Data Management Systems
Final Project Report

Group #7

Student Names & ID:

Lok Yung Chan, Carson(100926449)

Beamish, Samuel (100875987)

Krishna Mallick (100876443)

Zaid Nakhuda (100876416)

Introduction:

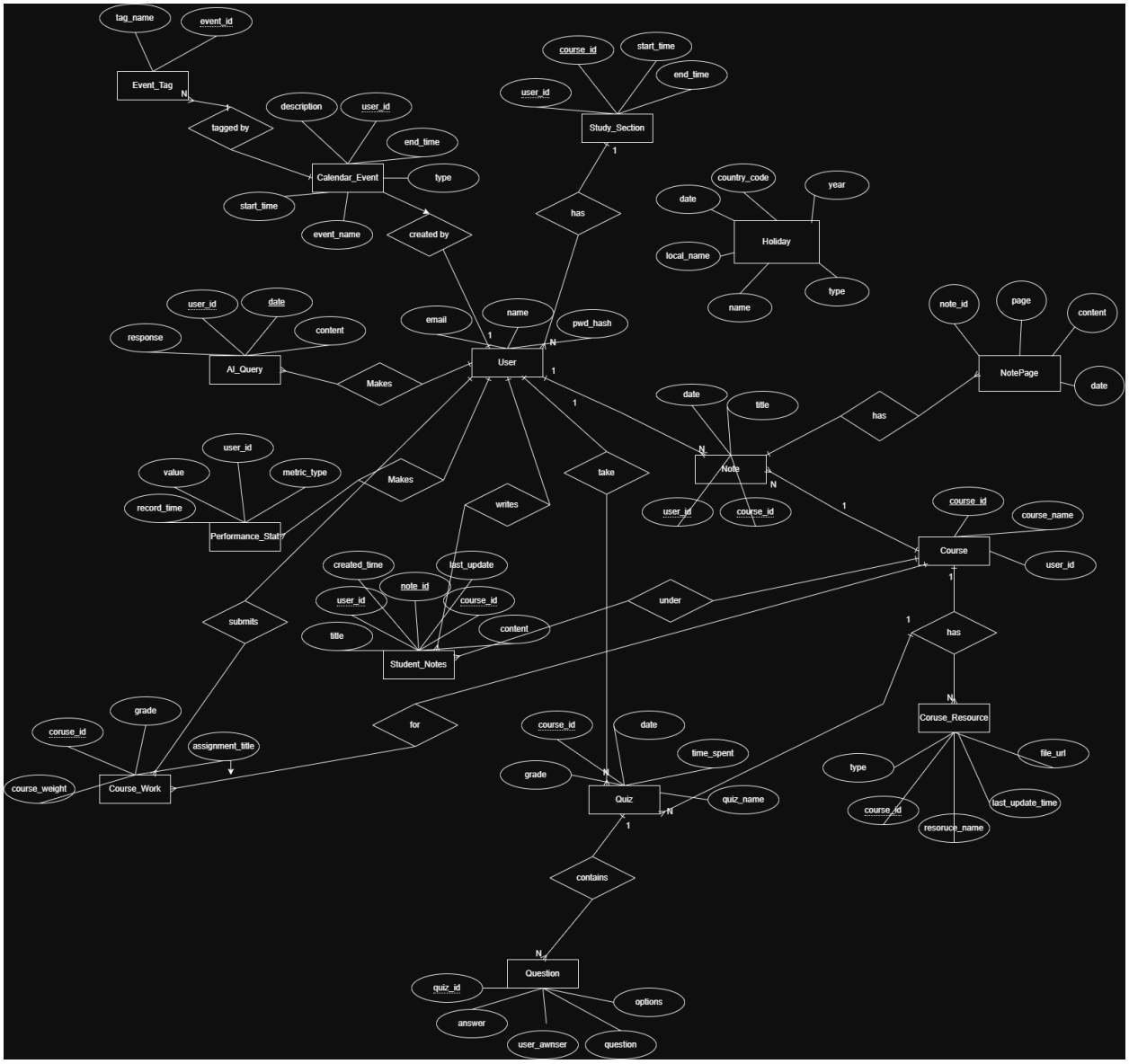
This report outlines the creation and evaluation of a full-stack study platform designed to help students stay organized with their courses, assignments, and weekly schedules. The app lets users securely sign up and log in, then manage their personalized course details like course codes, instructors, credits, and descriptions; all in one place. From the course dashboard, students can easily add, view, and update course information stored in a MongoDB database. A built-in calendar and task list offer a clear view of upcoming assignments and study plans. Helpful tools like search and filter options, dynamic forms for adding assignments and quizzes, and simple data visualizations make it easier for students to plan ahead and stay on top of their workload.

