Criteria C: Development
Word count: 1183

List of techniques used in developing the solution

Name of technique	Page nos.	Success criterions
Referencing and embedding relational methods	1, 2, 5	2, 3
Data validation in DB schema	1	16
Mongoose functions to perform DB transactions	2, 5	2, 3, 5, 9
Try/catch statements for error handling	2	16
Input sanitization	2	16
Regular expressions to search name	2	2
String formatting to insert data into HTML/CSS templates	3	7, 8
SMTP connection using Nodemail library	3	7, 8
Environment variables to store private data	3	12
Business logic/mathematics and date parsing/operations	4	8
IO operations and data parsing	5	10
Nested loops	5	10
Express Middlewares	6	13, 14, 16, 2
Express Routes: use of HTTP methods and response codes	7	2, 3, 5, 9
Asynchronous programming	7	17 (speedy UX)
Use of cron schedules and mongodump utility	7	11
Use of postman, mongosh, and chrome devtools	8	18 (ease of development)
React state to maintain application state	9, 10, 11, 13, 14	2, 3, 4
React props to pass arguments to components	9, 11, 14	2, 3, 4, 18
API queries to fetch and update data	10, 11, 14	2, 3, 6, 9
localStorage to store passkey and building	8, 11, 15	1, 15, 13
Javascript embedding for dynamic rendering with data	10, 11	2, 3
Conditional rendering	9, 11	1, 2, 17
Flexbox to structure UI and CSS classes	12	17
useEffect hook	13	2, 3
Callback functions	9, 11, 14	2, 5, 9, 15
Modularization of code and following conventions	9, 15	18

List of libraries used (NPM package manager)

Name of library and purpose	Page nos.	Success criterions
Backend		
cors: Cors middleware for express	6	All
dotenv: Read the .env file for senstitive environment variables	3	12
email-validator: Validate whether a string is an email	1	16
escape-string-regex: Sanitize regular expressions	2	16
express: Web server framework library	6, 7	2-9
express-rate-limit: Rate limiting middleware for express	6	14

mongoose: ODM library for MongoDB databases	1, 2, 5	2-10
Frontend		
fontawesome: For icons such as the time and warning icon	13	17
react-bootstrap: For UI elements such as button and input	10, 11, 14	17
react: Javascript framework for UI	10-14	1-9, 17
axios: Making requests to the API	14	2-9
react-toastify: Showing toasts to the user	9, 13	17

Backend

Student.js – Schema and Model for Student collection (CRUD logic layer)

```
import mongoose from "mongoose";
const { Schema, model } = mongoose;
import validator from "email-validator";
import lateArrivalSchema from "../schema/lateArrival.js";

const studentSchema = new Schema({
  // Name of the student, should be a string and is required and unique
  name: { type: String, required: true, unique: true },
  // Grade of the student, should be a number between 6 and 12, and is required
  grade: {
    type: Number,
    required: true,
    validate: (grade) => Number.isInteger(grade) && grade >= 6 && grade <= 12,
  },
  // Section of the student, should be one of the specified letters, and is required
  section: {
    type: String,
    required: true,
    validate: (section) => ["A", "B", "C", "D", "E", "F"].includes(section),
  },
  // Email of the student, should be a valid email format, and is required
  email: { type: String, required: true, validate: validator.validate },
  // Guardians of the student, should contain at least one guardian, each with a name and email, an
  guardians: {
    type: [
      {
        name: { type: String, required: true },
        email: { type: String, required: true, validate: validator.validate },
      },
    ],
    validate: (guardians) => guardians.length > 0,
    required: true,
  },
  // Teachers of the student, should contain exactly two teacher references, and is required
  teachers: {
    type: [{ type: Schema.Types.ObjectId, ref: "Teacher" }],
    validate: (teachers) => teachers.length == 2,
    required: true,
  },
  // Unique identifier for the student, should be a string and is required and unique
  id: { type: String, required: true, unique: true },
  // Late arrivals of the student, an array containing late arrival information
  lateArrivals: {
    type: [lateArrivalSchema],
  },
});

// Creating indexes for quick access and ensuring uniqueness
studentSchema.index({ name: 1 }, { unique: true });
studentSchema.index({ id: 1 }, { unique: true });

const Student = model("Student", studentSchema);
export default Student;
```

Data validation

Use of external library to validate email

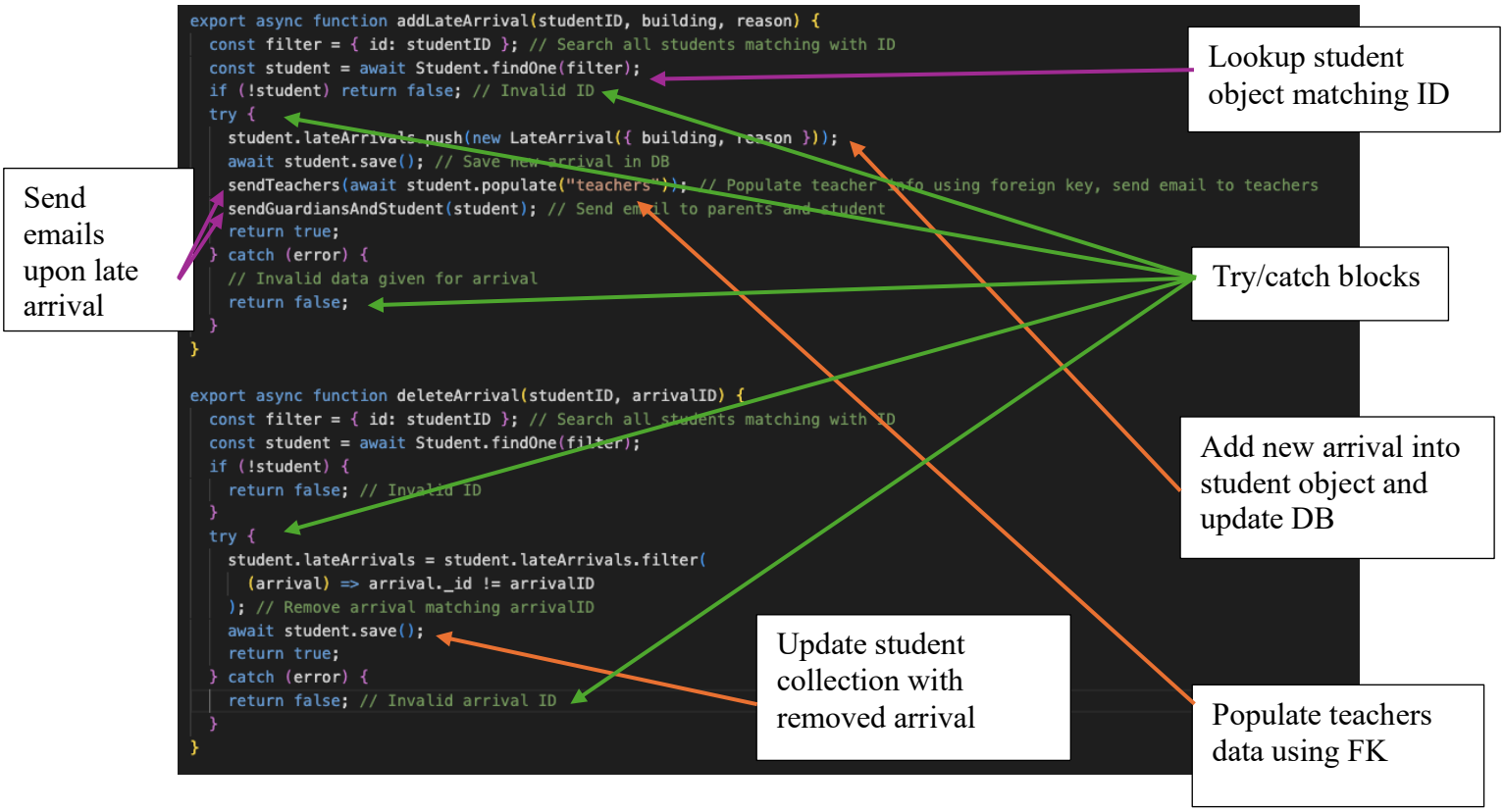
Students indexed by name and ID

Referencing

Embedding

Structure to the database was provided using *Schemas* and *Models*. The teacher entity is linked to the Student entity via *referencing* (a traditional primary/foreign key setup in a relational database to prevent repetition of data (denormalization). Data was directly *embedded* into the Student document for the LateArrival and Guardian entities rather than being referenced, since there is no repetition for arrivals and unlikely to be too much repetition for the same guardian. *Data validation* is performed on the section, grade, guardians and teachers attributes of the student to ensure data consistency.

