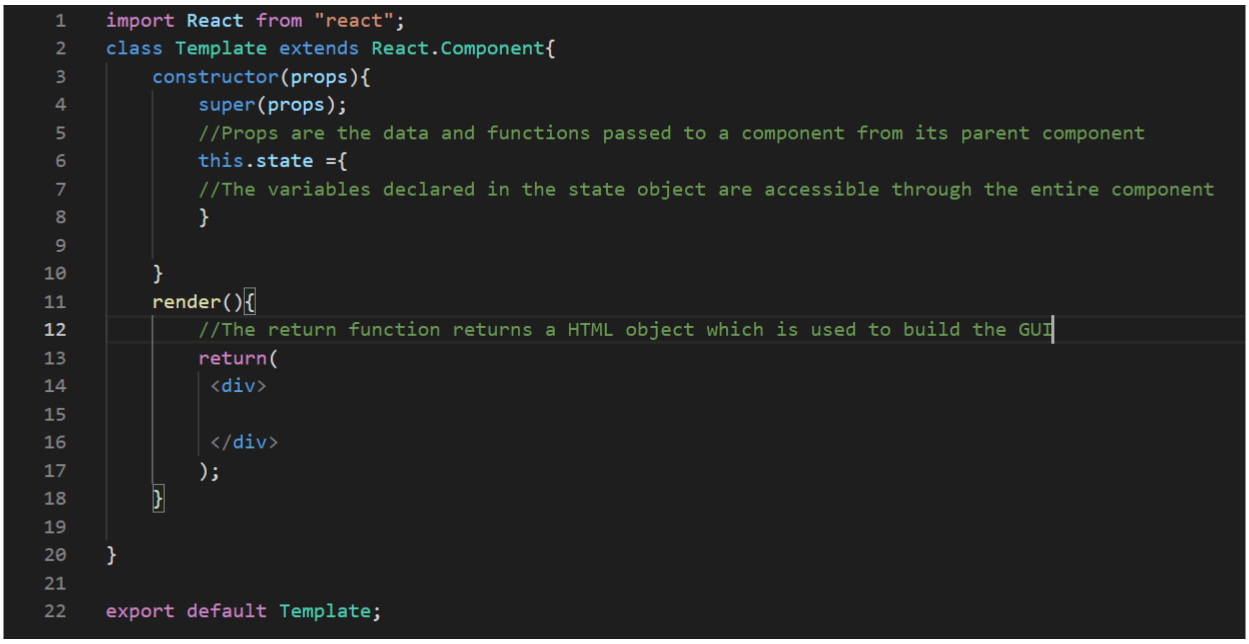
Criterion C: Development

__Techniques used:__

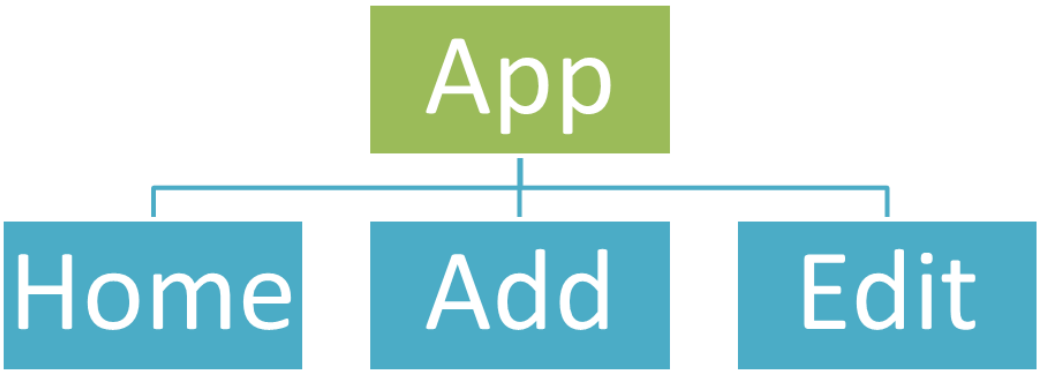
React JS components: Similar to objects in OOP, React JS has components that split the application into different parts to form the final product. The basic template of a React JS component is as follows:



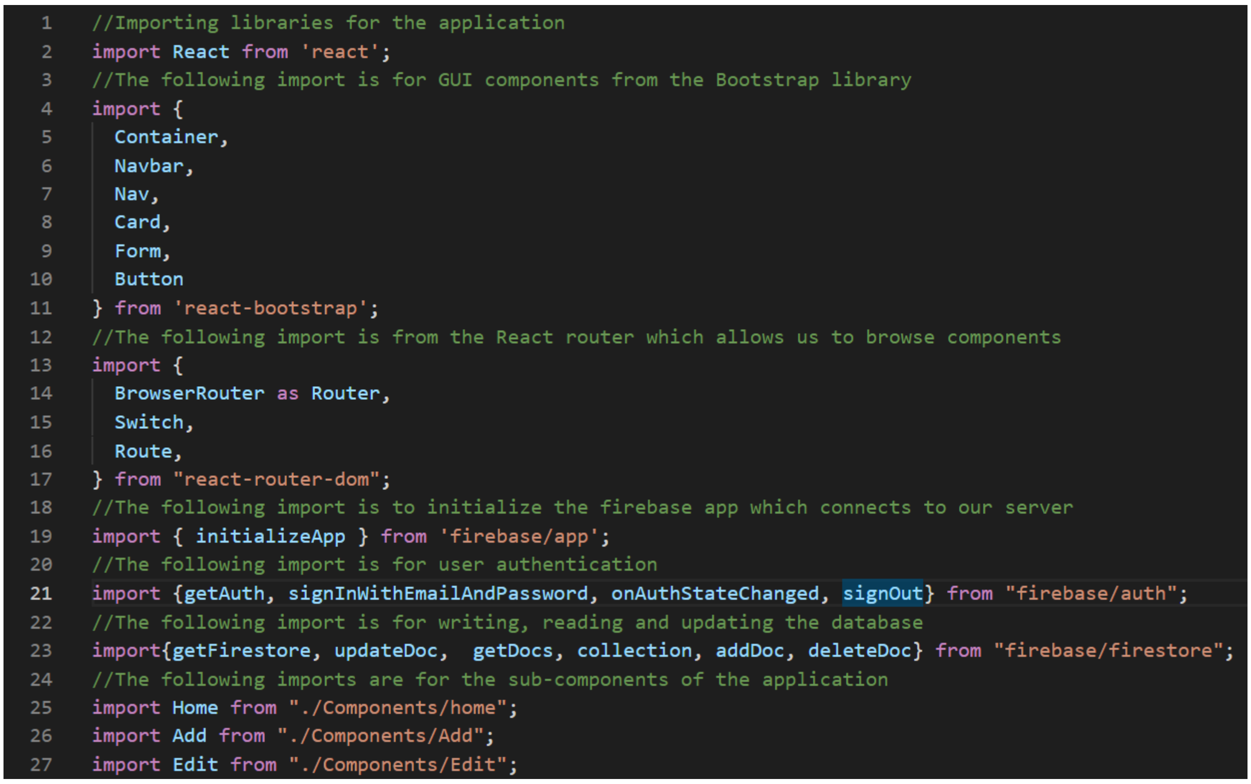
```
import React from "react";
class Template extends React.Component{
    constructor(props){
        super(props);
        //Props are the data and functions passed to a component from its parent component
        this.state ={
        //The variables declared in the state object are accessible through the entire component
        }

    }
    render(){
        //The return function returns a HTML object which is used to build the GUI
        return(
         <div>

         </div>
        );
    }

}

export default Template;
```

The main component of the project is the *App* component, which uses the react-router library to present the other components.

Component Hierarchy of the application:

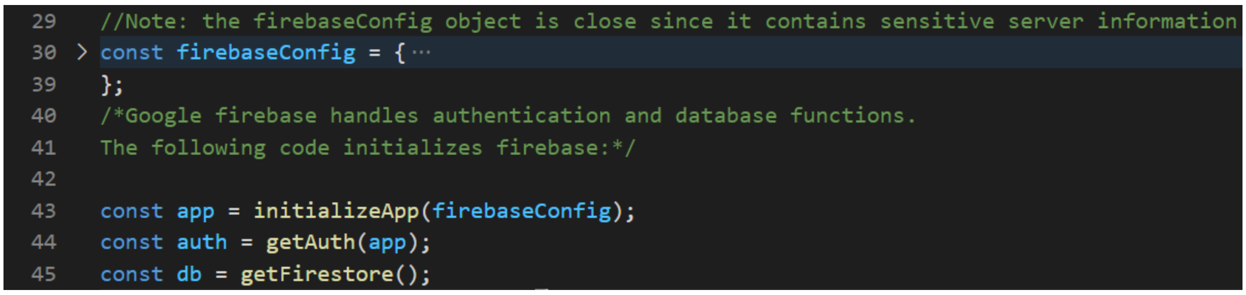


Libraries used:



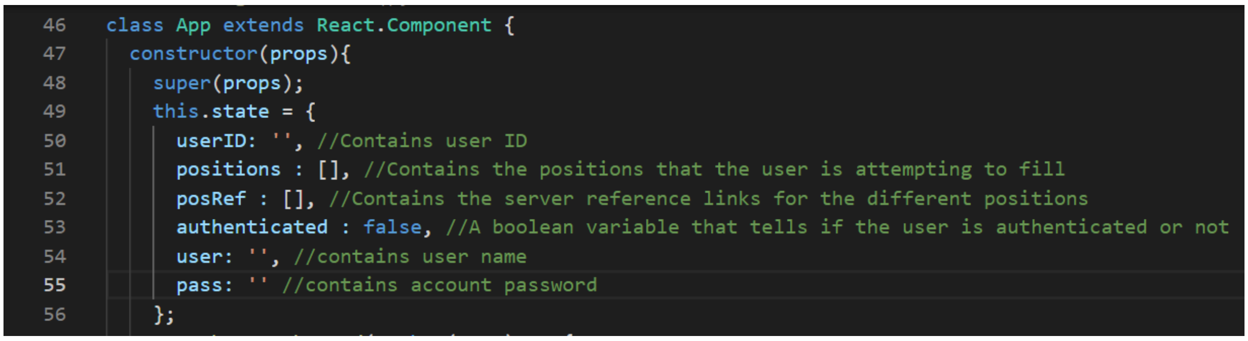
```
//Importing libraries for the application
import React from 'react';
//The following import is for GUI components from the Bootstrap library
import {
  Container,
  Navbar,
  Nav,
  Card,
  Form,
  Button
} from 'react-bootstrap';
//The following import is from the React router which allows us to browse components
import {
  BrowserRouter as Router,
  Switch,
  Route,
} from "react-router-dom";
//The following import is to initialize the firebase app which connects to our server
import { initializeApp } from 'firebase/app';
//The following import is for user authentication
import {getAuth, signInWithEmailAndPassword, onAuthStateChanged, signOut} from "firebase/auth";
//The following import is for writing, reading and updating the database
import{getFirestore, updateDoc,  getDocs, collection, addDoc, deleteDoc} from "firebase/firestore";
//The following imports are for the sub-components of the application
import Home from "./Components/home";
import Add from "./Components/Add";
import Edit from "./Components/Edit";
```

Initializing Firebase:



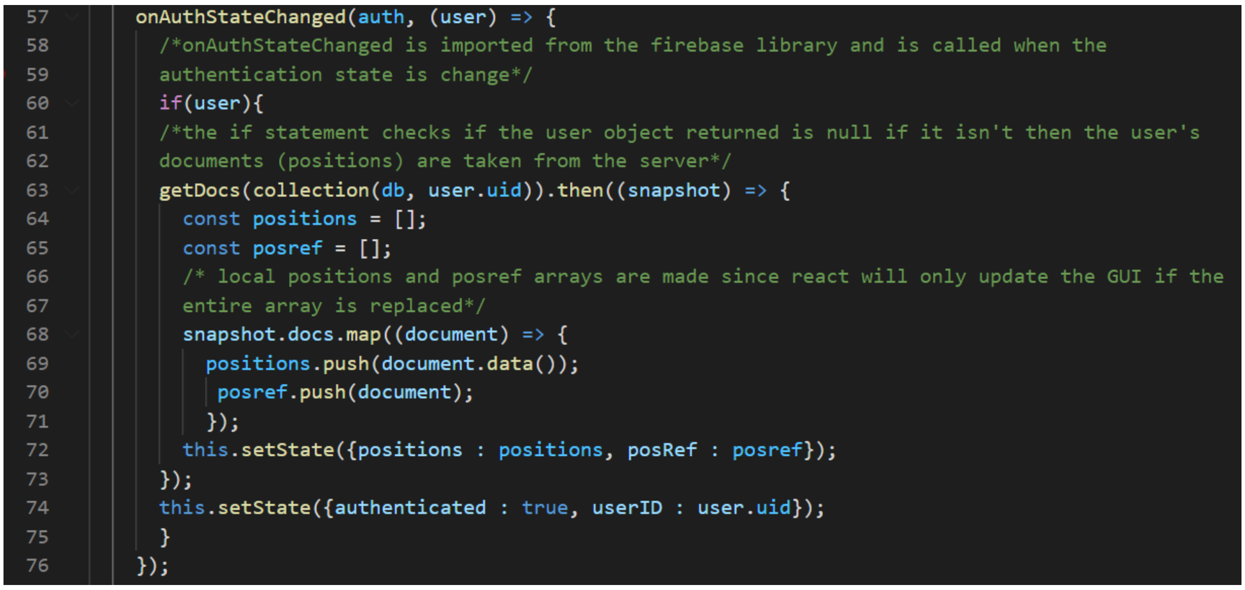
```
//Note: the firebaseConfig object is close since it contains sensitive server information
> const firebaseConfig = {⋯
};
/*Google firebase handles authentication and database functions.
The following code initializes firebase:*/

const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const db = getFirestore();
```