On the technical side, the platform is built using a React frontend and a Node.js/Express backend, working together seamlessly. User accounts are protected with secure password hashing using bcrypt, and all incoming data is thoroughly checked with Zod schemas before being saved to the database. The backend provides RESTful APIs for handling users, courses, assignments, and resources, which the frontend accesses using Axios. There's also a feature to export coursework and grades as an XML file, and a neat integration with the Nager public holiday API that automatically highlights Canadian holidays in the student's calendar. Altogether, this project demonstrates practical full-stack development skills, secure and structured data handling, and real-world API integration in building a tool that supports students in managing their academic life.

Contribution

Krishna - handeled api setups and calls, implemented ui/ux features	25%
Carson - most of backend, also helped with frontend ui features	25%
Sam - created skeleton code, added backend front end changes, controlled git hub configuration	25%
Zaid - created the important notes feature, assisted backend and front end	25%

Diagrams



Link:
https://drive.google.com/file/d/1ifaJw6Qhlq6XOLAgov3RO61ULu8NWaed/view?usp=sharing

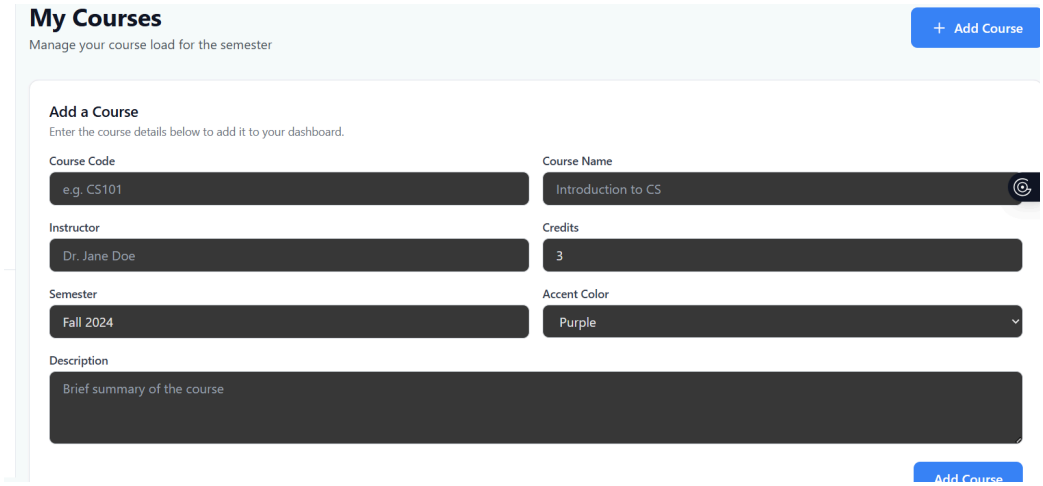
Link:
https://drive.google.com/file/d/1XnWwbQhC7HRPhJdAx4ISvwuWaPaswrj_/view?usp=sharing

Frontend Features:

Register/Login:
All users start navigating the application from the Landing and Authentication Page. When users attempt to make a new account, the form in the authentication page, sets the state to register, and passes the input value to `handleManualAuth()` from App.jsx. In the handler, a new user object is made with the input value and send to the backend:

```
if(authScreen === "signup"){
      let new_user;
      try{
        new_user={
          username: authScreen === "signup"
          ? authForm.name
          : authForm.email.split("@")[0],
          email: authForm.email,
          pswd_hash: authForm.password
        };
        //alert("made user obj");
      }catch(error){
        setAuthError("Unexpected error, please try again.");
        return;
      }
      //pass it to backend//
      try{
        const response = await
axios.post("http://localhost:3000/api/user",new_user);
        const savedUser = response.data;
```

Course Page:
The course page is where users can view all the courses they created. The top part of the page is a form to create new courses:



Upon clicking add course, the form will pass its value to `handleAddCourse` where data are packed to a course object, then push to database using axios.post()

```
const handleAddCourse = async(e) => {
    const newCourseObj ={
      user_id: user._id,
      course_code: courseFormData.code.trim(),
      course_name: courseFormData.course_name,
      instructor: courseFormData.instructor || "TBD",
      credit: Number(courseFormData.credit) || 0,
      description: courseFormData.description || "",
      color: courseFormData.color
    };
    try{
      const response = await
axios.post("http://localhost:3000/api/course", newCourseObj);
      const savedCourse = response.data;
      alert("successful");
```