crud.js – CRUD operations into database (CRUD logic layer)



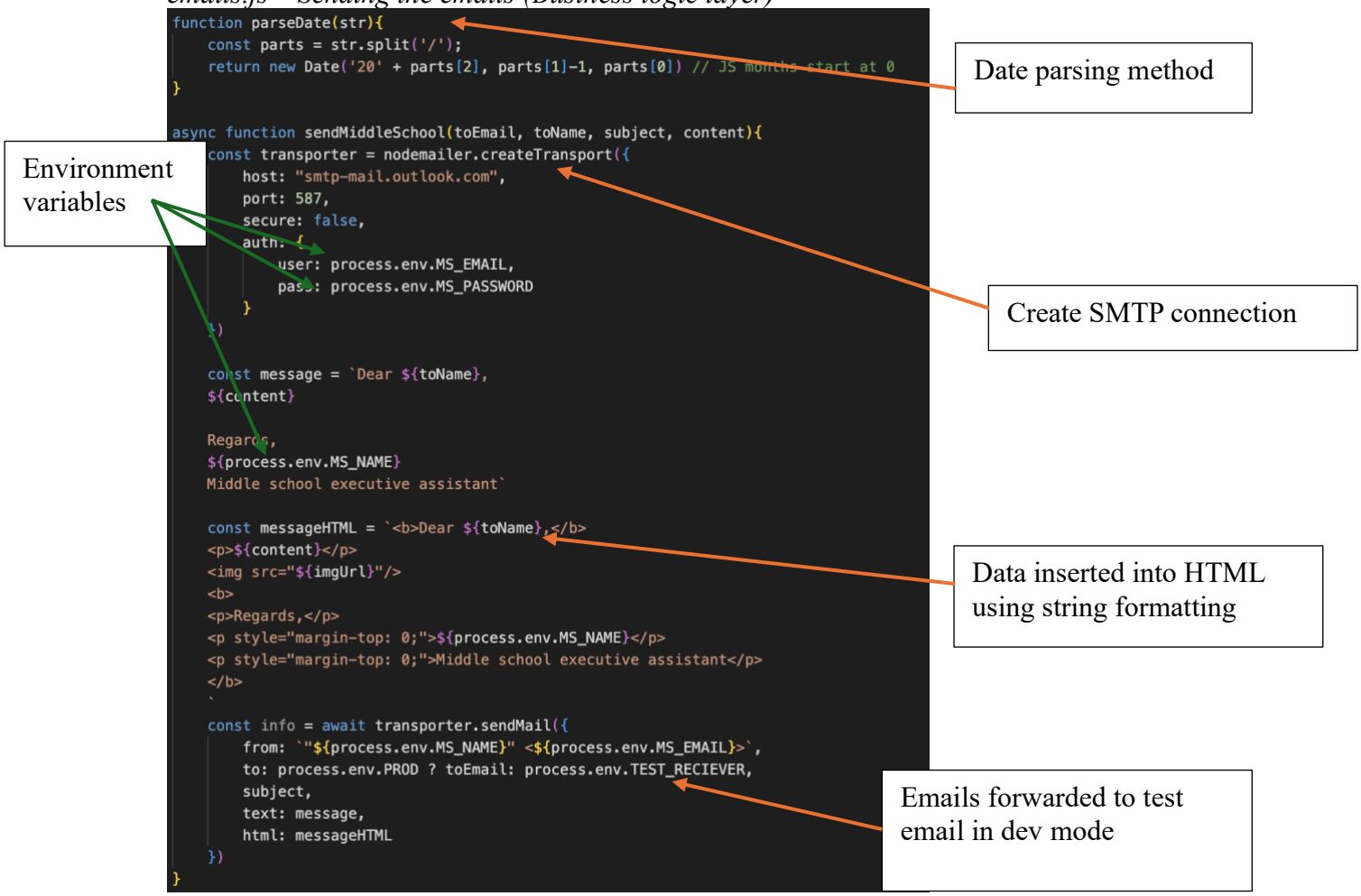

```
export async function addLateArrival(studentID, building, reason) {
  const filter = { id: studentID }; // Search all students matching with ID
  const student = await Student.findOne(filter);
  if (!student) return false; // Invalid ID
  try {
    student.lateArrivals.push(new LateArrival({ building, reason }));
    await student.save(); // Save new arrival in DB
    sendTeachers(await student.populate("teachers")); // Populate teacher info using foreign key, send email to teachers
    sendGuardiansAndStudent(student); // Send email to parents and student
    return true;
  } catch (error) {
    // Invalid data given for arrival
    return false;
  }
}

export async function deleteArrival(studentID, arrivalID) {
  const filter = { id: studentID }; // Search all students matching with ID
  const student = await Student.findOne(filter);
  if (!student) {
    return false; // Invalid ID
  }
  try {
    student.lateArrivals = student.lateArrivals.filter(
      (arrival) => arrival._id != arrivalID
    ); // Remove arrival matching arrivalID
    await student.save();
    return true;
  } catch (error) {
    return false; // Invalid arrival ID
  }
}
```

The *addLateArrival* and *deleteArrival* functions respectively contain the logic for fetching the student matching an ID (findOne), adding/deleting a new arrival record by instantiating new LateArrival object or filtering existing objects and updating the DB (save) using appropriate *mongoose CRUD functions.* Error handling done using *try/catch* blocks to detect errors in fetching or updating data due to invalid data/parameters, after which a "false" is returned to signify an unsuccessful transaction.


```
export async function queryStudentsStartingWithName(name) {
  const sanitizedName = escapeStringRegexp(name);
  const regexPattern = new RegExp(`^${sanitizedName}`, "i"); // starts with name ignoring case
  const results = await Student.find({ name: { $regex: regexPattern } }).select(
    "name id"
  );
  return results;
}

export async function queryStudentByID(id) {
  const student = await Student.findOne({ id });
  return student;
}
mongoose.connect(`mongodb://127.0.0.1:${process.env.MONGOPORT}/csia`);
console.log("connected to mongodb");
```

User input is *sanitized* using *escape-string-regex* library to remove special characters which could cause buggy or insecure behavior. *Regular expression* query of "^…." was used with the "I" option to find all names starting with the input ignoring case differences.

emails.js – Sending the emails (Business logic layer)



