**Lebanese American University**

**Department of Computer Science and Mathematics**

**CSC599: Capstone Project**

**Smart Devices Website**

by

**Ralph Abou Jaoude**

**Christopher Chalhoub**

16 May 2022

The website aims to provide the user with the most optimal experience. We are a state-of-the-art shop, and we provide modern and up to date technologies. The clients are provided with the best support and a variety of choices so that they would not need to look elsewhere for the services we provide. The features that are provided are: 360-degree view for the products, an online chat support with an admin, and a recommendation system for the user.

Our mission at "Smart Devices" is to deliver high-quality items at the lowest possible price, and we aspire to become your new hub and destination for all your technological needs. Furthermore, all of our products are of great quality and will provide you with the satisfaction and performance you require. Our goal is to become a part of every home, and in the future, we want to be more than a referral service; we want to be an influencer in the technology scene in Lebanon for all technology users, whether they are IT professionals or regular consumers. Additionally, we provide excellent service to our clients. Customers feel at ease when they purchase at our stores because of our great care and approach to them. Furthermore, we strive to provide the most up-to-date technologies, newest generation, intense gaming performance, integrated tech gadgets, and more to our clients. Furthermore, our vision is to be known as a market leader in the region, delivering the greatest quality performance, innovation, and dedication to surpass our loyal customers' expectations.

What drove us to implement such project is that before entering this major in the university and not having any prior experience to coding or creating websites, creating a website that includes what we implemented was out of the question, we would have never imagined being able to do such a thing. What we implemented was one of our goals since we always use such websites in our daily life in order to stay up to date with the new technological advancements in this era and being able to purchase trending devices. Furthermore, we are able to customize the website's functionality and design according to what was the most practical. Moreover, in order to implement this website, there were a lot of stuff that we did not know how to implement such as encrypting the passwords that the user sends such as using the “bcrypt” which is a password-hashing function. Therefore, we acquired a great deal of information regarding stuff we did not know anything about.

**User Guide**

When the user enters the website, he/she should login by entering his email and password. If the user does not have an existing account, he should press the Sign-up button in order to go to the registration page and create an account and then go to the login page to login.

After logging in, the user is directly transferred to the home page.

The user should click the devices button to be able to view the available devices which include their name, price, and their specifications.

The user can inspect the device by clicking on the “view 360” button, then a page pops up where he is able to drag the device left or right by using the mouse to check all the angles of the device.

The user can get a recommendation by clicking on the “recommend me a device” button. Then he is transferred to a page where he is asked three consecutive questions about certain specification of the devices that he is looking for. Additionally, the user can check the device chosen by the recommendation. After being given the recommendation, he can repeat the process by a press of a button which is “Recommend Again”.

The user should click the add button of the devices he desires in order to add it to the shopping cart.

The products that are in the shopping cart can be viewed by pressing the button that is located on top left of the screen. When this button is pressed the user can view the total amount and quantity of the products, he/she can deduct, add, or modify. Moreover, the user is able to remove either a single product or all the products in order to clear the shopping cart.

To buy the products, the user should click the checkout button which is in the shopping cart section, when it is pressed, he is taken to a page where he should fill his information, then click checkout for the purchase to be completed.

If the user requires immediate assistance, by pressing “chat with an admin” button he is able to live chat with an admin in order to resolve any issues or inquiries that he has.

The user can also contact the company by entering the contact us page, he is able to fill in his name, email, and the message he wants to send the company and then click the submit button.

If the user wants to logout from his account, there is a logout button on the top right of the page in order for the user to logout and go to the home page.

# **Technologies used:**

**HTML:** The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript.

**CSS:** Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

**JavaScript:** JavaScript is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. JavaScript is used by programmers across the world to create dynamic and interactive web content like applications and browsers.

**Node.js:** Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services, but was designed with real-time, push-based architectures in mind.

**Express.js:** Express is a Node.js web application framework that provides broad features for building web and mobile applications. It is used to build a single page, multipage, and hybrid web application. It's a layer built on the top of the Node.js that helps manage servers and routes.