Views:

Calendar view

Weekly Schedule

Nigerian Holidays in November 2025

National Youth Day

Time	Monday Nov 3	Tuesday Nov 4	Wednesday Nov 5	Thursday Nov 6	Friday Nov 7
8 AM					
9 AM	lecture Room 301 09:00 - 10:30		lecture Room 301 09:00 - 10:30		
10 AM					lab Physics Lab 2 10:00 - 12:00
11 AM		lecture Room 205 11:00 - 12:30			
12 PM					
1 PM	lecture Science Building A 13:00 - 14:30				
2 PM				tutorial Room 108 14:00 - 15:00	

-- view 1:

```
db.createView(
  "quiz_question_details",
  "quiz_question",
  [
    {
      $lookup: {
        from: "quiz",
        localField: "quiz_id",
        foreignField: "quiz_id",
        as: "quiz"
      }
    },
    { $unwind: "$quiz" },

    {
      $lookup: {
        from: "user",
        localField: "quiz.user_id",
        foreignField: "user_id",
        as: "user"
      }
    },
    { $unwind: "$user" },

    {
      $lookup: {
        from: "course",
        localField: "quiz.course_id",
        foreignField: "course_id",
        as: "course"
      }
    },
    { $unwind: "$course" },

    {
      $project: {
        question_id: 1,
        quiz_id: 1,
        user_id: "$quiz.user_id",
        user_name: "$user.name",
        course_id: "$quiz.course_id",
        course_name: "$course.course_name",
        question: 1,
        user_answer: 1,
        grade: 1,
        quiz_date: "$quiz.date"
      }
    }
  ]
)
```

-- view 2:

```
db.createView(
  "all_user_top_quizzes",
  "quiz_question",
  [
    {
      $group: {
        _id: { quiz_id: "$quiz_id", user_id: "$user_id" },
        quiz_grade: { $sum: "$grade" }
      }
    },

    {
      $group: {
        _id: "$_id.user_id",
        top_grade: { $max: "$quiz_grade" },
        quizzes: { $push: { quiz_id: "$_id.quiz_id", quiz_grade: "$quiz_grade" } }
      }
    },

    { $unwind: "$quizzes" },

    {
      $match: {
        $expr: { $eq: ["$quizzes.quiz_grade", "$top_grade"] }
      }
    },

    {
      $project: {
        _id: 0,
        user_id: "$_id",
        quiz_id: "$quizzes.quiz_id",
        quiz_grade: "$quizzes.quiz_grade"
      }
    }
  ]
)
```

-- view 3:

```
db.createView(
  "user_section_compare_average_time",
  "study_section",
  [
    {
      $lookup: {
        from: "user",
        localField: "user_id",
        foreignField: "user_id",
        as: "user"
      }
    },
    { $unwind: "$user" },

    {
      $addFields: {
        duration: {
          $divide: [
            { $subtract: ["$end_time", "$start_time"] },
            1000 * 60
          ]
        }
      }
    },

    {
      $group: {
        _id: "$user_id",
        name: { $first: "$user.name" },
        sections: {
          $push: {
            section_id: "$section_id",
            start_time: "$start_time",
            end_time: "$end_time",
            duration: "$duration"
          }
        },
        avg_duration: { $avg: "$duration" }
      }
    },

    { $unwind: "$sections" },

```

```
    {
      $project: {
        user_id: "$_id",
        name: 1,
        section_id: "$sections.section_id",
        start_time: "$sections.start_time",
        end_time: "$sections.end_time",
        comparison_to_avg: {
          $cond: [
            { $gt: ["$sections.duration", "$avg_duration"] },
            "Above Average",
            "Below Average"
          ]
        }
      }
    }
  ]
)
```