The custom function *parseDate* was written to parse dd/mm/yy strings for the semester dates into Date objects. The *external library* Nodemail was used to create an SMTP connection to send emails via Outlook. *HTML and CSS* was used to make the emails aesthetically pleasing and *Javscript string formatting* using ${variable name}$ was used to insert dynamic data into the email.

```
export function numTimesLate(student) {
  let cutoffDate;
  if (isSenior(student)) {
    const now = new Date();
    if (parseDate(constants.semesters[1]) < now) {
      // currently 2nd semester going on
      cutoffDate = parseDate(constants.semesters[1]);
    } else {
      // currently 1st sem
      cutoffDate = parseDate(constants.semesters[0]);
    }
  } else {
    const now = new Date();
    now.setDate(1); // first day of current month
    cutoffDate = now;
  }
  const recentArrivals = student.lateArrivals.filter(
    (arrival) => arrival.arrivalTime > cutoffDate
  );
  return recentArrivals.length;
}
```

Cutoff date set to start of semester

Cutoff date set to start of month

Conditional statements were used to set cutoff dates for senior and middle school students, the custom *parseDate* method was called to use Date objects to compare the dates of arrivals to the cutoff date to determine number of times late in current semester/month.

```
export async function sendTeachers(student) {
  const arrival = student.lateArrivals.slice(-1)[0];
  student.teachers.forEach((teacher) => {
    const content = `Kindly excuse your student ${
      student.name
    } for coming late to class today. They entered ${
      arrival.building
    } at ${formatAMPM(arrival.arrivalTime)} and were late because of ${
      arrival.reason
    }.`;
    const subject = `Late arrival of ${student.name}`;
    isSenior(student)
      ? sendSeniorSchool(teacher.email, teacher.name, subject, content)
      : sendMiddleSchool(teacher.email, teacher.name, subject, content);
  });
}
```

Latest arrival object for the student is obtained

Javascript ternary operator to handle if/else logic elegantly

importData.js – Import student and teacher data from JSON file into database


```
import Student from "./model/Student.js";
import Teacher from "./model/Teacher.js";
import fs from "fs/promises";
import mongoose from "mongoose";
import "dotenv/config";

mongoose.connect(`mongodb://127.0.0.1:${process.env.MONGOPORT}/csia`);
console.log("connected!");

const data = await fs.readFile("./data.json");
const sections = JSON.parse(data); // Parse data into JS object
for (let i = 0; i < sections.length; i++) {
  // Loop over sections
  const section = sections[i];
  const teacher1 = await Teacher.create({
    name: section.teacher1,
    email: section.teacher1email,
  }); // Add form teacher from section to DB
  const teacher2 = await Teacher.create({
    name: section.teacher2,
    email: section.teacher2email,
  }); // Add co-form teacher from section to DB
  const students = section.students;
  for (let j = 0; j < students.length; j++) {
    // Loop over students in section
    const student = students[j];
    const { name, guardians, id, email } = student;
    await Student.create({
      // Add student object to DB
      name,
      guardians,
      id,
      email,
      teachers: [teacher1._id, teacher2._id], // Reference both teachers
      section: section.section,
      grade: section.grade,
    });
  }
}
console.log("Done.");
```

Asynchronous read IO operations were conducted to read data from the JSON file which was then *parsed into a Javsacript object.* These new teacher and student objects were inserted into the database using the *mongoose Create method*. A *nested loop* was used to iterate over each section and then each student in each section. Teacher ID was added as a field to student objects being created following the referencing relationship between the two entities.

index.js – API routes (API layer)

```
import {
  queryStudentByID,
  queryStudentsStartingWithName,
  addLateArrival,
  deleteArrival,
} from "./crud.js";
import express from "express";
import { rateLimit } from "express-rate-limit";
import cors from "cors";
import "dotenv/config";
import "./model/Teacher.js";
import { numTimesLate } from "./emails.js";

const app = express();

// Use static middleware before auth or rate limiting
app.use("/static", express.static("static"));

const limiter = rateLimit({
  windowMs: 5 * 60 * 1000, // 15 minutes
  max: 50,
});

app.use(cors());

app.use((req, res, next) => {
  if (req.path !== "/queryName") {
    // Excluded due to high volume
    limiter(req, res, next);
  } else {
    next();
  }
});

app.use((req, res, next) => {
  if (req.path !== "/queryName") {
    // Excluded due to no senstitive data
    const provided = req.headers["passkey"];
    const required = process.env.PASSKEY;
    if (!provided || provided !== required) {
      return res.status(401).json({ error: "Unauthorized" });
    }
    next();
  } else {
    next();
  }
});

app.use(express.json());
```

CRUD operation modules and external libraries imported

Static middleware

CORS middleware

Query name endpoint excluded from rate limiting

HTTP passkey header accession

401 response

Parse JSON body to JS object

Middlewares (pre-processing/intercepting of requests before they are handled) were created for the API using the *app.use* method. Initially, requests for static files are intercepted. Then, CORS protocols are applied for all dynamic requests to enable the webapp to access the API. Then, rate limiting middleware using the *express-rate-limit* library is applied before authentication to prevent brute force attacks. Authentication middleware is applied which uses *HTTP header* to access the passkey and returns *401 Unauthorized response code.*

```
app.get("/queryName", async (req, res) => {
  if (!req.query.name) {
    return res.status(400).json("Missing name");
  }
  return res
    .status(200)
    .json(await queryStudentsStartingWithName(req.query.name));
});

app.get("/queryID", async (req, res) => {
  if (!req.query.studentID) {
    return res.status(400).json("Missing ID");
  }
  const student = await queryStudentByID(req.query.studentID);
  if (!student) {
    return res.status(400).json("Invalid ID");
  }
  const studentObject = student.toObject();
  studentObject.wasLateThrice = numTimesLate(student) >= 3;
  return res.status(200).json(studentObject);
});

app.post("/arrival", async (req, res) => {
  const { studentID, building, reason } = req.body;
  const successful = await addLateArrival(studentID, building, reason);
  if (successful) return res.status(200).json("Successful");
  return res.status(400).json("Invalid body");
});

app.delete("/arrival", async (req, res) => {
  const { studentID, arrivalID } = req.query;
  const successful = await deleteArrival(studentID, arrivalID);
  if (successful) return res.status(200).json("Successful");
  return res.status(400).json("Invalid body");
});
```

HTTP GET method

400 response for invalid requests

200 response for successful requests

Document converted into JS object to be able to add additional attributes to it

HTTP POST method

async/await syntax

HTTP DELETE method

Express routes were used to handle the API requests and return a response, with appropriate *HTTP methods* (*GET* for querying user from ID/name, *POST* for adding arrival, *DELETE* for deleting arrival). *Asynchronous programming* via the use of *async/await* syntax was used to ensure the server can handle multiple requests preventing performance issues by not blocking the main thread. Appropriate *HTTP response codes* were returned (400 when crud layer responded with an invalid transaction, indicating invalid body/query, 200 when request was successfully processed).

backup.sh – Database backup setup

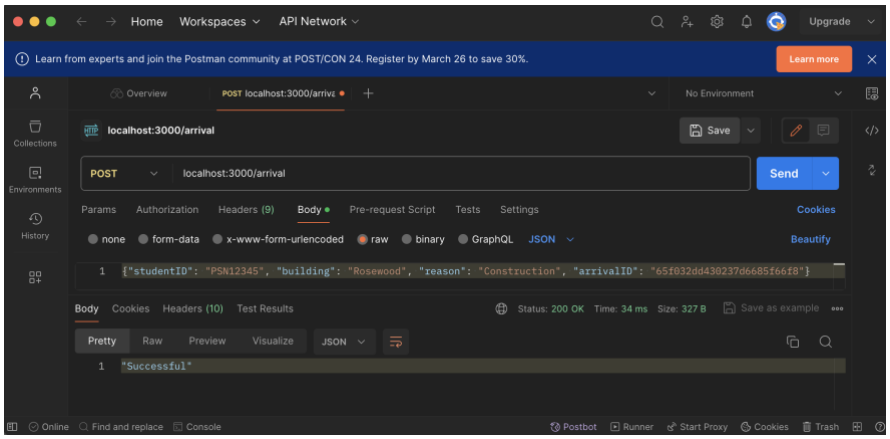
```
#!/bin/bash

PORT="XXXX"
DBNAME="XXXX"
TIMESTAMP=$(date '+%Y%m%d%H%M%S')
mkdir -p "backup/$TIMESTAMP"
BACKUP_SCRIPT_PATH="/usr/bin/mongodump --host 127.0.0.1 --port $PORT --db $DBNAME --out backup/$TIMESTAMP"
CRON_SCHEDULE="0 0 * * *"
(crontab -l ; echo "$CRON_SCHEDULE $BACKUP_SCRIPT_PATH") | sort - | uniq - | crontab -
```

Make a folder with the current date to store backup

Cron schedule to run every day at midnight

Run mongodump to dump database to specified folder

Use of Postman software



All the API endpoints were tested using this software by ensuring that the ensuring responses were correct in normal and abnormal requests.