Component State:



```
class App extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      userID: '', //Contains user ID
      positions : [], //Contains the positions that the user is attempting to fill
      posRef : [], //Contains the server reference links for the different positions
      authenticated : false, //A boolean variable that tells if the user is authenticated or not
      user: '', //contains user name
      pass: '' //contains account password
    };
```

Checking if user is authenticated:

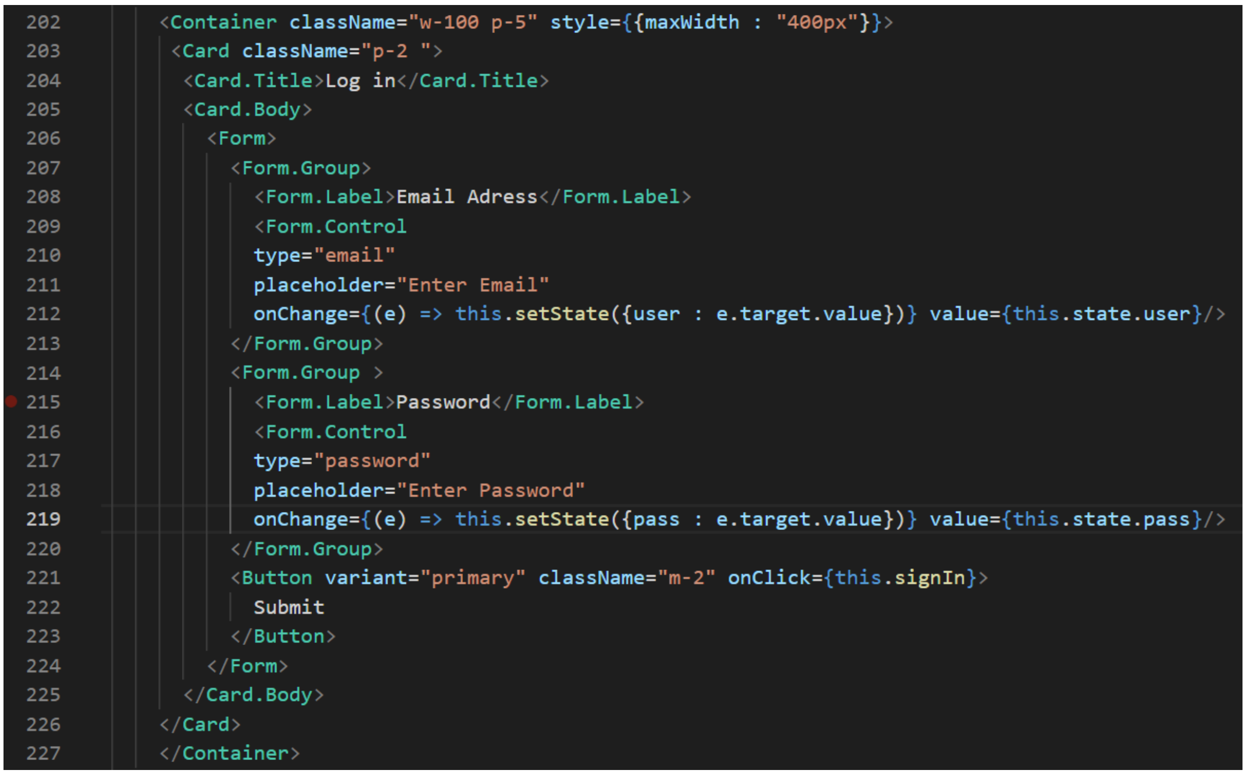


```
onAuthStateChanged(auth, (user) => {
  /*onAuthStateChanged is imported from the firebase library and is called when the
  authentication state is change*/
  if(user){
  /*the if statement checks if the user object returned is null if it isn't then the user's
  documents (positions) are taken from the server*/
  getDocs(collection(db, user.uid)).then((snapshot) => {
    const positions = [];
    const posref = [];
    /* local positions and posref arrays are made since react will only update the GUI if the
    entire array is replaced*/
    snapshot.docs.map((document) => {
      positions.push(document.data());
       posref.push(document);
      });
    this.setState({positions : positions, posRef : posref});
  });
  this.setState({authenticated : true, userID : user.uid});
  }
});
```

The Sign In function

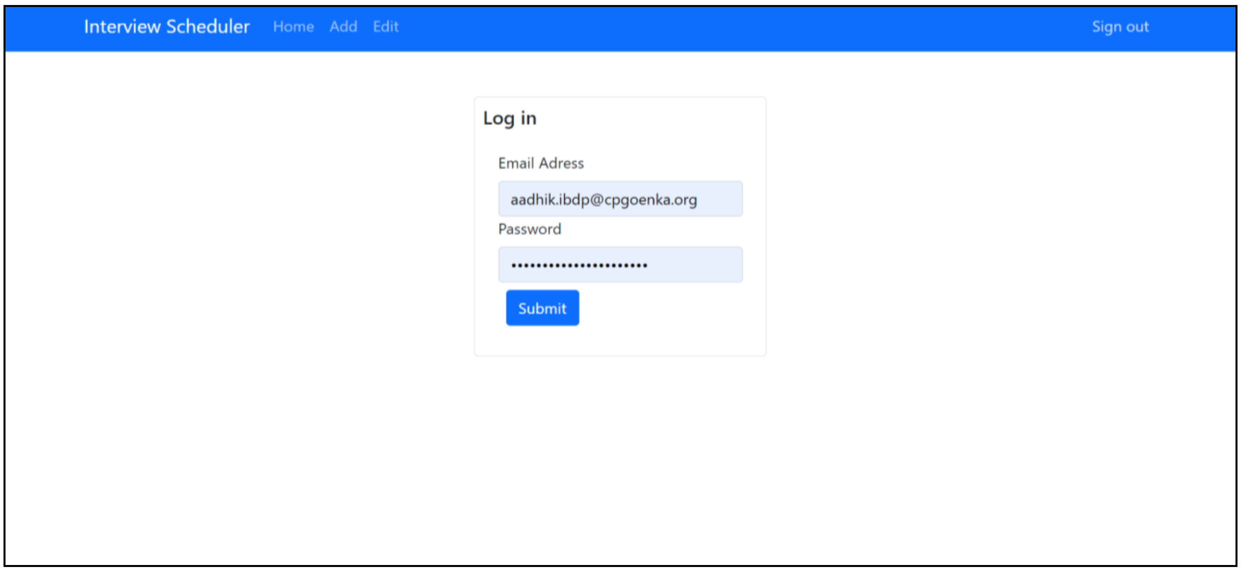
GUI:

This GUI is built primarily using the imported bootstrap libraries. The details mentioned within the $className$ and $style$ attributes set the maximum width, padding and margin of the different components. The $< Form.Label >$ component labels the $< Form.control >$ component which is the input field where data is entered. The $value$ attribute identifies the state variable which stores the data inputted and the $onChange$ attribute defines a function to update the state variable as the data inputted is changed.



```
<Container className="w-100 p-5" style={{maxWidth : "400px"}}>
 <Card className="p-2 ">
  <Card.Title>Log in</Card.Title>
  <Card.Body>
    <Form>
      <Form.Group>
        <Form.Label>Email Adress</Form.Label>
        <Form.Control
        type="email"
        placeholder="Enter Email"
        onChange={(e) => this.setState({user : e.target.value})} value={this.state.user}/>
      </Form.Group>
      <Form.Group >
        <Form.Label>Password</Form.Label>
        <Form.Control
        type="password"
        placeholder="Enter Password"
        onChange={(e) => this.setState({pass : e.target.value})} value={this.state.pass}/>
      </Form.Group>
      <Button variant="primary" className="m-2" onClick={this.signIn}>
        Submit
      </Button>
    </Form>
  </Card.Body>
</Card>
</Container>
```

When the $< Button >$ component is clicked, the $onClick$ attribute is used to call the $signIn$ function which handles authentication. The finished GUI component looks like:



Log in

Email Adress

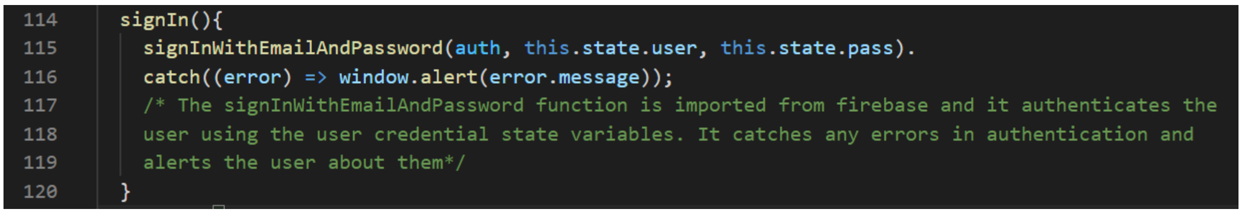
aadhik.ibdp@cpgoenka.org

Password

•••••••••••••••••••••••

Screenshot 1: Log in screen

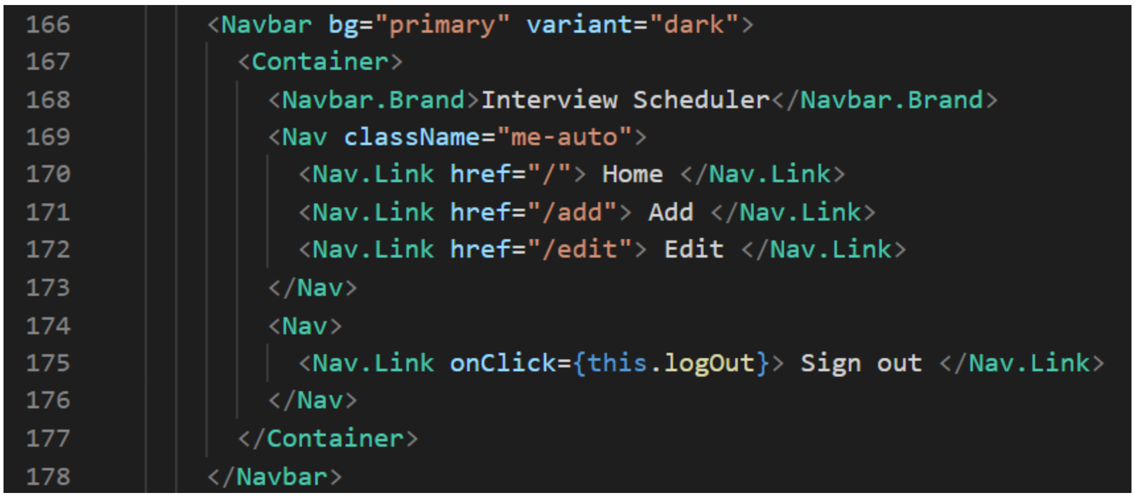
Logic for $signIn$ function:



```
signIn(){
  signInWithEmailAndPassword(auth, this.state.user, this.state.pass).
  catch((error) => window.alert(error.message));
  /* The signInWithEmailAndPassword function is imported from firebase and it authenticates the
  user using the user credential state variables. It catches any errors in authentication and
  alerts the user about them*/
}
```

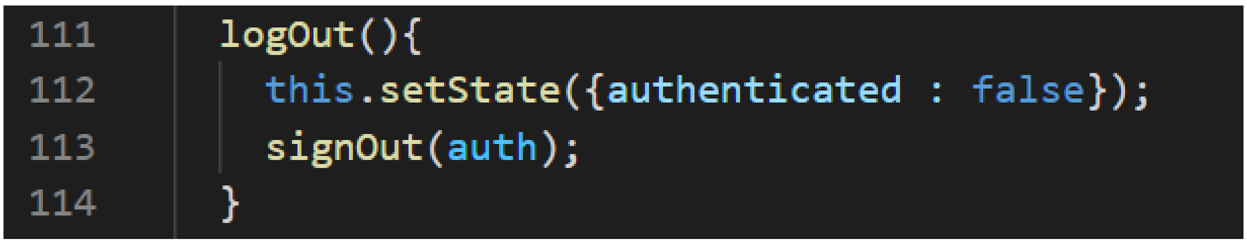
Component Navigation

The following is the code for the navigation bar. There are two $< Nav >$ within the navigation bar these separate it into the component navigation buttons and the Sign out button. Each of the $< Nav.Link >$ component references a link for the corresponding component through the $href$ attribute.