-- view 5:

```
db.createView(
  "v_failed_quizzes_same_course",
  "quiz",
  [
    {
      $group: {
        _id: { user_id: "$user_id", course_id: "$course_id" },
        quiz_count: { $sum: 1 }
      }
    },
    {
      $match: { quiz_count: { $gt: 1 } }
    },

    {
      $lookup: {
        from: "quiz",
        localField: "_id",
        foreignField: null,
        pipeline: [
          {
            $match: {
              $expr: {
                $and: [
                  { $eq: ["$user_id", "$$ROOT._id.user_id"] },
                  { $eq: ["$course_id", "$$ROOT._id.course_id"] }
                ]
              }
            }
          }
        ],
        as: "quiz_docs"
      }
    },
    { $unwind: "$quiz_docs" },

    {
      $lookup: {
        from: "quiz_question",
        localField: "quiz_docs.quiz_id",
        foreignField: "quiz_id",
        as: "questions"
      }
    },

```

```
    {
      $lookup: {
        from: "user",
        localField: "quiz_docs.user_id",
        foreignField: "user_id",
        as: "user"
      }
    },
    { $unwind: "$user" },

    {
      $lookup: {
        from: "course",
        localField: "quiz_docs.course_id",
        foreignField: "course_id",
        as: "course"
      }
    },
    { $unwind: "$course" },

    {
      $addFields: {
        avg_grade: { $avg: "$questions.grade" }
      }
    },

    { $match: { avg_grade: { $lt: 50 } } },

    {
      $project: {
        _id: 0,
        quiz_id: "$quiz_docs.quiz_id",
        user_id: "$quiz_docs.user_id",
        user_name: "$user.name",
        course_id: "$quiz_docs.course_id",
        course_name: "$course.course_name",
        avg_grade: 1,
        quiz_date: "$quiz_docs.date"
      }
    },

    { $sort: { user_name: 1, course_name: 1, quiz_date: 1 } }
  ]
)
```

-- view 6:

```
db.createView(
  "v_assignments_due_within_week",
  "course_work",
  [
    {
      $match: {
        due_date: {
          $gte: new Date(),
          $lt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000)
        }
      }
    },

    {
      $lookup: {
        from: "user",
        localField: "user_id",
        foreignField: "user_id",
        as: "user"
      }
    },
    { $unwind: "$user" },

    {
      $lookup: {
        from: "course",
        localField: "course_id",
        foreignField: "course_id",
        as: "course"
      }
    },
    { $unwind: "$course" },

    {
      $project: {
        cw_id: 1,
        user_id: 1,
        user_name: "$user.name",
        course_id: 1,
        course_name: "$course.course_name",
        cw_title: 1,
        cw_type: 1,
        description: 1,
        due_date: 1,
        grade: 1
      }
    },

    { $sort: { due_date: 1 } }
  ]
)
```

View 8

```
db.createView(
  "user_accumulated_study_time",
  "study_section",
  [
    {
      $lookup: {
        from: "user",
        localField: "user_id",
        foreignField: "user_id",
        as: "user"
      }
    },
    { $unwind: "$user" },
    {
      $addFields: {
        duration_minutes: {
          $divide: [
            { $subtract: ["$end_time", "$start_time"] },
            1000 * 60
          ]
        }
      }
    },
    {
      $group: {
        _id: "$user_id",
        name: { $first: "$user.name" },
        time_accumulated_in_hour: {
          $sum: { $divide: ["$duration_minutes", 60] }
        }
      }
    },
    {
      $project: {
        _id: 0,
        user_id: "$_id",
        name: 1,
        time_accumulated_in_hour: 1
      }
    }
  ]
)
```

-- view 9:

```
db.createView(
  "user_1001_assignment_or_quiz_tag",
  "calendar_event",
  [
    { $match: { user_id: 1001 } },
    {
      $lookup: {
        from: "event_tag",
        localField: "event_id",
        foreignField: "event_id",
        as: "tags"
      }
    },
    { $unwind: "$tags" },
    {
      $match: {
        "tags.tag_name": { $in: ["assignment", "quiz"] }
      }
    },
    {
      $project: {
        _id: 0,
        event_name: 1,
        start_date: 1,
        end_date: 1,
        description: 1
      }
    }
  ]
)

```

-- view 10:

```
db.createView(
  "user_1001_courses_average",
  "course_work",
  [
    { $match: { user_id: 1001 } },
    {
      $lookup: {
        from: "course",
        localField: "course_id",
        foreignField: "course_id",
        as: "course"
      }
    },
    { $unwind: "$course" },
    {
      $group: {
        _id: "$course.course_name",
        average_course_grade: { $avg: "$grade" }
      }
    },
    {
      $project: {
        _id: 0,
        course_name: "$_id",
        average_course_grade: 1
      }
    }
  ]
)
```

Due to the change of database from MySQL to MongoDB, and change of schemas, some views from Phrase II of the project could not be recreated.