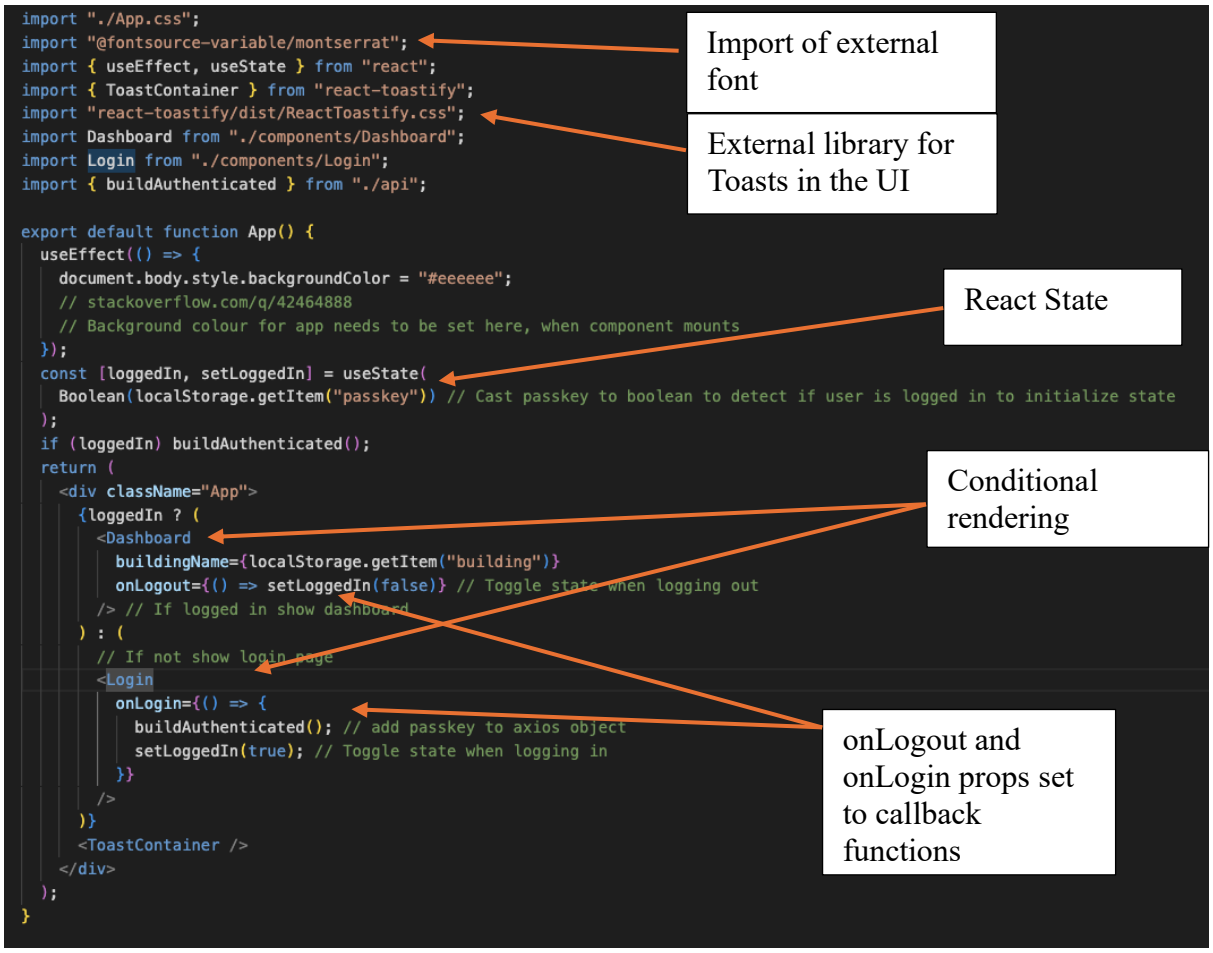
Use of mongosh CLI tool to view and manipulate database

The mongosh CLI tool was used to view the database during development for debugging purposes.

Frontend (frontend layer)

Use of chrome developer tools to debug the web application
Styles tab in Devtools used to debug flexbox and box model, network tab to debug API requests sent, console tab to check for javascript errors and application tab to view and edit localStorage.

App.js – Root react component attached to the HTML body




```
import "./App.css";
import "@fontsource-variable/montserrat";
import { useEffect, useState } from "react";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import Dashboard from "./components/Dashboard";
import Login from "./components/Login";
import { buildAuthenticated } from "./api";

export default function App() {
  useEffect(() => {
    document.body.style.backgroundColor = "#eeeeee";
    // stackoverflow.com/q/42464888
    // Background colour for app needs to be set here, when component mounts
  });
  const [loggedIn, setLoggedIn] = useState(
    Boolean(localStorage.getItem("passkey")) // Cast passkey to boolean to detect if user is logged in to initialize state
  );
  if (loggedIn) buildAuthenticated();
  return (
    <div className="App">
      {loggedIn ? (
        <Dashboard
          buildingName={localStorage.getItem("building")}
          onLogout={() => setLoggedIn(false)} // Toggle state when logging out
        /> // If logged in show dashboard
      ) : (
        // If not show login page
        <Login
          onLogin={() => {
            buildAuthenticated(); // add passkey to axios object
            setLoggedIn(true); // Toggle state when logging in
          }}
        />
      )}
      <ToastContainer />
    </div>
  );
}
```

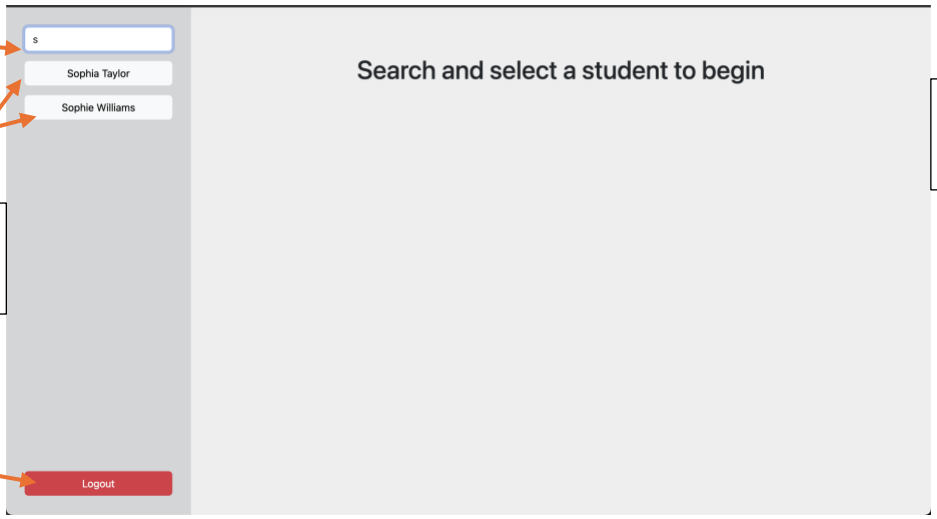
This root component keeps track of whether the user is logged in or not using the Boolean *loggedIn React state.* The value of this state is used with *conditional rendering* syntax to render the Dashboard if the user is logged in and the Login page if not. This state is updated when the child elements (Dashboard when logout clicked, Login when building clicked) bubble the state up using *callback functions* passed to their *props.* Updating the state causes the App to re-render, updating the UI after the user logs in/out.