**MongoDB:** MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

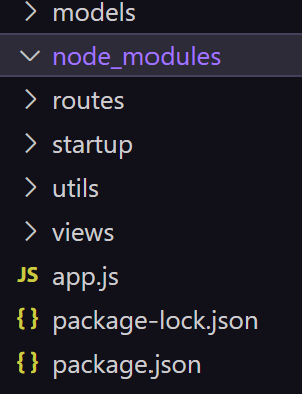
**Mongoose:** Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

**Socket.io:** Socket.io is an event-driven JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. It was developed to use open connections to facilitate real-time communication.

**Bcrypt:** Bcrypt is a password-hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher. The Bcrypt hashing function allows us to build a password security platform that scales with computation power and always hashes every password with a salt.

**Ejs:** Ejs is a simple templating language/engine that lets its user generate HTML with plain Javascript. EJS is mostly useful whenever you have to output HTML with a lot of Javascript.

# **Project Structure:**



**Models:** Contains all the files that refer to the mapping of models in the databases.

**Node\_modules:** Contains all the node modules and packages installed and used in this project.

**Routes:** Contains all the files that routes all the API calls coming from the client.

**Startup:** Contains all the files necessary to power up the server.

**Utils:** Contains all the files that provide services across the server.

**Views:** Contains all the EJS files that are used to render the views for the clients.

**app.js:** Is the file that runs the server and is usually started through it.

# Models:

**User:**

const User = mongoose.model(

  "User",

  new mongoose.Schema({

    username: String,

    password: String,

    email: String,

    isAdmin: {

      type: Boolean,

      default: false,

    },

  })

);

**Device:**

const Device = mongoose.model(

  "Device",

  new mongoose.Schema({

    title: String,

    price: Number,

    description: String,

    img: String,

    categories: [String],

  })

);

**Message:**

const Message = mongoose.model(

  "Message",

  new mongoose.Schema({

    name: String,

    email: String,

    message: String,

  })

);

**Chat:**

const Chat = mongoose.model(

  "Chat",

  new mongoose.Schema({

    users: [{ type: String }],

    sender: String,

    receiver: String,

    message: String,

  })

);

# **Functionalities**

**Login route:**

router.post("/", async *(req, res)* => {

  try {

    console.log("Login received....");

    let foundUser = await User.findOne({ email: req.body.email });

    if (foundUser) {

      let submittedPass = req.body.password;

      let storedPass = foundUser.password;

      const passwordMatch = await bcrypt.compare(submittedPass, storedPass);

      if (passwordMatch) {

        let username = foundUser.username;

        let token = jwt.sign({ username: username, isAdmin: foundUser.isAdmin }, PrivateKey);

        console.log("Sending token...");

        res.status(200).json({ token: token });

      } else {

        res.sendStatus(400);

      }

    }

  } catch {

    res.send("Internal server error");

  }

});

**Signup route:**

router.post("/", async *(req, res)* => {

  try {

    let foundUser = await User.findOne({ email: req.body.email });

    if (!foundUser) {

      let hashPassword = await bcrypt.hash(req.body.password, 10);

      let newUser = new User({

        username: req.body.username,

        email: req.body.email,

        password: hashPassword,

        isAdmin: req.body.isAdmin ?? false,

      });

      await newUser.save();

      res.status(200).send(newUser);

    } else {

      res.sendStatus(400);

    }

  } catch {

    res.status(500).send(error.message);

  }

});

**Get users route:**

router.get("/", async *(req, res)* => {

  try {

    const users = await User.find();

    res.status(200).send(users);

  } catch (err) {

    res.status(500).send(error.message);

  }

});

**Update user route:**

router.put("/:id/", async *(req, res)* => {

  try {

    let user = await User.findOne({ \_id: req.params.id });

    if (user) {

      user.username = req.body.username;

      user.email = req.body.email;

      user.isAdmin = req.body.isAdmin;

      await user.save();

      res.status(200).send(user);

    } else {

      res.sendStatus(400);

    }

  } catch (err) {

    res.status(500).send(err.message);

  }

});

**Delete user route:**

router.delete("/:id", async *(req, res)* => {

  try {

    const users = await User.findByIdAndDelete(req.params.id);

    res.status(200).send(users);

  } catch (err) {

    res.status(500).send(error.message);

  }

});