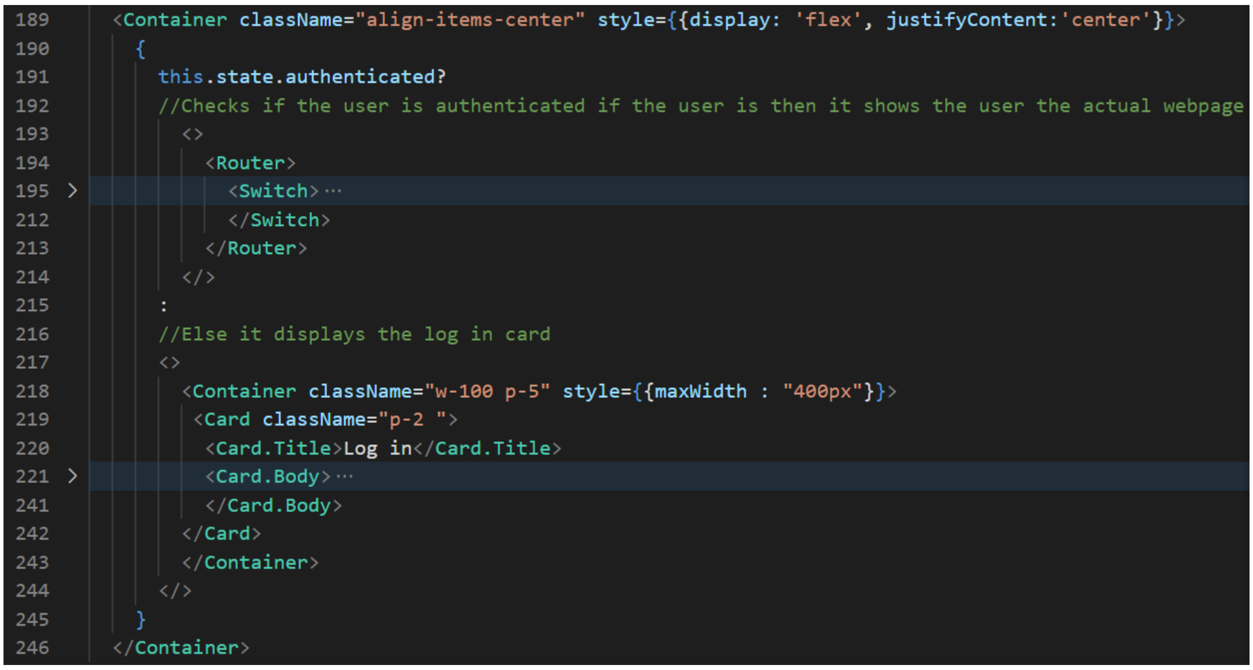
```
<Navbar bg="primary" variant="dark">
  <Container>
    <Navbar.Brand>Interview Scheduler</Navbar.Brand>
    <Nav className="me-auto">
      <Nav.Link href="/"> Home </Nav.Link>
      <Nav.Link href="/add"> Add </Nav.Link>
      <Nav.Link href="/edit"> Edit </Nav.Link>
    </Nav>
    <Nav>
      <Nav.Link onClick={this.logOut}> Sign out </Nav.Link>
    </Nav>
  </Container>
</Navbar>
```

When the Sign out $< Nav.Link >$ component is clicked it calls the $logOut$ function. The code for which is:



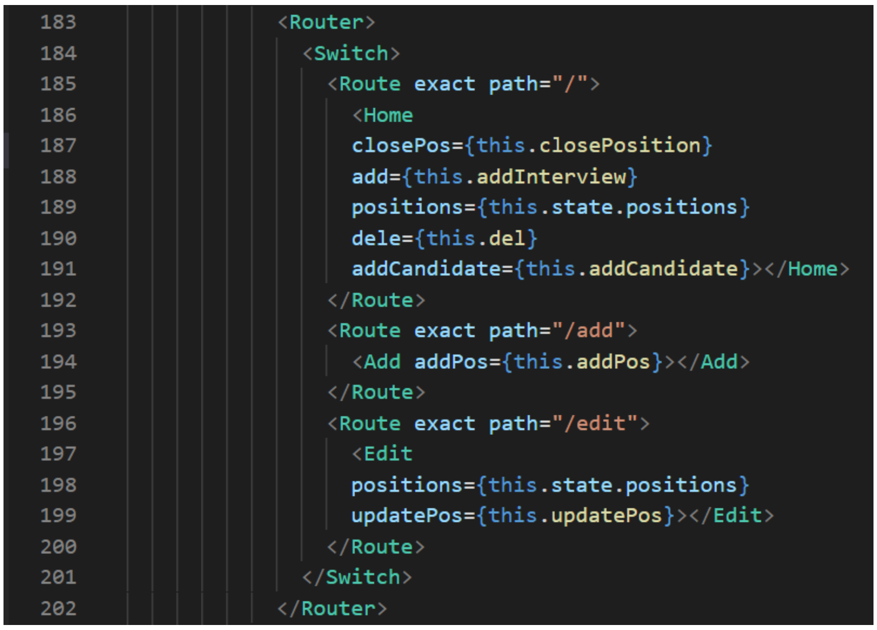
```
logOut(){
  this.setState({authenticated : false});
  signOut(auth);
}
```

The function sets the $authenticated$ variable to false and calls the $signOut$ function we imported from the firebase library.



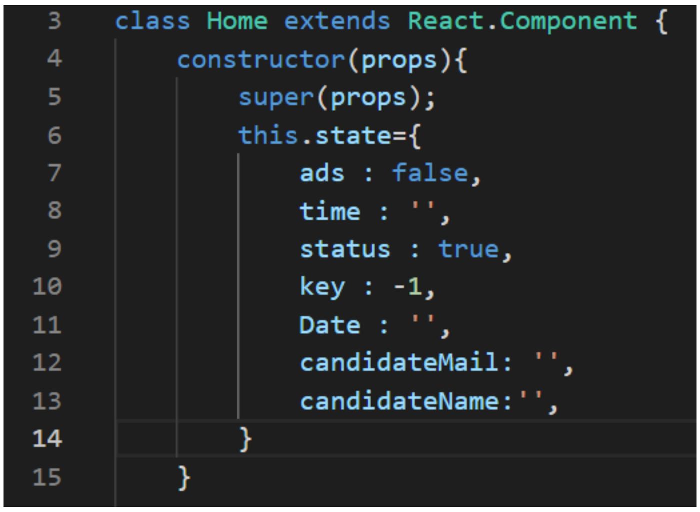
```
<Container className="align-items-center" style={{display: 'flex', justifyContent:'center'}}>
  {
    this.state.authenticated?
    //Checks if the user is authenticated if the user is then it shows the user the actual webpage
      <>
        <Router>
          <Switch>···
          </Switch>
        </Router>
      </>
    :
    //Else it displays the log in card
    <>
      <Container className="w-100 p-5" style={{maxWidth : "400px"}}>
       <Card className="p-2 ">
        <Card.Title>Log in</Card.Title>
        <Card.Body>···
        </Card.Body>
      </Card>
      </Container>
    </>
  }
</Container>
```

The next code snippet contains the *router* component which we imported from the react router library. The different $< Route >$ components present the imported components when the appropriate link is used in the address bar of the web browser. Each of the components we imported *Home, Add and Edit* all have attributes which consists of function and data that is being passed down to each from our main *App* component these attributes are called props.



```
<Router>
  <Switch>
    <Route exact path="/">
      <Home
      closePos={this.closePosition}
      add={this.addInterview}
      positions={this.state.positions}
      dele={this.del}
      addCandidate={this.addCandidate}></Home>
    </Route>
    <Route exact path="/add">
      <Add addPos={this.addPos}></Add>
    </Route>
    <Route exact path="/edit">
      <Edit
      positions={this.state.positions}
      updatePos={this.updatePos}></Edit>
    </Route>
  </Switch>
</Router>
```

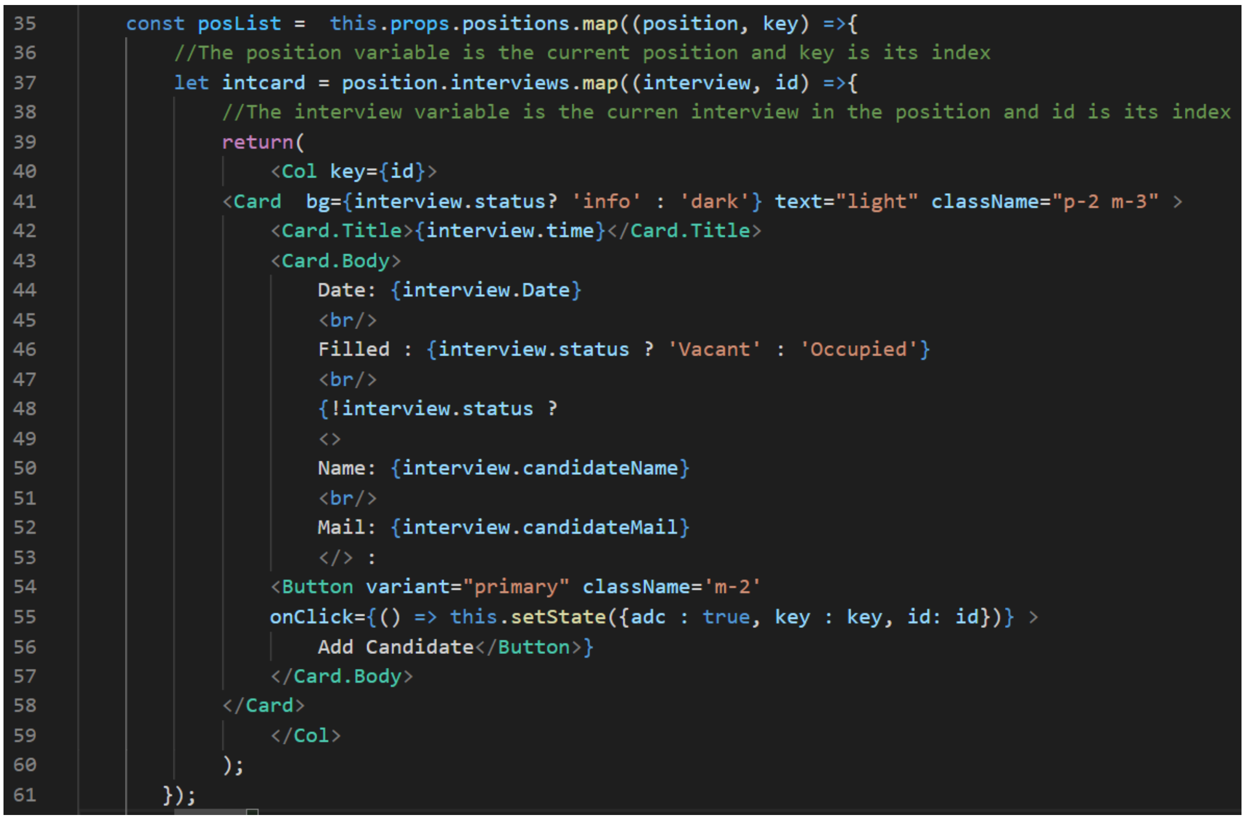
Home **component:**



```
class Home extends React.Component {
    constructor(props){
        super(props);
        this.state={
            ads : false,
            time : '',
            status : true,
            key : -1,
            Date : '',
            candidateMail: '',
            candidateName:'',
        }
    }
```

The state for the *home* component consists of the variables used to add new interviews to a position.