Login.jsx – Login page


```
import BuildingSelector from "./BuildingSelector";
import "../styles/Login.css";

export default function Login({ onLogin }) {
  return (
    <div className="picker">
      <h1 className="heading">Login</h1>
      <div className="buildings">
        <BuildingSelector onLogin={() => onLogin()} buildingName="rosewood" />
        <BuildingSelector onLogin={() => onLogin()} buildingName="chestnut" />
      </div>
    </div>
  );
}
```

The use of a separate component for the BuildingSelector (*modularization)* instead of repeating code enhances extensibility (another building could be easily added in the future)

Dashboard.jsx: Component for the main dashboard of the application




```jsx
export default function Dashboard({ buildingName, onLogout }) {
  const [students, setStudents] = useState([]);
  const [activeStudent, setActiveStudent] = useState(null);
  return (
    <div className="dashboard">
      <div className="sidebar">
        <div className="searchbar">
          <Form.Control
            onChange={async (event) => {
              const value = event.target.value; // Get the data entered in search bar
              if (value === "") {
                return setStudents([]);
              }
              const result = await authenticated.get("queryName", {
                params: { name: value }, // Make API request to get students matching name
              });
              setStudents(result.data); // Update the state with response data
            }}
            type="text"
            placeholder="Search name..."
          />
        </div>
        {students.map((student) => (
          <div key={student.id} className="student-name-container">
            <Button
              onClick={() => {
                // Update state when students elected
                setActiveStudent(student);
              }}
              className="student-button"
              variant="light"
            >
              {student.name}
            </Button>
          </div>
        ))}
```

Called when text changed

Make an API query

Update students state

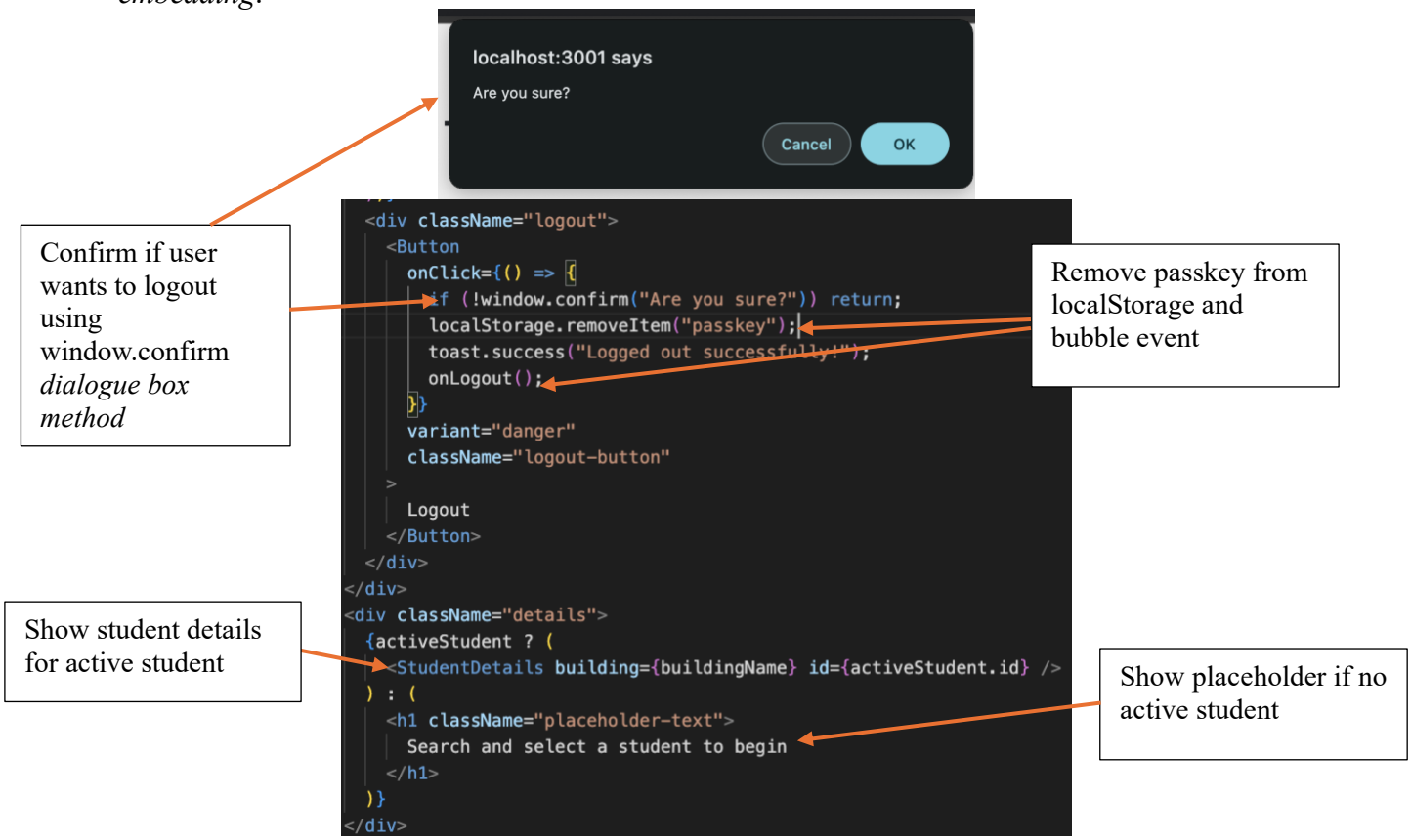
Bootstrap Form.Control text elements

Map students array state to elements

Set active student when button with student name clicked

Embedding JS expression into JSX

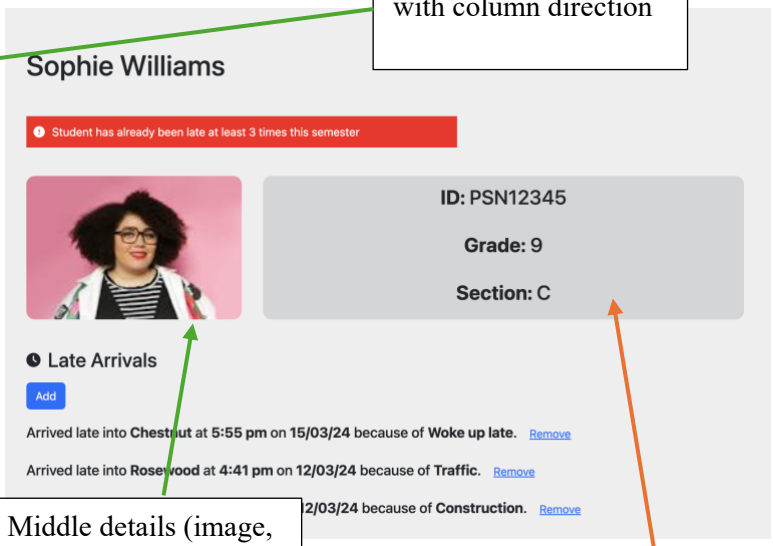
The Dashboard component renders the Dashboard screen of the application and maintains a *React state* of the list of students matching the search and the currently selected student. When the *Bootstrap text input element* is changed, an *API query* is made to find matching students with the name and update the students state. The *onClick* callback is used to detect button clicks for each student and update the *activeStudent* state. The JS *.map* function is used to map each student object to a Button with the student name filled using *Javascript embedding*.



