Tutorial 1: How to Create the Apache Cordova Environment, Create Login System, and Connect to HiveMQTT Server

Apache Cordova is an open source software which could be used to build cross platform apps with HTML, CSS, and JavaScript integrated with either Xcode or Android Studio. Xcode is an official platform developed by Apple to create iOS apps and Android Studio is the same but created by Google and designed only to build Android apps for developers.

To begin, we first must ensure that you have node.js installed on your computer. Please visit here to install the software: https://nodejs.org/en/download/package-manager

When you have installed node.js, please then use the npm ultitlty to install Apache Cordova by running the following command on the terminal if you have MacOS or Linux:

```
npm install -g cordova
```

However, if you have Windows, the command is a bit different. Please type the command below on your Windows terminal:

```
C:\>npm install -g cordova
```

Apache Cordova is now installed and we could begin creating our first app with it. Let us name our first project “HelloWorld.” Please type the following command:

```
$ cordova create hello com.example.hello HelloWorld
```

For the app to work, you must access the folder where the project is located in the terminal. The example below created the “HelloWorld” app in a folder called “hello.” Please cd into the appropriate folder.

Next, we want to add the platforms that we are targeting. For our tutorial, we plan on only creating iOS apps. However, before we could create our first iOS app, we do need a few software installed first. We need Xcode, the iOS-deploy tools, and CocoaPods. Remember that CocoaPods is a software that allows code to be implemented on Xcode. To install the iOS-deploy tools, please install Homebrew here at https://brew.sh/ on their official website. Please run the following command to install the iOS-deploy tools using Homebrew:

```
$ brew install ios-deploy
```

Then, install CocoaPods here at https://cocoapods.org/ for the software. However, you do need to also install it on the terminal. Please type the following command:

```
$ sudo gem install cocoapods
```

When all requirements are installed, please type the following command:

```
$ cordova platform add ios
```

The iOS platform is now ready for use. Please type the following command to run the project we made earlier:

```
$ open ./platforms/ios/HelloWorld.xcworkspace/
```

Xcode should appear and on the left, there should be a file structure. There should be a specific folder named “Staging.” We want to make our project files in that folder.

The default files should have a file called “index.html.” Please find the specific file because now we are going to build the sign up form. For the file, please type the following code:

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="css/test.css">
<script src="js/login.js" type="text/javascript"></script>
<meta name="viewport" content="width=device-width, initial-scale=1.0 , maximum-scale =
1.0">

<body>

  <div class="login">
    <form name="form-signin">

      <h1>SIGN IN</h1>

      <label for="userName">Username</label>
      <input type="email" id="userName" placeholder="Username" required>

      <label for="userPw">Password</label>
      <input type="password" id="userPw" placeholder="Password" required>
```

```
        <div id="remember">
        </div>
        <br>

        <input id="login_btn" type="button" value="Login" onclick="check(event)">
        <br>
        <label id="login-error"></label>
        <hr>

        <label id="links"><a href="signup.html"> Register here</a></label>
      </form>
   </div>
</body>

</html>
```

The above code allows us to create a login form with two input boxes asking for the username and the password. It also provides a link to the regirstration page, which we are going to build, where the user could make an account if he or she does not have an account.

Next, let us create another HTML file called “registepage.html” for the user to create an account. Please type the following code:

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="css/test.css">
<script src="js/login.js" type="text/javascript"></script>
<meta name="viewport" content="width=device-width, initial-scale=1.0 , maximum-scale =
1.0">

<body>
   <div class="login">
      <form name="form-register">
         <h1>REGISTER</h1>
         <label for="name">Username</label>
         <input type="email" id="name" placeholder="Username" required>
         <label for="pw">Password</label>
         <input type="password" id="pw" placeholder="Password" required>
         <label id="register-error" class="error-message"></label>
         <ul class="helper-text">
            <li class="length">Must be at least 8 characters long.</li>
            <li class="lowercase">Must contain a lowercase letter.</li>
            <li class="uppercase">Must contain an uppercase letter.</li>
```

```
            <!--                    <li class="special">Must contain a number or special
character.</li>-->
        </ul>
        <input id="rgstr_btn" type="button" value="Register" onclick="store(event)">
      </form>
   </div>
</body>

</html>
```

The code above allows us to create two text inputs where the user could type their desired username and password. Also, we state the password requirments. However, we are going to validate with JavaScript. The file structure should contain a folder called “js.” Please create another js file in the folder and name it “validate.js.” Please type the following code:

```
function store(event) {
        event.preventDefault(); // Prevent the form from submitting and reloading the page

        var name = document.getElementById('name');
        var pw = document.getElementById('pw');
        var errorLabel = document.getElementById('register-error');
        var lowerCaseLetters = /[a-z]/g;
        var upperCaseLetters = /[A-Z]/g;
        var numbers = /[0-9]/g;

        if (name.value.length === 0) {
                errorLabel.textContent = 'Please fill in username';
        } else if (pw.value.length === 0) {
                errorLabel.textContent = 'Please fill in password';
        } else if (pw.value.length < 8) {
                errorLabel.textContent = 'Password must be at least 8 characters long';
        } else if (!pw.value.match(upperCaseLetters)) {
                errorLabel.textContent = 'Please add 1 uppercase letter';
        } else if (!pw.value.match(lowerCaseLetters)) {
                errorLabel.textContent = 'Please add 1 lowercase letter';
        } else {
                localStorage.setItem('name', name.value);
                localStorage.setItem('pw', pw.value);
                errorLabel.innerHTML = 'Your account is created. <a href= "./index.html" style =
"color: white;">Go to login</a>';
        }
}

function check(event) {
```

```
        event.preventDefault(); // Prevent the form from submitting and reloading the page

        var storedName = localStorage.getItem('name');
        var storedPw = localStorage.getItem('pw');

        var userName = document.getElementById('userName');
        var userPw = document.getElementById('userPw');
        var errorLabel = document.getElementById('login-error');

        if (userName.value === storedName && userPw.value === storedPw) {
                window.location.href = "indexfile.html"
        } else {
                errorLabel.textContent = 'Error on login';
        }
}
```