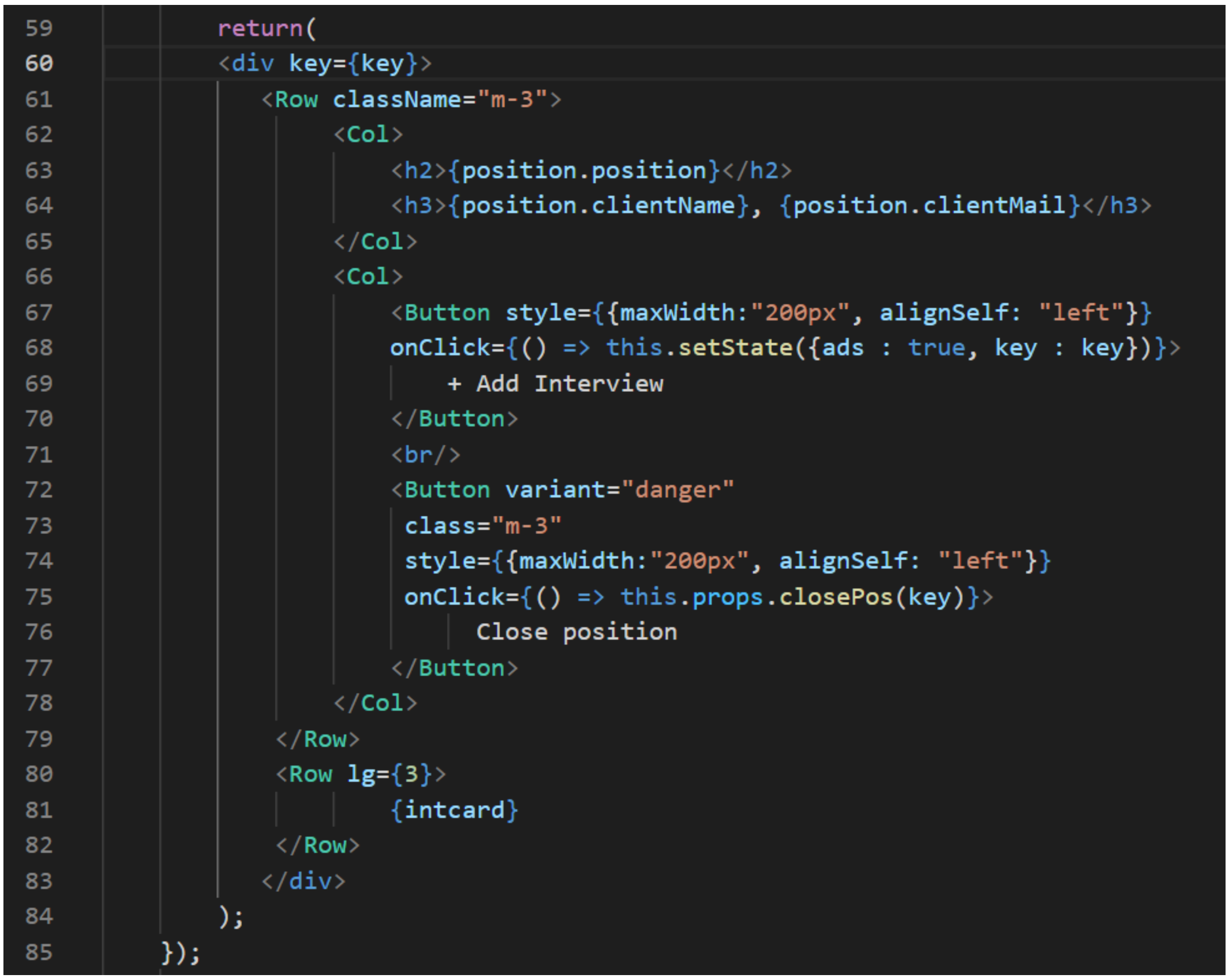
Array iteration:



```
    const posList =  this.props.positions.map((position, key) =>{
        //The position variable is the current position and key is its index
        let intcard = position.interviews.map((interview, id) =>{
            //The interview variable is the curren interview in the position and id is its index
            return(
                <Col key={id}>
            <Card  bg={interview.status? 'info' : 'dark'} text="light" className="p-2 m-3" >
                <Card.Title>{interview.time}</Card.Title>
                <Card.Body>
                    Date: {interview.Date}
                    <br/>
                    Filled : {interview.status ? 'Vacant' : 'Occupied'}
                    <br/>
                    {!interview.status ?
                    <>
                    Name: {interview.candidateName}
                    <br/>
                    Mail: {interview.candidateMail}
                    </> :
                <Button variant="primary" className='m-2'
                onClick={() => this.setState({adc : true, key : key, id: id})} >
                    Add Candidate</Button>}
                </Card.Body>
            </Card>
                </Col>
            );
        });
```

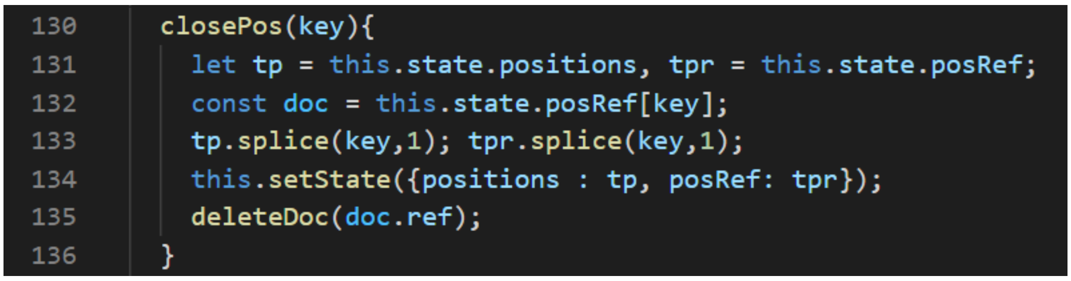
The *App* component passes the positions that we obtained from the server to the *Home* component. The *Home* component iterates through each interview of each position to form the interview cards for the home page. The *interview. status* is a Boolean variable that lets us know if the interview is occupied or not. Accordingly, the bg attribute sets the background colour of the interview card to $info$ (blue) or $dark$ (black) and it displays if the interview is occupied or vacant in the $filled$ field of the interview card. If the interview is occupied it shows the candidate's name and mail else it displays the add candidate button.

Further, we iterate through each position of the positions array and we display the Position, Client Name and Client mail. We also have the buttons for the add interview and the close position. Inside, each of the positions we use the $intcard$ component which we defined in line 37 of the $Home.js$ file in the appendix (also available on page 8)



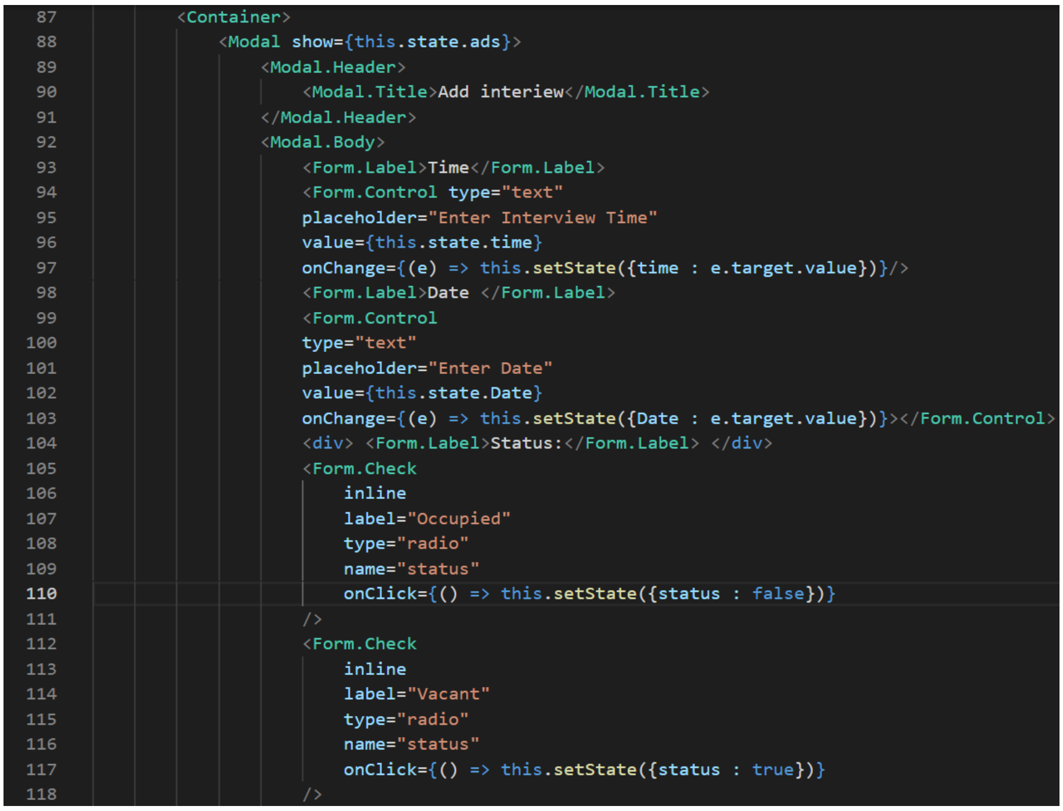
```
        return(
        <div key={key}>
            <Row className="m-3">
                <Col>
                    <h2>{position.position}</h2>
                    <h3>{position.clientName}, {position.clientMail}</h3>
                </Col>
                <Col>
                    <Button style={{maxWidth:"200px", alignSelf: "left"}}
                    onClick={() => this.setState({ads : true, key : key})}>
                        + Add Interview
                    </Button>
                    <br/>
                    <Button variant="danger"
                     class="m-3"
                     style={{maxWidth:"200px", alignSelf: "left"}}
                     onClick={() => this.props.closePos(key)}>
                         Close position
                    </Button>
                </Col>
             </Row>
             <Row lg={3}>
                    {intcard}
             </Row>
            </div>
        );
    });
```

When the close position button is clicked, it calls the $closePos$ prop function is called which removes the position from the $positions$ and $posRef$ array and then the server.



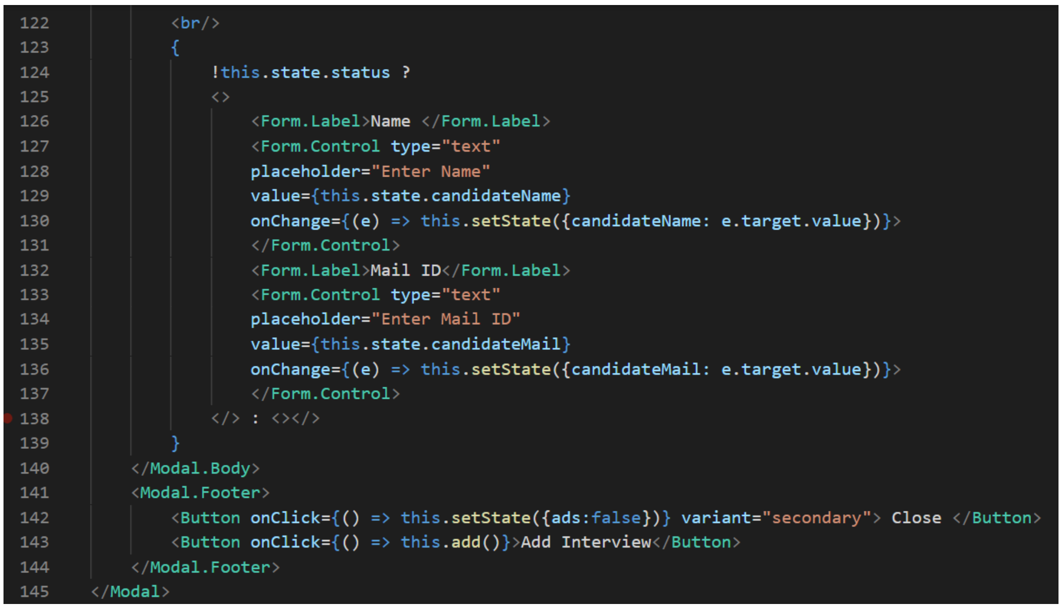
```
closePos(key){
  let tp = this.state.positions, tpr = this.state.posRef;
  const doc = this.state.posRef[key];
  tp.splice(key,1); tpr.splice(key,1);
  this.setState({positions : tp, posRef: tpr});
  deleteDoc(doc.ref);
}
```

When the add interview button is clicked, the ads state variable is set to true which opens up the add interview modal window and the key variable in state is set to the index of the position in the $positions$ array which we obtain from the $positions.map$ function on line 34 of home.js which is included in the appendix. The code for which is attached below:



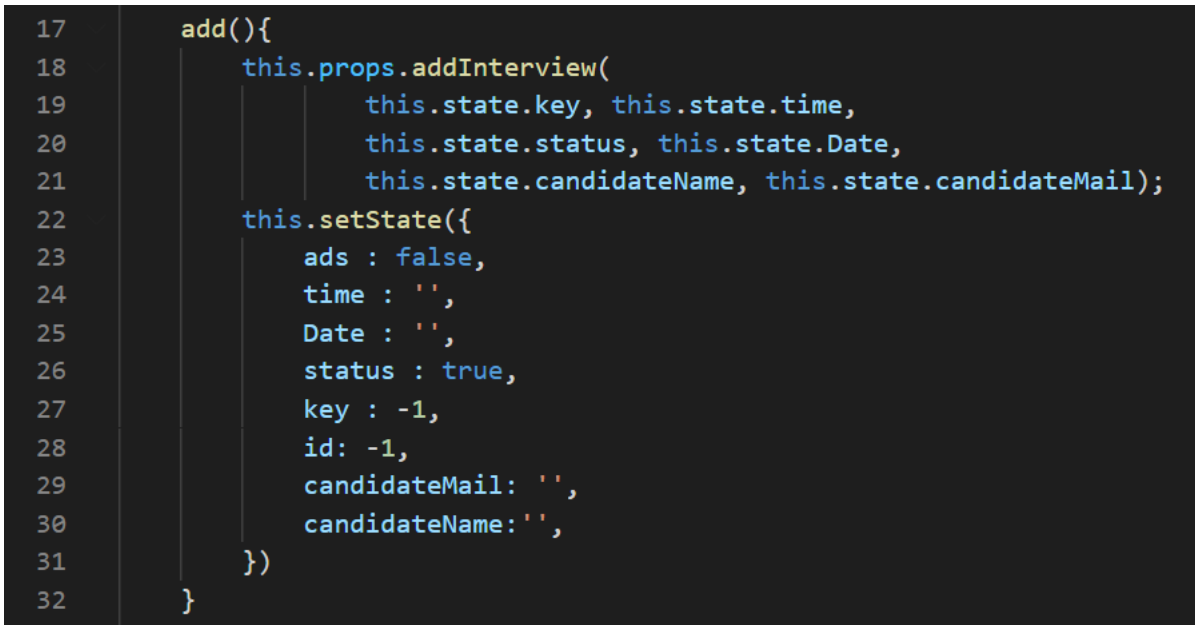
```
<Container>
  <Modal show={this.state.ads}>
    <Modal.Header>
      <Modal.Title>Add interiew</Modal.Title>
    </Modal.Header>
    <Modal.Body>
      <Form.Label>Time</Form.Label>
      <Form.Control type="text"
      placeholder="Enter Interview Time"
      value={this.state.time}
      onChange={(e) => this.setState({time : e.target.value})}/>
      <Form.Label>Date </Form.Label>
      <Form.Control
      type="text"
      placeholder="Enter Date"
      value={this.state.Date}
      onChange={(e) => this.setState({Date : e.target.value})}></Form.Control>
      <div> <Form.Label>Status:</Form.Label> </div>
      <Form.Check
        inline
        label="Occupied"
        type="radio"
        name="status"
        onClick={() => this.setState({status : false})}
      />
      <Form.Check
        inline
        label="Vacant"
        type="radio"
        name="status"
        onClick={() => this.setState({status : true})}
      />
```

The code snippet above shows fields for interview time, date and status. If the interview status is set to occupied it displays fields for the candidate's name and mail which is presented in the code below:



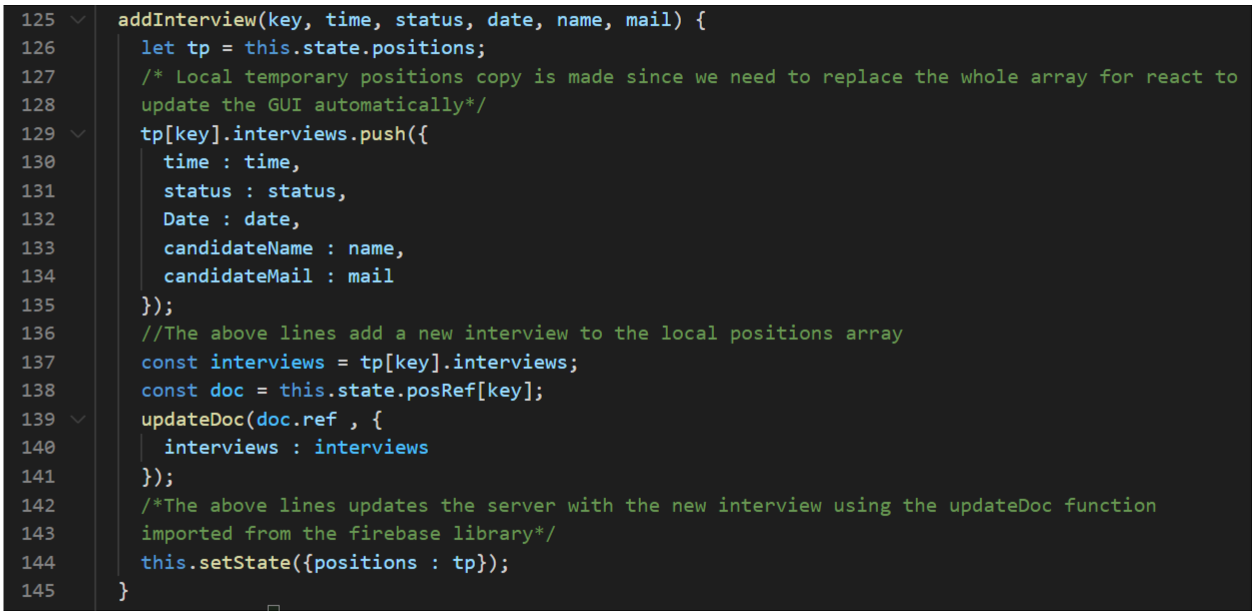
```
        <br/>
        {
            !this.state.status ?
            <>
                <Form.Label>Name </Form.Label>
                <Form.Control type="text"
                placeholder="Enter Name"
                value={this.state.candidateName}
                onChange={(e) => this.setState({candidateName: e.target.value})}>
                </Form.Control>
                <Form.Label>Mail ID</Form.Label>
                <Form.Control type="text"
                placeholder="Enter Mail ID"
                value={this.state.candidateMail}
                onChange={(e) => this.setState({candidateMail: e.target.value})}>
                </Form.Control>
            </> : <></>
        }
    </Modal.Body>
    <Modal.Footer>
        <Button onClick={() => this.setState({ads:false})} variant="secondary"> Close </Button>
        <Button onClick={() => this.add()}>Add Interview</Button>
    </Modal.Footer>
</Modal>
```

When the add interview button is clicked, it calls the *add* function:



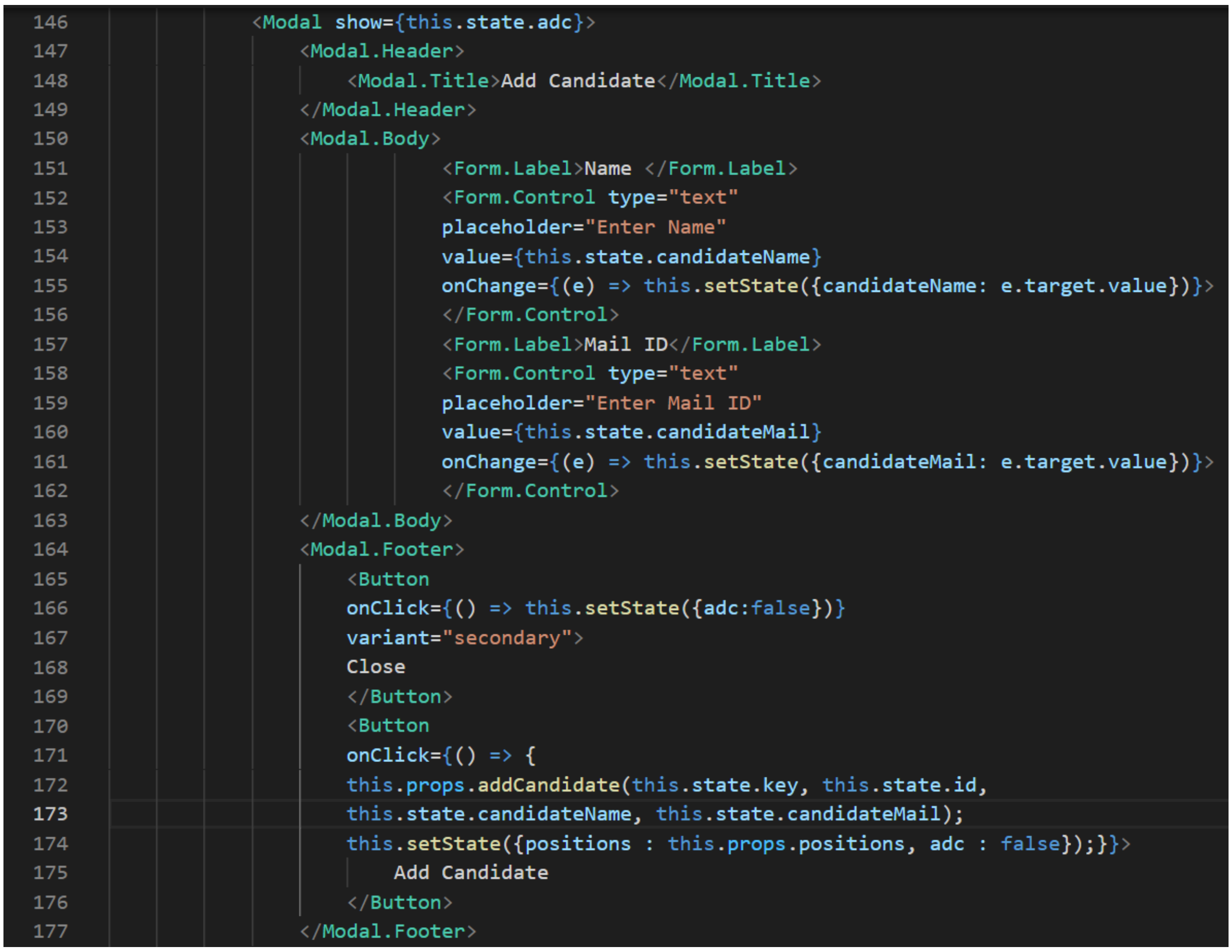
```
add(){
    this.props.addInterview(
            this.state.key, this.state.time,
            this.state.status, this.state.Date,
            this.state.candidateName, this.state.candidateMail);
    this.setState({
        ads : false,
        time : '',
        Date : '',
        status : true,
        key : -1,
        id: -1,
        candidateMail: '',
        candidateName:'',
    })
}
```

The *add* function calls the *addInterview* prop function passed down from the *App* component and then resets the state variables so that the fields are empty when a new interview is added. The *addInterview* prop from the *App* component:



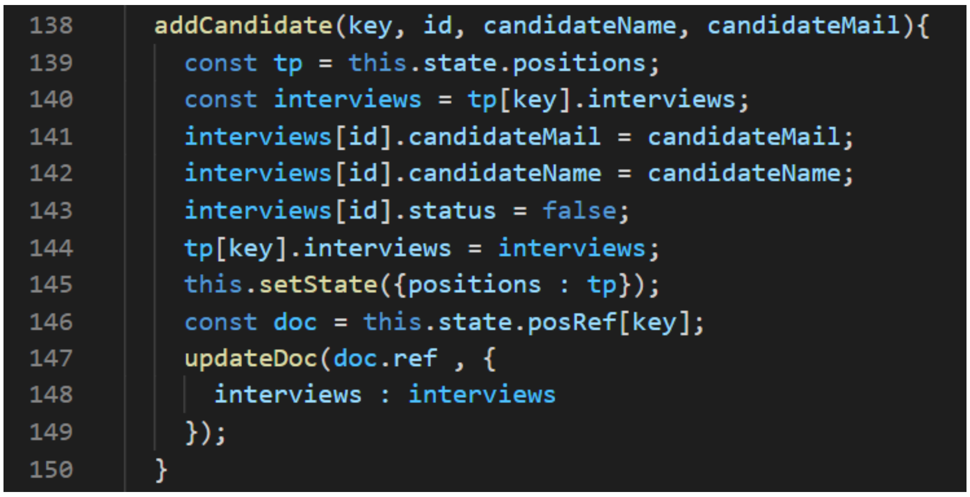
```
addInterview(key, time, status, date, name, mail) {
  let tp = this.state.positions;
  /* Local temporary positions copy is made since we need to replace the whole array for react to
  update the GUI automatically*/
  tp[key].interviews.push({
    time : time,
    status : status,
    Date : date,
    candidateName : name,
    candidateMail : mail
  });
  //The above lines add a new interview to the local positions array
  const interviews = tp[key].interviews;
  const doc = this.state.posRef[key];
  updateDoc(doc.ref , {
    interviews : interviews
  });
  /*The above lines updates the server with the new interview using the updateDoc function
  imported from the firebase library*/
  this.setState({positions : tp});
}
```

In case, the interview is occupied the interview card mentions the candidate’s name and mail otherwise it has a button to add candidate (refer to $Home.js$ lines 51-53 in appendix or page 8). When the add candidate button is clicked it sets the adc state variable to true opening up the add candidate modal window and it also sets the key and id state variables to the index of the position in the $positions$ array and the index of the interview in the $interviews$ array of the position respectively.