The *activeStudent* state is used to *conditionally render* either the student's details by passing the student ID and building as a *prop* to the StudentDetails component or show a placeholder text if there is no active student.

Student details: Component displaying student info, late arrivals and buttons to add/delete records, StudentDetails.css: Styling for student details

```css
.student-details {
  display: flex;
  flex-direction: column;
}

.student-detail {
  margin-top: 40px;
}

.middle-details {
  display: flex;
}

.image-container {
  flex: 0 0 30%;
}

.student-image {
  width: 100%;
  max-height: 100%;
  border-radius: 12px;
}
```



Root container flexbox with column direction

Middle details (image, ID, grade, section) container

Right details container (inside the middle details)

Image and container with border radius

```css
.right-details-container {
  flex-grow: 1;
  justify-content: space-around;
  align-items: center;
  display: flex;
  flex-direction: column;
  background-color: #d4d5d6;
  margin-left: 30px;
  border-radius: 12px;
}

.icon-container {
  display: flex;
  align-items: center;
}

.icon {
  height: 20px;
  margin-right: 10px;
}

.add-arrival {
  margin-top: 15px;
  margin-bottom: 15px;
}
```

Container with icon and “Late Arrivals” text

Spacing next to icon using margin-right

Spacing around the “add-arrival” element

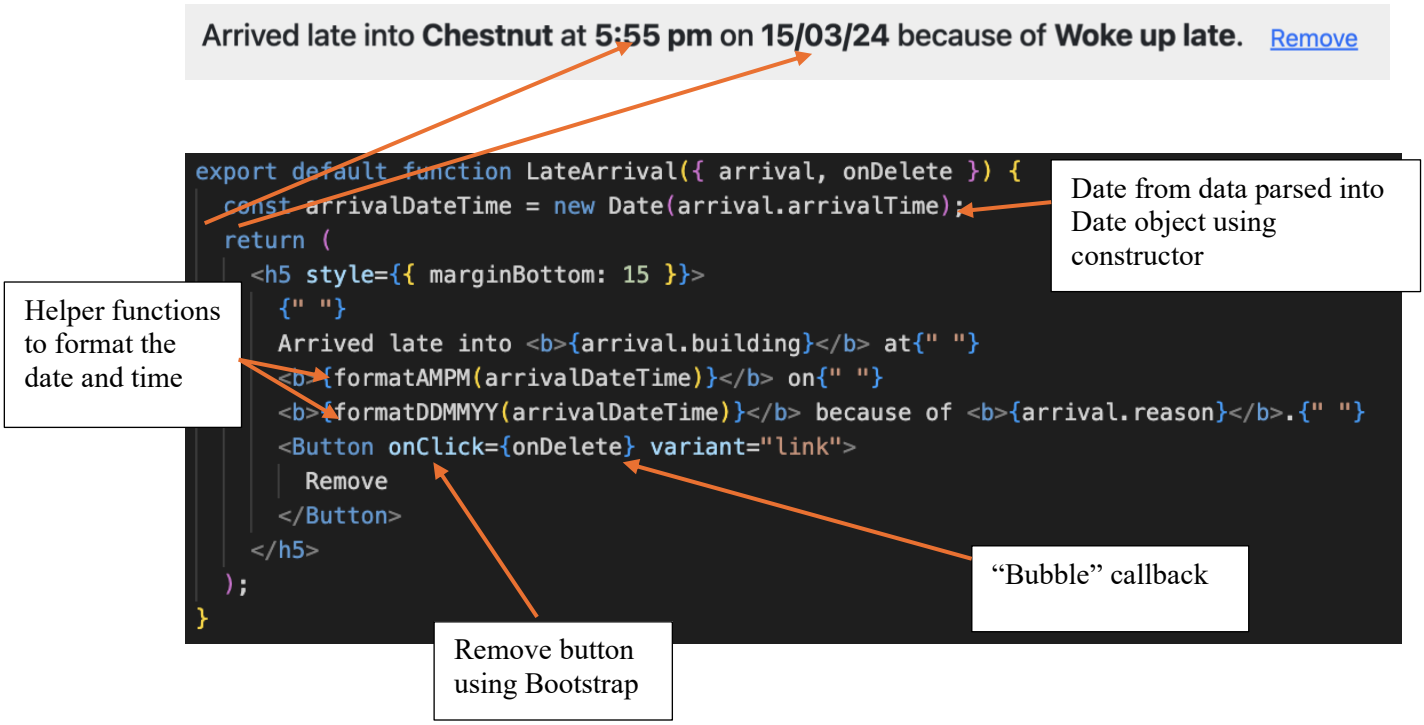
Flexbox was used to structure the elements in the UI. The *flex-direction* set to *column* arranges elements vertically in the root element. Each sub container is itself a flex container, with the middle details container arranged in a row. *Justify-content* and *align-items* flex attributes were used to ensure that the ID, Grade and Section were horizontally centered and equally spaced. Margins were added using the margin attribute to create enough spacing.

StudentDetails.jsx: Lines 1-28 and 116-123


```
import "../styles/StudentDetails.css";
import { useEffect, useState } from "react";
import Warning from "./Warning";
import { authenticated, getImage } from "../api";
import Button from "react-bootstrap/Button";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faClock } from "@fortawesome/free-solid-svg-icons";
import LateArrival from "./LateArrival";
import { toast } from "react-toastify";

export default function StudentDetails({ id, building }) {
  const [student, setStudent] = useState(null);
  const [recordsChanged, setRecordsChanged] = useState(false); // to update UI after records chan
  useEffect(() => {
    async function getData() {
      const response = await authenticated.get("/queryID", {
        // Fetch data from API
        params: { studentID: id },
      });
      setStudent(response.data); // Update state
    }
    getData();
  }, [id, recordsChanged]); // Dependency on recordsChanged and ID to update details when they are changed
  let reversedArrivals = [];
  if (student) reversedArrivals = [...student.lateArrivals].reverse();
  return (
    <div>
      {student ? ( // When student details are fetched, show them
        <div className="student-details">…
        </div>
      ) : (
        <p>Loading...</p> // Otherwise display loading
      )}
    </div>
  );
}
```