**Get all chats by user route:**

router.get("/:username", async *(req, res)* => {

  try {

    const messages = await Chat.find({ users: req.params.username });

    res.status(200).send(messages);

  } catch (err) {

    res.status(500).send(err.message);

  }

});

**Add a new chat message by user route:**

router.post("/:username", async *(req, res)* => {

  try {

    const chat = new Chat({

      users: ["admin", req.params.username],

      sender: req.body.sender,

      receiver: req.body.receiver,

      message: req.body.message,

    });

    await chat.save();

  } catch (err) {}

});

**Get all Contact us messages:**

router.post("/", async *(req, res)* => {

  try {

    console.log("Submitting form...");

    console.log(req.body);

    let newMessage = new Message({

      name: req.body.name,

      email: req.body.email,

      message: req.body.message,

    });

    await newMessage.save();

    res.sendStatus(200);

  } catch (err) {

    res.send("Internal Server Error");

  }

});

**Add a new Contact us message:**

router.post("/", async *(req, res)* => {

  try {

    console.log("Submitting form...");

    console.log(req.body);

    let newMessage = new Message({

      name: req.body.name,

      email: req.body.email,

      message: req.body.message,

    });

    await newMessage.save();

    res.sendStatus(200);

  } catch (err) {

    res.send("Internal Server Error");

  }

});

**Get all devices route:**

router.get("/", async *(req, res)* => {

  console.log("Getting Devices...");

  let devices = await Device.find();

  res.send(devices);

});

**Get device by id route:**

router.get("/:id", async *(req, res)* => {

  try {

    const device = await Device.findById(req.params.id);

    if (!device) {

      return res.status(404).send("Device with given ID does not exist!");

    }

    res.status(200).send(device);

  } catch (error) {

    res.status(500).send(error.message);

  }

});

**Add a new device route:**

router.post("/", async *(req, res)* => {

  try {

    let device = new Device({

      title: req.body.title,

      price: req.body.price,

      description: req.body.description,

      img: req.body.img,

      categories: req.body.categories,

    });

    await device.save();

    res.status(200).send(device);

  } catch (error) {

    res.status(500).send(error.message);

  }

});

**Update a device by id:**

router.put("/:id", async *(req, res)* => {

  try {

    const device = await Device.findByIdAndUpdate(req.params.id, {

      title: req.body.title,

      price: req.body.price,

      description: req.body.description,

      img: req.body.img,

      categories: req.body.categories,

    });

    if (!device) {

      return res.status(404).send("Device with given ID does not exist!");

    }

    await device.save();

    res.status(200).send(device);

  } catch (error) {

    res.status(500).send(error.message);

  }

});

**Delete device by id:**

router.delete("/:id", async *(req, res)* => {

  try {

    const device = await Device.findByIdAndDelete(req.params.id);

    if (!device) {

      return res.status(404).send("Device with given ID does not exist!");

    }

    res.status(200).send(device);

  } catch (error) {

    res.status(500).send(error.message);

  }

});

**Starting up the database:**

*()* => {

  mongoose

    .connect("mongodb://localhost/smartdevices")

    .then(*()* => {

      console.log("Connected...");

      createAdmin();

    })

    .catch(*(err)* => {

      console.log("Error Connecting to MongoDB...", err);

    });

};

**Redirect user calls to routes in the server:**

*(app)* => {

  app.get("/", *(req, res)* => {

    res.render("pages/index");

  });

  app.get("/devices", *(req, res)* => {

    res.render("pages/devices");

  });

  app.get("/about-us", *(req, res)* => {

    res.render("pages/about-us");

  });

  app.get("/contact-us", *(req, res)* => {

    res.render("pages/contact-us");

  });

  app.get("/recommend", *(req, res)* => {

    res.render("pages/recommend");

  });

  app.get("/register", *(req, res)* => {

    res.render("pages/register");

  });

  app.get("/login", *(req, res)* => {

    res.render("pages/login");

  });

  app.get("/checkout", *(req, res)* => {

    res.render("pages/checkout");

  });

  app.get("/admin", *(req, res)* => {

    res.render("pages/admin");

  });

  app.use("/api/users", users);

  app.use("/api/auth", auth);

  app.use("/api/form", contact);

  app.use("/api/devices", devices);