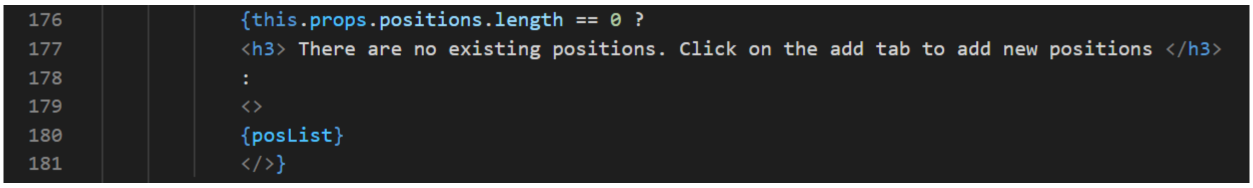
```
            <Modal show={this.state.adc}>
                <Modal.Header>
                    <Modal.Title>Add Candidate</Modal.Title>
                </Modal.Header>
                <Modal.Body>
                            <Form.Label>Name </Form.Label>
                            <Form.Control type="text"
                            placeholder="Enter Name"
                            value={this.state.candidateName}
                            onChange={(e) => this.setState({candidateName: e.target.value})}>
                            </Form.Control>
                            <Form.Label>Mail ID</Form.Label>
                            <Form.Control type="text"
                            placeholder="Enter Mail ID"
                            value={this.state.candidateMail}
                            onChange={(e) => this.setState({candidateMail: e.target.value})}>
                            </Form.Control>
                </Modal.Body>
                <Modal.Footer>
                    <Button
                    onClick={() => this.setState({adc:false})}
                    variant="secondary">
                    Close
                    </Button>
                    <Button
                    onClick={() => {
                    this.props.addCandidate(this.state.key, this.state.id,
                    this.state.candidateName, this.state.candidateMail);
                    this.setState({positions : this.props.positions, adc : false});}}>
                        Add Candidate
                    </Button>
                </Modal.Footer>
```

The Add candidate modal window has fields for the candidate's mail and name and a button a to call the $addCandidate$ prop function which uses the key and id state variables to locate the interview that the candidate is being added to. The $addCandidate$ prop function then updates the server and the state variable.



```
addCandidate(key, id, candidateName, candidateMail){
  const tp = this.state.positions;
  const interviews = tp[key].interviews;
  interviews[id].candidateMail = candidateMail;
  interviews[id].candidateName = candidateName;
  interviews[id].status = false;
  tp[key].interviews = interviews;
  this.setState({positions : tp});
  const doc = this.state.posRef[key];
  updateDoc(doc.ref , {
    interviews : interviews
  });
}
```

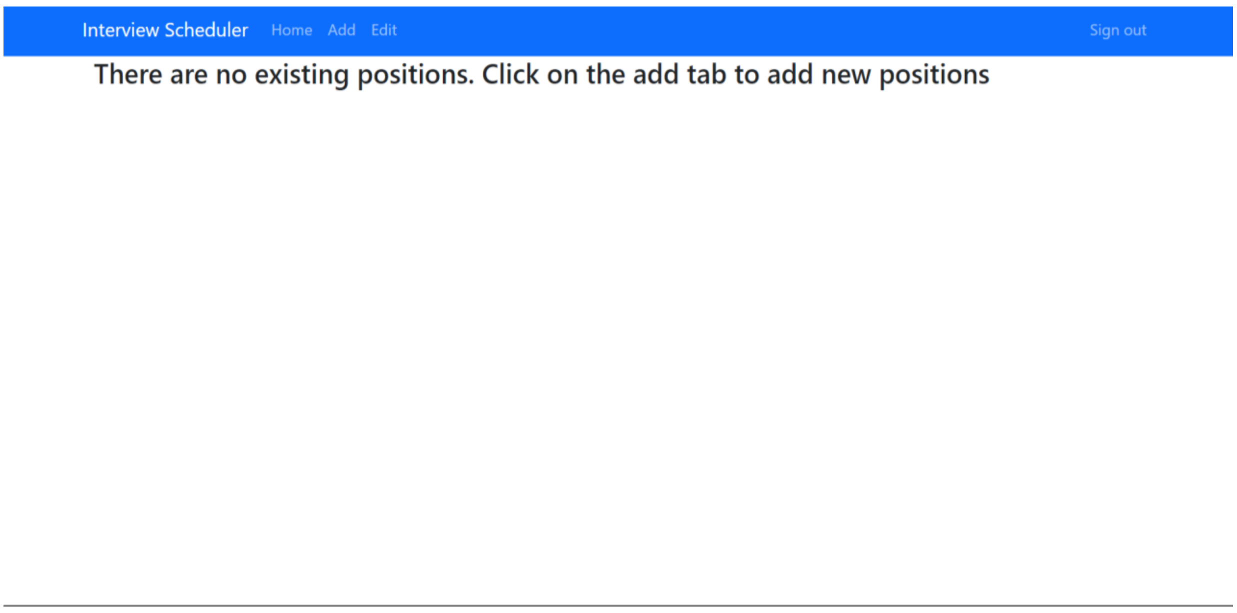
Finally, we come to the return function of the home page which handles GUI for the *Home* component.



```
            {this.props.positions.length == 0 ?
            <h3> There are no existing positions. Click on the add tab to add new positions </h3>
            :
            <>
            {posList}
            </>}
```

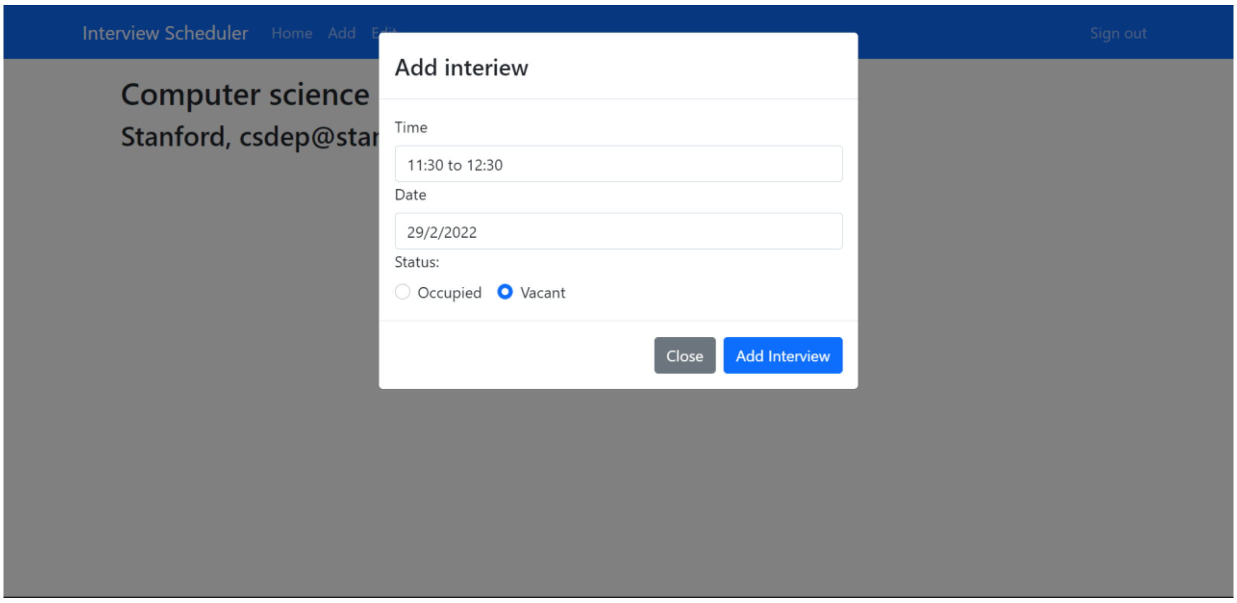
If there are positions to be displayed the *posList* GUI component is rendered which is defined on line 34 on page 7 (It consists of the positions and interviews) else it shows a message saying that there are no existing positions and tells the users to add them using the add tab.

Finished screenshots of the home page:



There are no existing positions. Click on the add tab to add new positions

Screenshot 2: No existing pictures



Interview Scheduler Home Add Edit Sign out

Computer science

Stanford, csdep@star

Add interiew

Time

11:30 to 12:30

Date

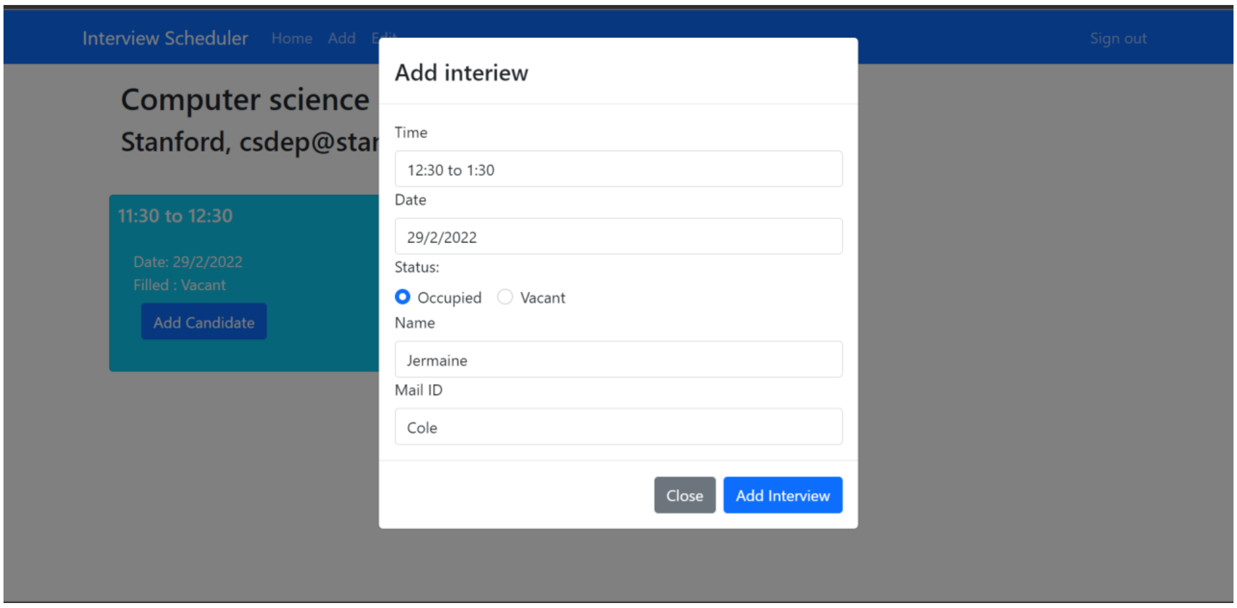
29/2/2022

Status:

Occupied [✓ Vacant]

Close | Add Interview

Screenshot 3: Add interview modal window for vacant interview



Interview Scheduler Home Add Edit Sign out

Computer science

Stanford, csdep@star

11:30 to 12:30

Date: 29/2/2022

Filled : Vacant

Add Candidate

Add interiew

Time

12:30 to 1:30

Date

29/2/2022

Status:

[✓ Occupied] Vacant

Name

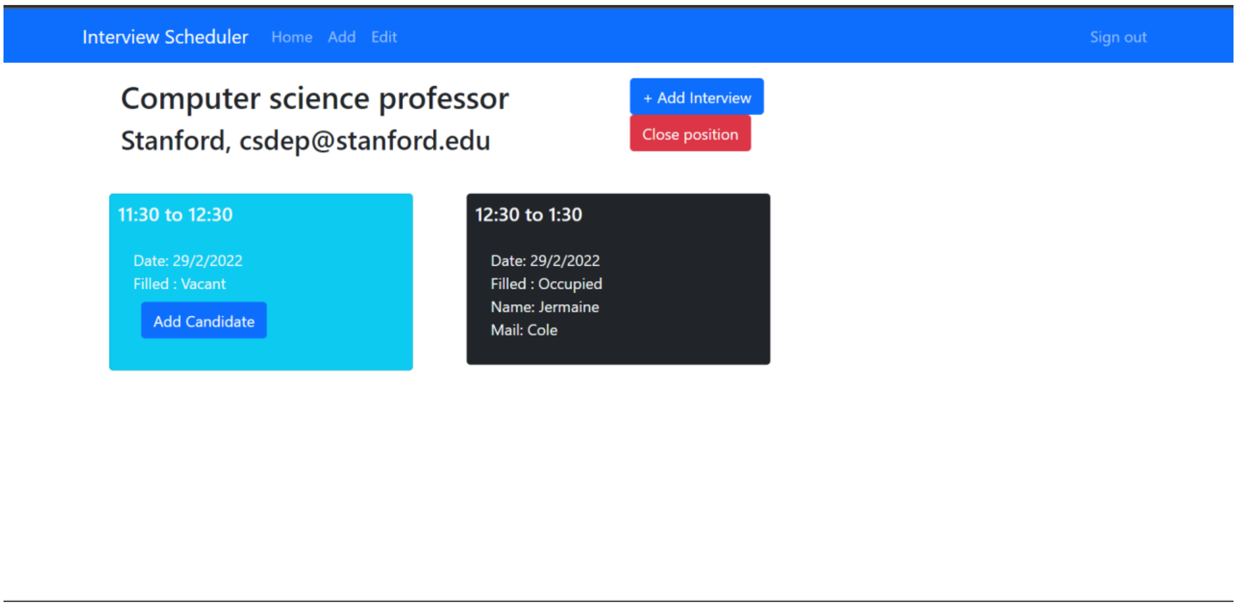
Jermaine

Mail ID

Cole

Close | Add Interview

Screenshot 4: Add interview modal window for occupied interview

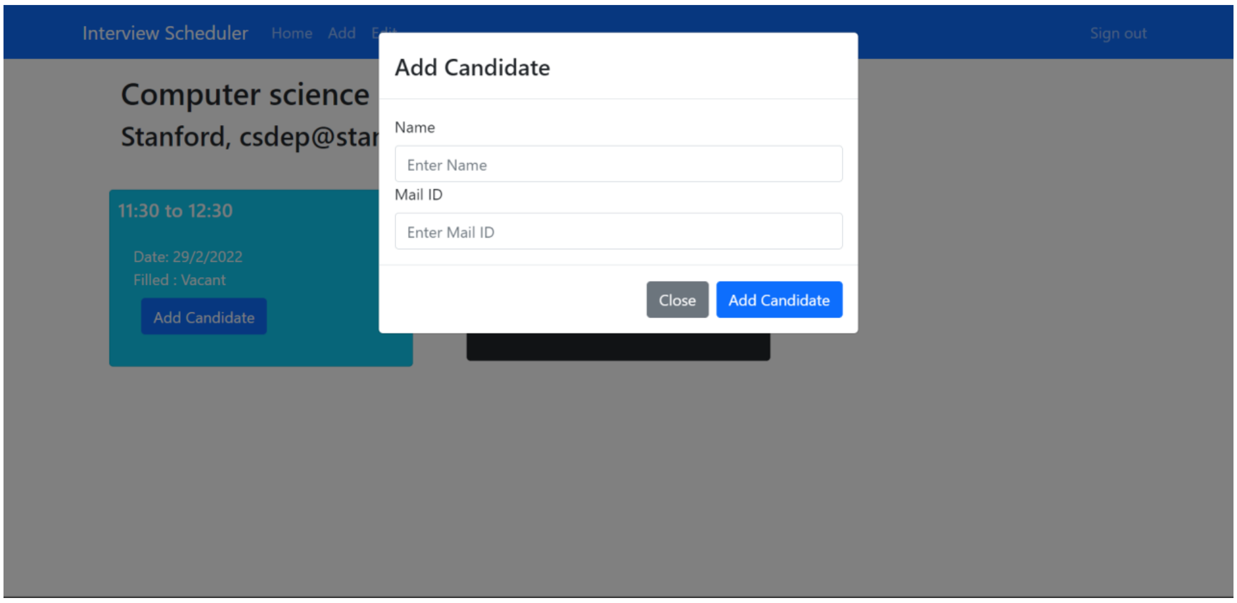


Computer science professor
Stanford, csdep@stanford.edu

11:30 to 12:30
Date: 29/2/2022
Filled : Vacant

12:30 to 1:30
Date: 29/2/2022
Filled : Occupied
Name: Jermaine
Mail: Cole

Screenshot 5: Home page with positions

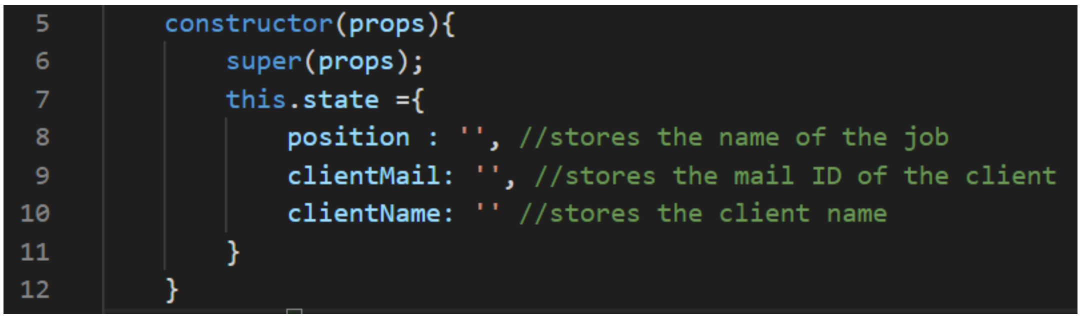


Add Candidate
Name
Enter Name
Mail ID
Enter Mail ID

Screenshot 6: Add candidate modal window

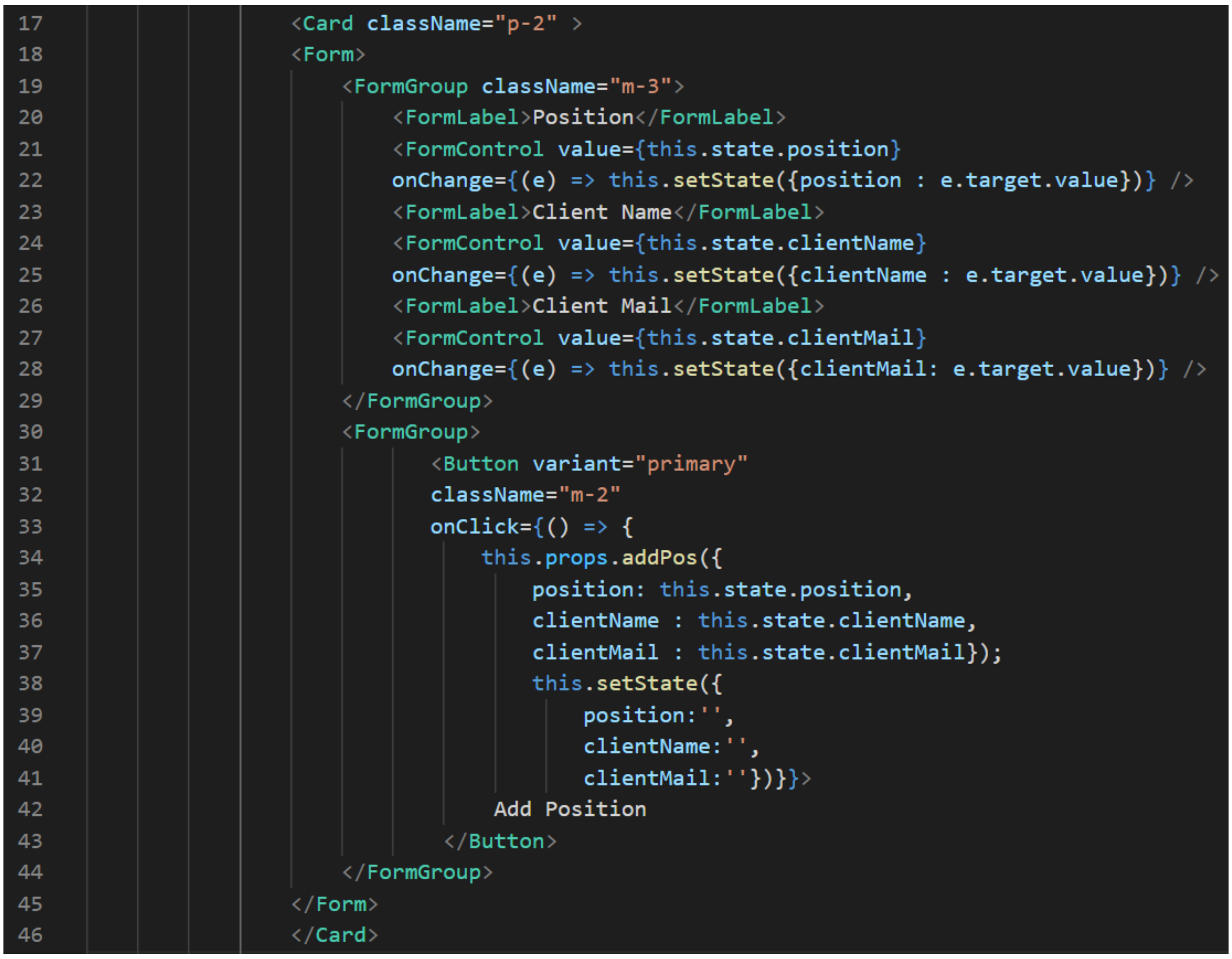
Add component:

The Add component has the simple function of allowing the user to add new positions. The state variables for the component are:



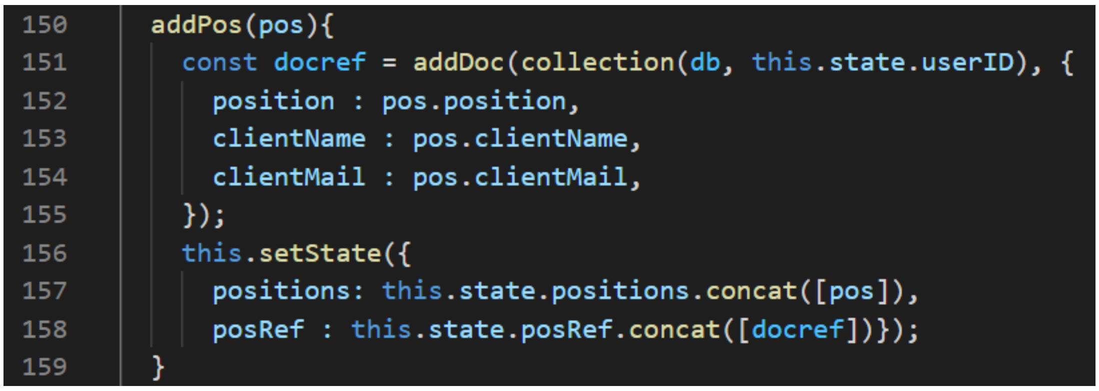
```
    constructor(props){
        super(props);
        this.state ={
            position : '', //stores the name of the job
            clientMail: '', //stores the mail ID of the client
            clientName: '' //stores the client name
        }
    }
```

The GUI for the component is:



```
                <Card className="p-2" >
                <Form>
                    <FormGroup className="m-3">
                        <FormLabel>Position</FormLabel>
                        <FormControl value={this.state.position}
                        onChange={(e) => this.setState({position : e.target.value})} />
                        <FormLabel>Client Name</FormLabel>
                        <FormControl value={this.state.clientName}
                        onChange={(e) => this.setState({clientName : e.target.value})} />
                        <FormLabel>Client Mail</FormLabel>
                        <FormControl value={this.state.clientMail}
                        onChange={(e) => this.setState({clientMail: e.target.value})} />
                    </FormGroup>
                    <FormGroup>
                           <Button variant="primary"
                           className="m-2"
                           onClick={() => {
                               this.props.addPos({
                                   position: this.state.position,
                                   clientName : this.state.clientName,
                                   clientMail : this.state.clientMail});
                                   this.setState({
                                       position:'',
                                       clientName:'',
                                       clientMail:''})}}>
                                Add Position
                             </Button>
                    </FormGroup>
                </Form>
                </Card>
```