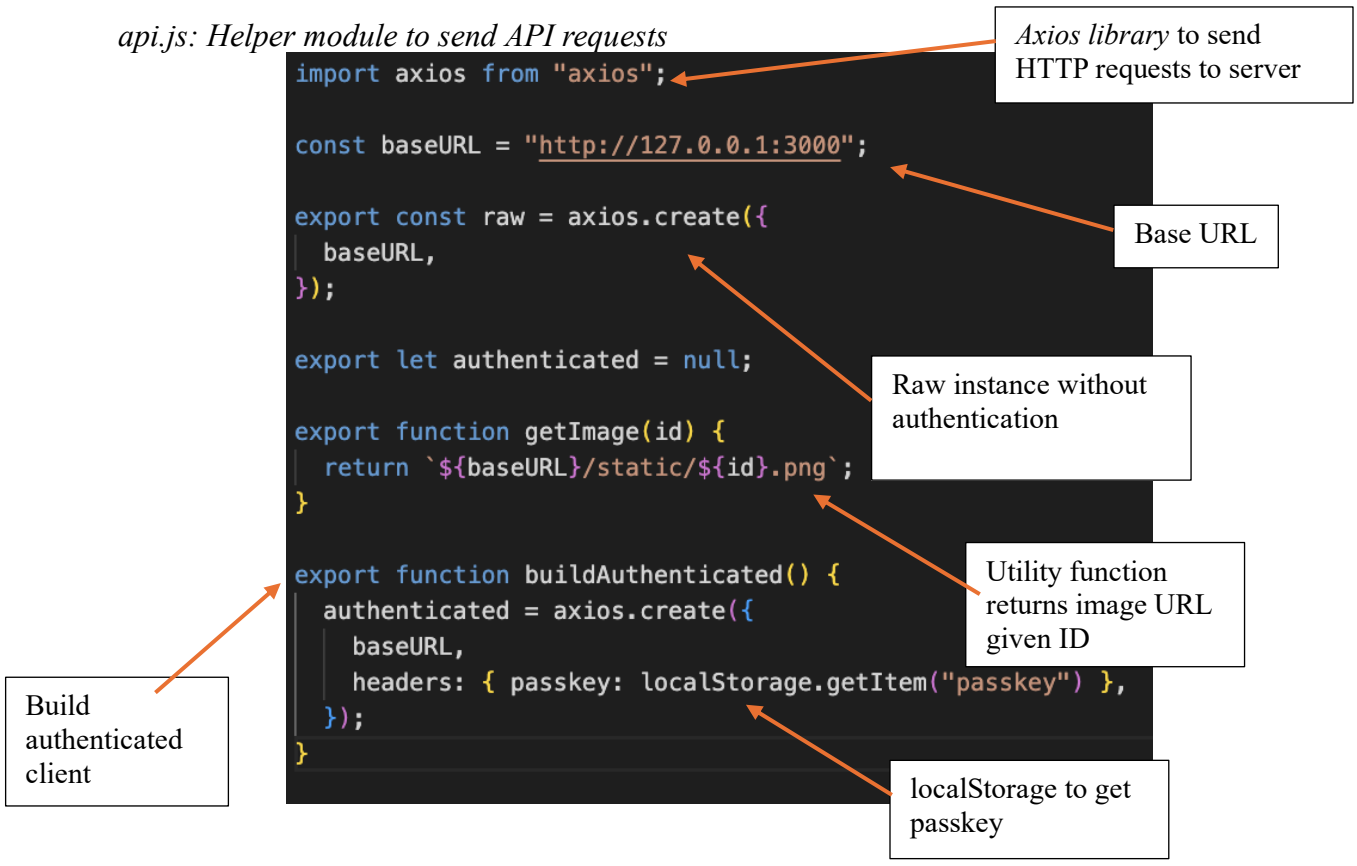
The *useEffect hook* is used to automatically *fetch student data from the API* when the component is mounted. The *useEffect dependencies* of *id* and *recordsChanged* ensure that when the ID is changed (a different student is selected) or arrival records are updated (added/deleted) the component is re-rendered and UI is updated. The *Javascript spread (…) operator* and the *reverse() method* were used to reverse the arrivals array, so that the latest arrivals are shown at the top.

LateArrival.jsx: Component to display the late arrival record information



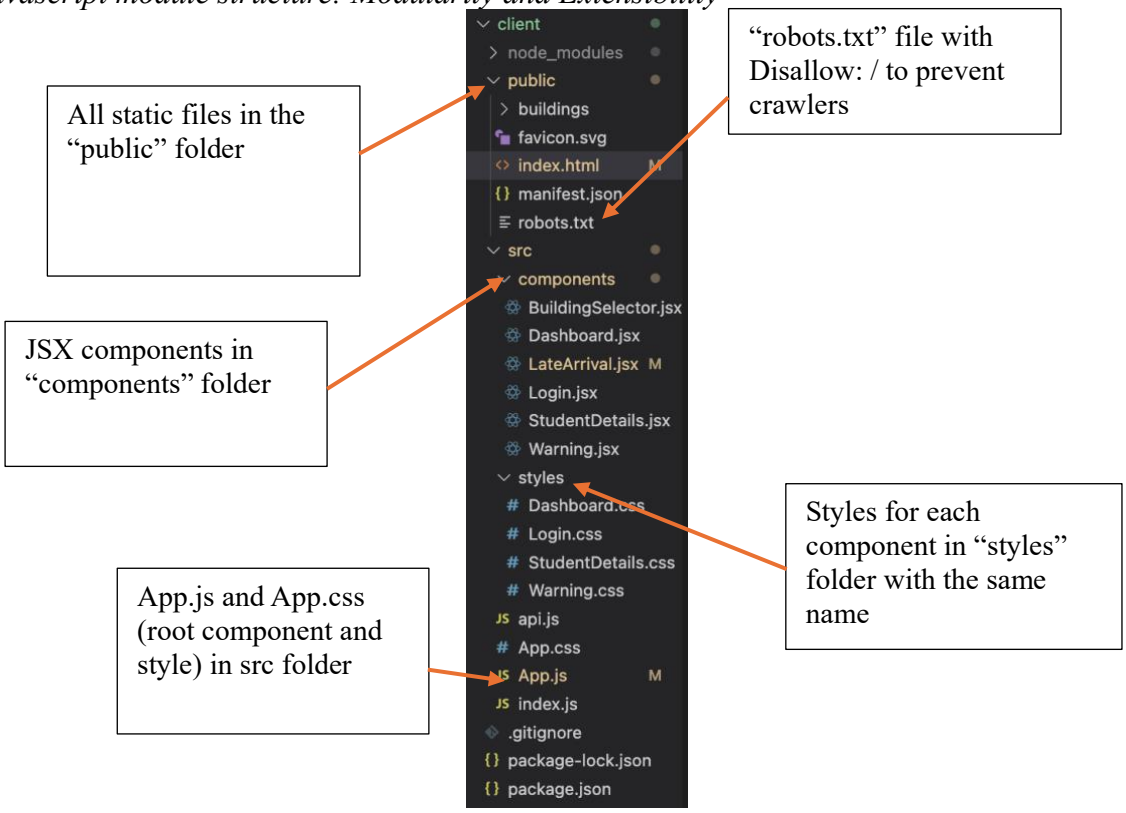
The *LateArrival* component displays each late arrival and the Button to remove it, which when clicked calls the *onDelete* function passed as a prop (a technique used as "bubbling") to tell the parent component (*StudentDetails*) that the record has been deleted, so that the the *DELETE API request* can be sent and the *recordsChanged* state can be updated, updating the UI.

api.js: Helper module to send API requests



Api.js is a separate module that performs the task of maintaining authentication and sending API requests, showcasing *modularization* of code. The authenticated client is built using the passkey stored in *localStorage*, a non-volatile key value store to ensure that the login remains even after the tab is closed.

Javascript module structure: Modularity and Extensibility



The organized file structure and descriptive module names make the code readable and extensible, allowing future maintainers to easily understand and edit the codebase. Standard conventions such as having the style files with the same name as the module files and having the root component directly in the "src" folder were followed. Standard conventions for variable and function names were followed throughout, such as making them camel case and naming them appropriately.

Bibliography

Aphiwat Chuangchoem. “Black Analog Alarm Clock at 7:01.” *Pexels*, www.pexels.com/photo/black-analog-alarm-clock-at-7-01-359989. Accessed 18 May 2023.

“Array.Prototype.forEach() - JavaScript | MDN.” *MDN Web Docs*, 17 Apr. 2023, developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach. Accessed 14 June 2023.

The Axios Instance | Axios Docs. axios-http.com/docs/instance. Accessed 2 Aug. 2023.

Bezkoder. “Node.js, Express and MongoDb: Build a CRUD Rest Api Example - BezKoder.” *BezKoder*, 1 Dec. 2022, www.bezkoder.com/node-express-mongodb-crud-rest-api. Accessed 12 Jun. 2023.