};

**Initiating the server socket connection:**

*(server)* => {

  io = require("socket.io")(server);

  console.log("Initiating server socket...");

  io.on("connection", *(socket)* => {

    console.info(`Client connected [id=${socket.id}]`);

    socket.join(socket.request.\_query.id);

    socket.on("identify", async *(socket)* => {

      console.log("Socket:", socket.username);

      const messages = await getMessages(socket.username);

      io.emit("messages", { messages });

    });

    socket.on("send-message", async *(socket)* => {

      console.log(`From "${socket.username}": ${socket.message}`);

      const message = await sendMessage(

        socket.sender,

        socket.receiver,

        socket.message

      );

      io.emit("new-message", { message });

    });

    socket.on("disconnect", *()* => {

      console.info(`Client disconnected [id=${socket.id}]`);

    });

  });

};

**Initiating the client socket connection:**

socket.on("connect", *()* => {

  socket.emit("identify", { username });

});

socket.on("messages", *(server)* => {

  displayMessages(server.messages);

});

socket.on("new-message", *(server)* => {

  displayNewMessage(server.message);

});

# **Techniques Used**

**CRUD operations (present in the routes above)**

**LinkedList:**

const choice122 = [

  { choice: "Portable", next: null },

  { choice: "Stationary", next: null },

];

const choice121 = [

  { choice: "Budget", next: null },

  { choice: "Performance", next: null },

];

const choice12 = [

  { choice: "Gaming", next: choice121 },

  { choice: "Business", next: choice122 },

  { choice: "Programming", next: choice121 },

];

//Phone Choices

const choice111 = [

  { choice: "Android", next: null },

  { choice: "IOS", next: null },

];

const choice11 = [

  { choice: "Budget", next: choice111 },

  { choice: "Performance", next: choice111 },

];

//Main Choices

const choice1 = [

  { choice: "Phone", next: choice11 },

  { choice: "Laptop", next: choice12 },

];

**Recursive Decision Tree:**

const decide = *(choice)* => {

  chosen.push(choice.choice);

  if (choice.next == null) {

    console.log(chosen);

    displayChoice([]);

    getResult(chosen);

    return;

  }

  displayChoice(choice.next);

};

const displayChoice = *(choice)* => {

  recommendContainer.innerHTML = "";

  if (choice == null) {

    return;

  }

  choice.forEach(*(c)* => {

    const div = document.createElement("div");

    div.className = "recommend";

    div.addEventListener("click", *()* => {

      decide(c);

    });

    div.textContent = c.choice;

    recommendContainer.appendChild(div);

  });

};

Here we would wait for the user to choose an option and then we would append it to the chosen options list and proceed with the next set of choices. If the next set doesn’t exist (null), we proceed to get the result.

**Result:**

const getResult = *()* => {

  let maxCount = 0;

  for (let i = 0; i < devices.length; i++) {

    devices[i].count = 0;

  }

  chosen.forEach(*(choice)* => {

    for (let i = 0; i < devices.length; i++) {

      if (devices[i].categories.includes(choice)) {

        if (choice == "IOS" || choice == "Android") {

          console.log("I AM HERE");

          devices[i].count += 5;

        }

        devices[i].count++;

      }

      if (maxCount < devices[i].count) maxCount = devices[i].count;

    }

  });

  const chosenDevices = [];

  for (let i = 0; i < devices.length; i++) {

    if (devices[i].count == maxCount) chosenDevices.push(devices[i]);

  }

  chosenDevices.forEach(*(device)* => {

    displayDevice(device);

  });

  recommendResult.innerHTML += `<br /><br /><div><button class="button"><a href="/recommend">Recommend Again</a></button></div>`;

};

Here we would recommend the device(s) that most suite the user’s choices by checking each device’s categories. If the choice exists in one of them then the device’s weight is upped a level (or 5 if it is IOS or Android). The device(s) with the highest weight will be returned to the user.

# **References**

<https://nodejs.org/en/>

<https://expressjs.com/en/4x/api.html>

<https://socket.io/docs/v4/>

<https://github.com/kelektiv/node.bcrypt.js>

<https://www.w3schools.com>

<https://www.geeksforgeeks.org>

https://devhints.io