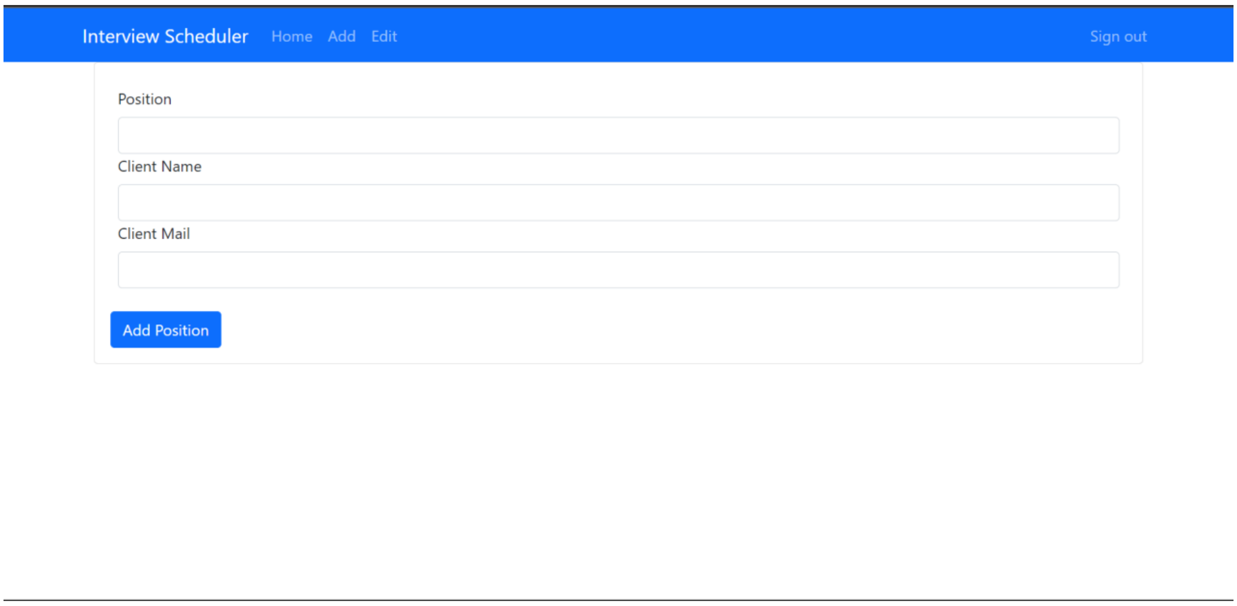
It has three fields to input the $position$, $clientMail$ and $clientName$ state variables. It has an Add position button which calls the $addPos$ prop function which adds the position to the state of the program and updates the server.



```
addPos(pos){
  const docref = addDoc(collection(db, this.state.userID), {
    position : pos.position,
    clientName : pos.clientName,
    clientMail : pos.clientMail,
  });
  this.setState({
    positions: this.state.positions.concat([pos]),
    posRef : this.state.posRef.concat([docref])});
}
```

The $addPos$ function creates a URL reference $docref$ which is concatenated to the $posRef$ array and the new position is added to the $position$s array.

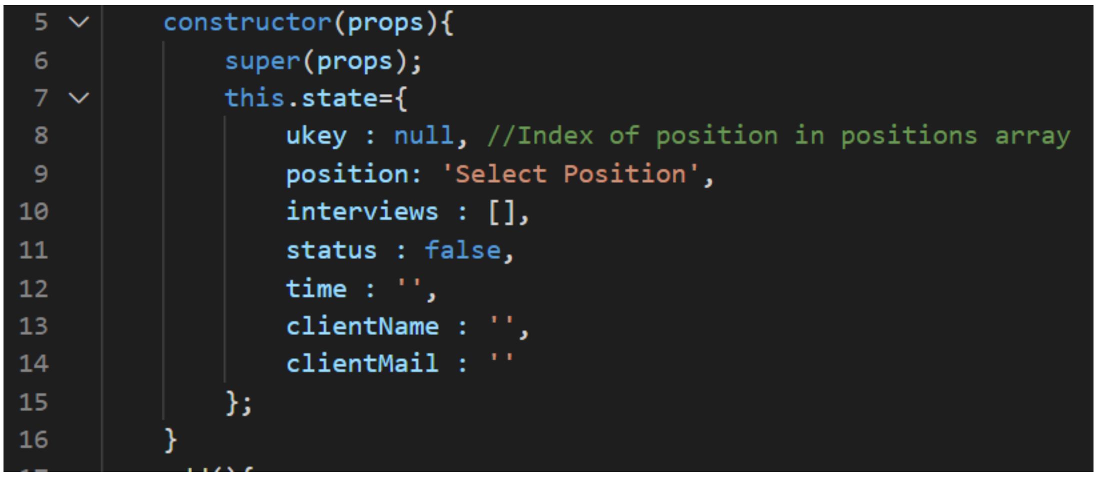
Finished screenshot of Add component:



Screenshot 7: Add page

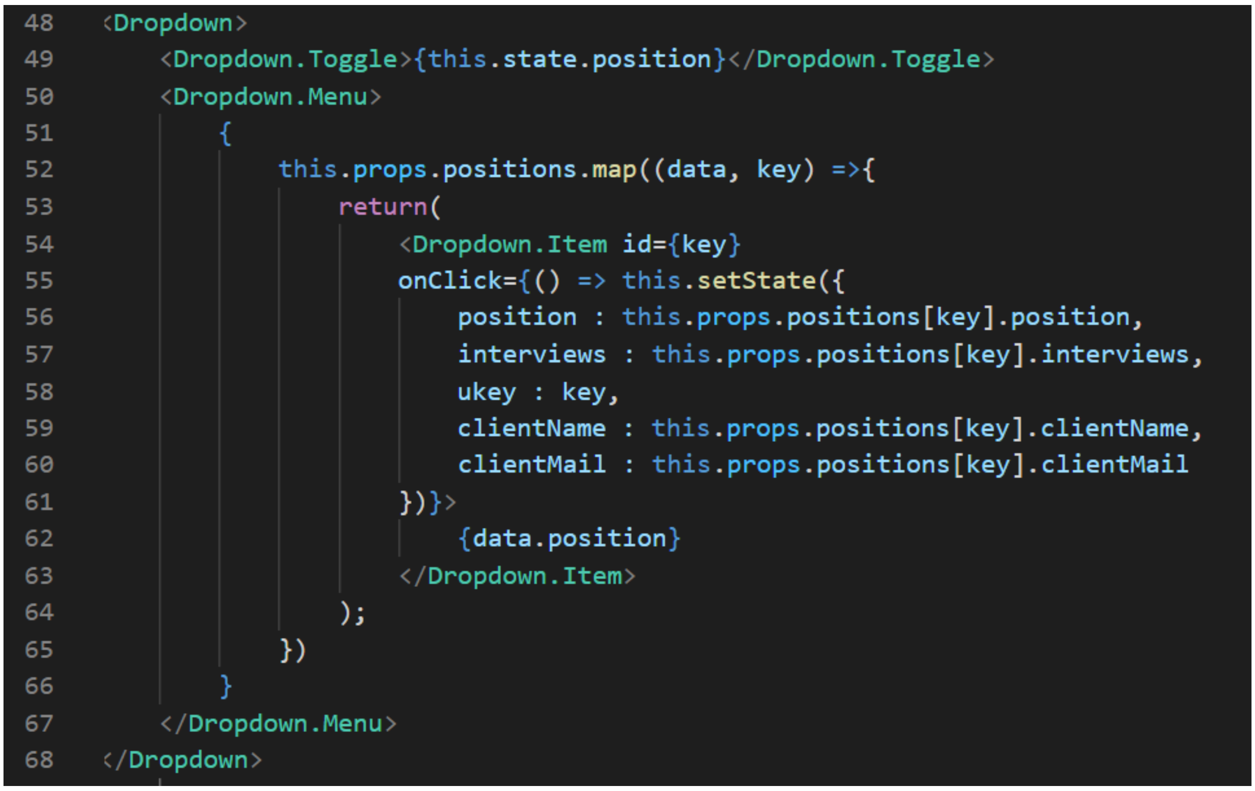
***Edit* component:**

The *Edit* component has the purpose of editing existing interviews. The *Edit* component reuses the code for the interview cards code from the *Home* component which is available on page 8. The state variables for the component are:



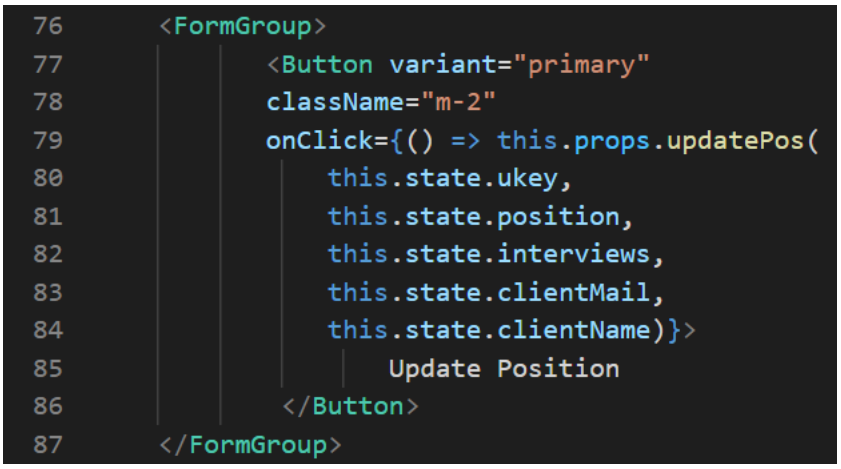
```
constructor(props){
    super(props);
    this.state={
        ukey : null, //Index of position in positions array
        position: 'Select Position',
        interviews : [],
        status : false,
        time : '',
        clientName : '',
        clientMail : ''
    };
}
```

These state variables store the attributes of the position that’s currently being edited. The component has a dropdown box from which the user can select the position to be edited.



```
<Dropdown>
    <Dropdown.Toggle>{this.state.position}</Dropdown.Toggle>
    <Dropdown.Menu>
        {
            this.props.positions.map((data, key) =>{
                return(
                    <Dropdown.Item id={key}
                    onClick={() => this.setState({
                        position : this.props.positions[key].position,
                        interviews : this.props.positions[key].interviews,
                        ukey : key,
                        clientName : this.props.positions[key].clientName,
                        clientMail : this.props.positions[key].clientMail
                    })}>
                        {data.position}
                    </Dropdown.Item>
                );
            })
        }
    </Dropdown.Menu>
</Dropdown>
```

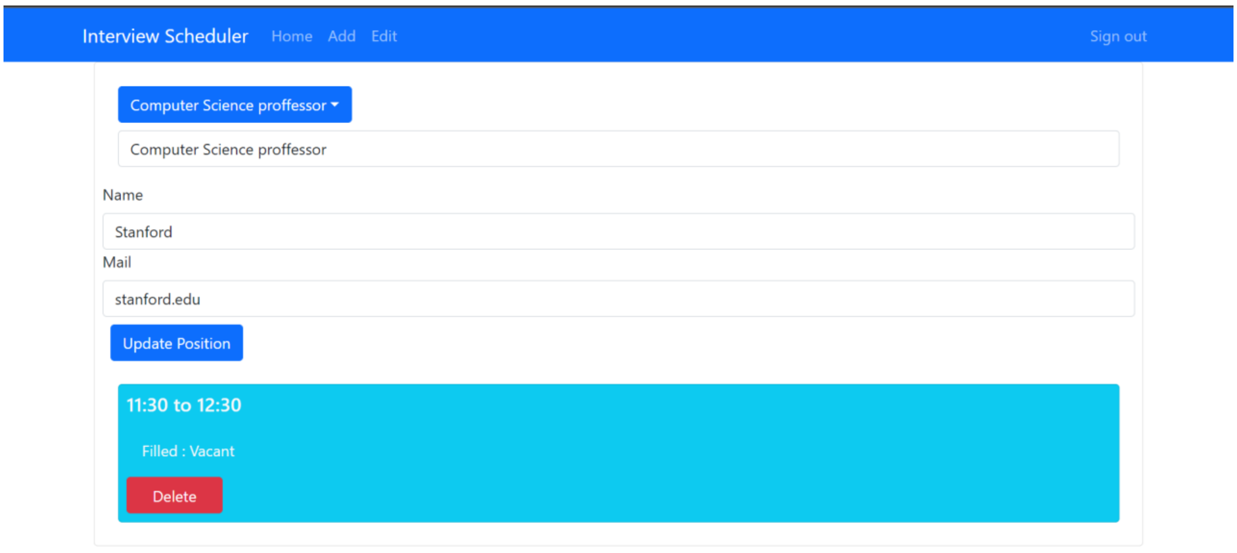
Other than the dropdown list the component reuses code from the Add component to make three fields to edit the position name and client name and mail. Finally, it has an update position button which calls the $updatePos$ prop function.



```
<FormGroup>
        <Button variant="primary"
        className="m-2"
        onClick={() => this.props.updatePos(
            this.state.ukey,
            this.state.position,
            this.state.interviews,
            this.state.clientMail,
            this.state.clientName)}>
                Update Position
        </Button>
</FormGroup>
```

The $updatePos$ function uses the index of the position to find its URL reference and update the server with the newly edited position. Similarly, the state is also update.

Finished screenshot of edit page:



Computer Science proffessor

Name

Stanford

Mail

stanford.edu

11:30 to 12:30

Filled : Vacant

Screenshot 8: Edit component

Word Count: 1050