The code above checks grabs the ID of the password txt box and ensures that it meets the password requirements. Also, the form grabs the ID of the text boxes of the login page and if the user enters the correct username and password, the data is saved with JavaScript Local Storage and a session is created. If not, a text would appear stating that there is an error with log in underneath the password box.

Let us create the home page for the user when he/she is logged on. We want to now create the form where the user could connect to HiveMQTT. Please create another HTML file called “home.html” and add the following code:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>HiveMQTT</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.2/mqttws31.min.js"
type="text/javascript"></script>
  <script src="js/connect.js" type="text/javascript"></script>
  <meta name="viewport" content="width=device-width, initial-scale=1.0 , maximum-scale =
1.0">
</head>

<body>
  <div class="wrapper">
    <button id="logout_btn" onclick="logout()">Logout</button>
    <h1 id="Main_heading"> <b>Connection to HiveMQTT</b></h1>
```

```
      <br>
      <br>
      <form id="connection-information-form">
        <!-- <b>Host:</b>
        <input id="host" type="text" name="host" placeholder="broker address">
        <b>Port:</b>
        <input id="port" type="text" name="port"><br>
        <b>Username and Password:</b>
        <input id="username" type="text" name="Username" placeholder="Username"><br>

        <input id="password" type="password" name="password"
placeholder="password"><br> -->
        <b>Subscription topic:</b>
        <input id="subscription_topic" type="text" name="subscription_topic">
        <br>
        <br>
        <input type="button" onclick="startConnect()" value="Connect">
        <input type="button" onclick="startDisconnect()" value="Disconnect">
        <br>
        <br><b>Publish Topic and Message:</b>
        <input id="publish" type="text" name="publish" placeholder="Topic">

        <input id="Message" type="text" name="message" placeholder="Message">
        <input type="button" onclick="publishMessage()" value="Publish">
      </form>
      <div id="messages"></div>
   </div>
</body>

</html>
```

The above code allows us to create a form where the user could enter a topic to subscribe to and also publish a topic and message to HiveMQTT. Next, we want to use JavaScript to connect to the HiveMQTT broker. Please go back to the js folder and create a folder called “connect.js.” Then, please type the following code:

```
let client, host, port, clientID, topic;

function startConnect() {
        clientID = `clientID - ${Math.floor(Math.random() * 100)}`;
        host = "broker.hivemq.com";
        port = 8000;

        appendMessage(`Connected to ${host} on port ${port}`);
```

```
        appendMessage(`Current Client ID: ${clientID}`);

        client = new Paho.MQTT.Client(host, Number(port), clientID);
        client.onConnectionLost = onConnectionLost;
        client.onMessageArrived = onMessageArrived;

        client.connect({
                onSuccess: onConnect
                // userName: document.getElementById("username").value,
                // password: document.getElementById("password").value
        });
}

function onConnect() {
        topic = document.getElementById("subscription_topic").value;
        appendMessage(`Subscribed to topic: ${topic}`);
        client.subscribe(topic);
}

function onConnectionLost(responseObject) {
        appendMessage("ERROR: Connection is lost.");
        if (responseObject.errorCode !== 0) {
                appendMessage(`ERROR: ${responseObject.errorMessage}`);
        }
}

function onMessageArrived(message) {
        console.log(`OnMessageArrived: ${message.payloadString}`);
        appendMessage(`Topic: ${message.destinationName} | Message:
${message.payloadString}`);
}

function startDisconnect() {
        client.disconnect();
        appendMessage("Disconnected.");
}

function publishMessage() {
        const msg = document.getElementById("Message").value;
        const topic = document.getElementById("publish").value;

        const message = new Paho.MQTT.Message(msg);
        message.destinationName = topic;
```

```
        client.send(message);
        // appendMessage(`Message to topic ${topic} is sent.`);
}

function appendMessage(message) {
        document.getElementById("messages").innerHTML += `<span>${message}</span><br>`;
}

function logout() {
        localStorage.removeItem('name');
        localStorage.removeItem('pw');
        window.location.href = 'index.html'; // Redirect to login page after logging out
}
```

The above code allows us to connect to the HiveMQTT broker with a default value for broker.hivemq.com at Port 8000. Then, we create functions to handle the user's input to subscribe to a topic. We also handle the user's input when a topic and message is published to the broker.