/snippets/javascript/how-to-create-a-popup-form-using-javascript.html.

L_KL_K 2. “Compare Passwords Bcryptjs.” *Stack Overflow*, 17 Oct. 2016, https://stackoverflow.com/questions/40076638/compare-passwords-bcryptjs.

Mai Brooklyn, Mike. “Autocomplete / Suggestion / Typeahead Search Using Trie Data Structure and Algorithm.” *Mike Mai*, 8 June 2020, https://maideveloper.com/blog/autocomplete-suggestion-typeahead-search-using-trie-data-structure-and-algorithm.

“MySQL 8.0 Reference Manual.” *MySQL*, Oracle Corporation, https://dev.mysql.com/doc/refman/8.0/en/.

“Node - Web Apis: MDN.” *Web APIs | MDN*, Mozilla Corporation, https://developer.mozilla.org/en-US/docs/Web/API/Node.

Reinman, Andris. “Using Gmail.” *Nodemailer*, 13 Jan. 2016, https://community.nodemailer.com/using-gmail/.

Sev, Chris. “How to Use EJS to Template Your Node Application.” *DigitalOcean*, DigitalOcean, 5 May 2021, https://www.digitalocean.com/community/tutorials/how-to-use-ejs-to-template-your-node-application.

Trulymittal. “Role Based Access Control.” *GitHub*, 21 Dec. 2020, https://github.com/trulymittal/role-based-access-control.

WebDevSimplified. “WebDevSimplified/Nodejs-Passport-Login.” *GitHub*, 10 July 2019, https://github.com/WebDevSimplified/Nodejs-Passport-Login.

“What Is EJS.” *EJS*, https://ejs.co/.