Chestnut Tree. www.britannica.com/plant/chestnut. Accessed 1 Sept. 2023.

Color Hex Color Codes. www.color-hex.com. Accessed 2 Jan. 2023.

“Conditional Rendering – React.” *React*, legacy.reactjs.org/docs/conditional-rendering.html. Accessed 4 Aug. 2023.

“Conditional (Ternary) Operator - JavaScript | MDN.” *MDN Web Docs*, 7 Sept. 2023, developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator. Accessed 26 Aug. 2023.

Coyier, Chris. “A Complete Guide to Flexbox | CSS-Tricks.” *CSS-Tricks*, 9 Dec. 2022, css-tricks.com/snippets/css/a-guide-to-flexbox. Accessed 26 Aug. 2023.

“Cross-Origin Resource Sharing (CORS) - HTTP | MDN.” *MDN Web Docs*, 19 Dec. 2023, developer.mozilla.org/en-US/docs/Web/HTTP/CORS. Accessed 20 Jun. 2023.

"CSS: Cascading Style Sheets | MDN." *MDN Web Docs*, 5 Mar. 2024, developer.mozilla.org/en-US/docs/Web/CSS. Accessed 21 Aug. 2023.

Express Cors Middleware. expressjs.com/en/resources/middleware/cors.html. Accessed 29 May 2023.

"Font Awesome." *Font Awesome*, fontawesome.com. Accessed 24 Aug. 2023.

Fontsource. "Montserrat | Fontsource." *Fontsource*, fontsource.org/fonts/montserrat/install. Accessed 24 Aug. 2023.

Hall, Jesse. *Getting Started With MongoDB and Mongoose | MongoDB*. 19 May 2022, www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose. Accessed 4 Apr. 2023.

"How Do I Change the Background Color of the Body?" *Stack Overflow*, stackoverflow.com/q/42464888. Accessed 10 Sept. 2023.

"How Do You Display JavaScript Datetime in 12 Hour AM/PM Format?" *Stack Overflow*, stackoverflow.com/a/8888498. Accessed 10 May 2023.

"How to Get Current Formatted Date Dd/Mm/Yyyy in Javascript and Append It to an Input." *Stack Overflow*, stackoverflow.com/a/12409344. Accessed 10 May 2023.

"How to Return an Empty Jsx Element From the Render Function in React?" *Stack Overflow*, stackoverflow.com/a/55545704. Accessed 29 July 2023.

"HTTP Response Status Codes - HTTP | MDN." *MDN Web Docs*, 3 Nov. 2023, developer.mozilla.org/en-US/docs/Web/HTTP/Status. Accessed 1 June 2023.

Introduction | React Bootstrap. react-bootstrap.netlify.app/docs/getting-started/introduction. Accessed 21 Aug. 2023.

Irmak, Mehmet R. "FETCHING DATA WITH AXIOS IN REACT - Mehmet R. IRMAK - Medium." *Medium*, 8 May 2023, medium.com/@mehmet.r.river/fetching-data-with-axios-in-react-42f79490a97b. Accessed 7 Aug. 2023.

Lepilkina, Diana. “Embedding Images in HTML Email: Have the Rules Changed? | Mailtrap Blog.” *Mailtrap*, 11 Mar. 2024, mailtrap.io/blog/embedding-images-in-html-email-have-the-rules-changed. Accessed 18 May 2023.

Lukehoban. “GitHub - Lukehoban/Es6features: Overview of ECMAScript 6 Features.” *GitHub*, github.com/lukehoban/es6features. Accessed 4 Apr. 2023.

“MissingSchemaError: Schema Hasn’t Been Registered for Model ‘User.’” *Stack Overflow*, stackoverflow.com/a/27540516. Accessed 25 Mar. 2023.

“MongoDB Relationships: Embed or Reference?” *Stack Overflow*, stackoverflow.com/q/5373198. Accessed 5 Mar. 2023.

Mongoose v8.2.1: Query Population. mongoosejs.com/docs/populate.html#setting-populated-fields. Accessed 15 Apr. 2023.

Mongoose v8.2.3: Schemas. mongoosejs.com/docs/guide.html. Accessed 25 Mar. 2023.

“No Restricted Globals.” *Stack Overflow*, stackoverflow.com/a/44998181. Accessed 19 Aug. 2023.

“Overview - Express-rate-limit.” *Express-rate-limit*, express-rate-limit.mintlify.app/overview. Accessed 8 June 2023.

PaulHalliday. “How to Use Axios With React.” *DigitalOcean*, 2 Dec. 2021, www.digitalocean.com/community/tutorials/react-axios-react#step-5-using-a-base-instance-in-axios. Accessed 7 Aug. 2023.

POP, IMAP, and SMTP Settings for Outlook.com - Microsoft Support. support.microsoft.com/en-au/office/pop-imap-and-smtp-settings-for-outlook-com-d088b986-291d-42b8-9564-9c414e2aa040. Accessed 4 May 2023.

ProgrammingKnowledge. “How to Install MongoDB on Mac | Install MongoDB on macOS (2024).” *YouTube*, 12 Jan. 2024, www.youtube.com/watch?v=8gUQL2zlpvI. Accessed 15 Mar. 2023.

"Promise - JavaScript | MDN." *MDN Web Docs*, 29 Nov. 2023, developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. Accessed 9 Apr. 2023.

Quick, James. "How to Format Code With Prettier in Visual Studio Code." *DigitalOcean*, 29 Nov. 2021, www.digitalocean.com/community/tutorials/how-to-format-code-with-prettier-in-visual-studio-code. Accessed 12 Mar. 2023.

Quick Start – React. react.dev/learn. Accessed 12 July 2023.

React Functional Components: In-Depth Guide. 5 Sept. 2023, www.knowledgehut.com/blog/web-development/react-functional-components. Accessed 13 July 2023.

React-toastify | React-Toastify. fkhadra.github.io/react-toastify/introduction. Accessed 15 July 2023.

Reinman, Andris. *Nodemailer :: Nodemailer*. nodemailer.com. Accessed 3 May 2023.

Responsive Web Design Viewport. www.w3schools.com/css/css_rwd_viewport.asp. Accessed 20 Nov. 2023.

Rosewood Tree. housing.com/news/wp-content/uploads/2023/01/Teak-tree-shutterstock_1868125927-1200x700-compressed.jpg. Accessed 1 Sept. 2023.

Serving Static Files in Express. expressjs.com/en/starter/static-files.html. Accessed 17 June 2023.

Smallcombe, Mark. "SQL Vs NoSQL: 5 Critical Differences." *Integrate.io*, 29 Nov. 2018, www.integrate.io/blog/the-sql-vs-nosql-difference. Accessed 27 Dec. 2022.

"Storing Credentials in Local Storage." *Stack Overflow*, stackoverflow.com/q/7859972. Accessed 15 July 2023.

Today, JavaScript. *Understanding MVC Architecture: Beginner Tutorial With Node.js*. 4 Mar. 2023, blog.javascripttoday.com/blog/model-view-controller-architecture. Accessed 2 Jan. 2023.

useEffect – React. react.dev/reference/react/useEffect. Accessed 16 July 2023.

Using Express Middleware. expressjs.com/en/guide/using-middleware.html. Accessed 2 June 2023.

UXWing. “Time Late Icon PNG and SVG Vector Free Download.” *UXWing*, 1 Mar. 2022, uxwing.com/time-late-icon. Accessed 18 May 2023.

What Is MongoDB? — MongoDB Manual. www.mongodb.com/docs/manual. Accessed 15 Apr. 2023.

“Window: localStorage Property - Web APIs | MDN.” *MDN Web Docs*, 8 Apr. 2023, developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. Accessed 2 Aug. 2023.