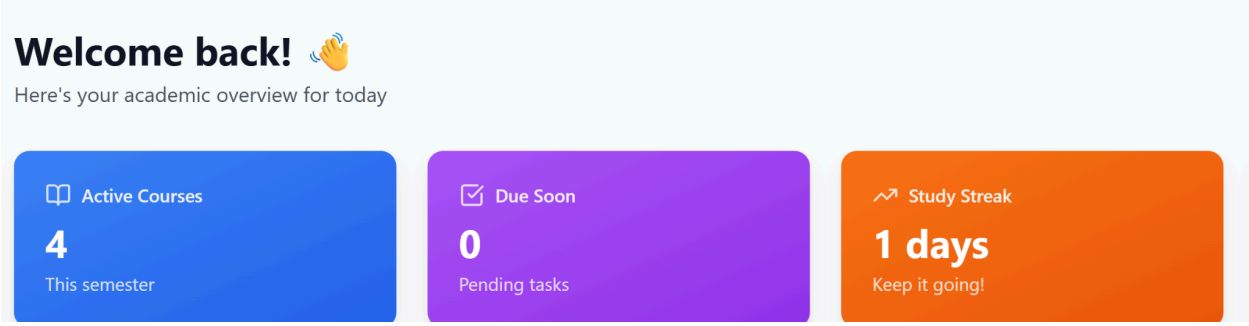
- Forms:
    - Forms to insert tuples in database(course work, assignment, quiz and course resource)

- Fetch API:

```
router.get("/holidays", validateQuery(Schemas.HolidayQuerySchema),async(req,res)=>{
    try{
        const {year, countryCode = "CA"} = req.query;
        if(!year){
            return res.status(400).json({error: "year is required for query"});
        }

        const holidays = await Holiday.find({
            country_code: countryCode,
            year: parseInt(year)
        });
        holidays.sort((a, b) => new Date(a.date) - new Date(b.date));


        res.json({year, countryCode, holidays});
    }catch(error){
        res.status(500).json({error: "Cannot load Holiday"});
    }
});
```

```
//for holiday//
router.post("/sync",validate(Schemas.HolidaySchema), async (req ,res)=>{
    try{
        const {year,countryCode} = req.body;

        //retrieving list of holiday from nager//
        const nager_url = `https://date.nager.at/api/v3/publicholidays/${year}/${countryCode}`;
        const response = await axios.get(nager_url);
        const holidays = response.data;

        let count = 0; //counter of saved entry//

        for(let h of holidays){
            try{
                await Holiday.updateOne(
                    {country_code: countryCode, year, date: h.date},
                    {
                        country_code: countryCode,
                        year,
                        date: h.date,
                        local_name: h.localName,
                        name: h.name,
                        type: h.types?.join(",")?? "Public",
                    },
                    {upsert: true}
                );
                count++;
            }catch(error){
                console.log("Skip duplicatd entry", h.date);
            }
        }

        res.json({
            messasge: "Holidays synced",
            saved: count,
        });
    }catch (error){
        console.error(error);
        res.status(600).json({error: "Holidays failed to sync with Nager"});
    }
});
```

- Data Visualization



Weekly Schedule

Nigerian Holidays in November 2025

National Youth Day

Time	Monday Nov 3	Tuesday Nov 4	Wednesday Nov 5	Thursday Nov 6	Friday Nov 7
8 AM					
9 AM	lecture Room 301 09:00 - 10:30		lecture Room 301 09:00 - 10:30		
10 AM					lab Physics Lab 2 10:00 - 12:00
11 AM		lecture Room 205 11:00 - 12:30			
12 PM					
1 PM	lecture Science Building A 13:00 - 14:30				

credits

Dr. Jane Doe
Fall 2025
wefwg

credits

Dr. Nobody
Fall 2026
wgwbdfb

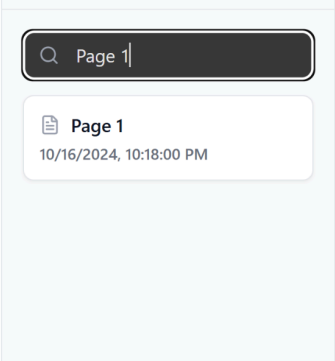
credits

Dr. Him
Fall 2025
ssgdhb

credits

dr. jane do
Fall 2024

- Search and Filter Functionality:



Backend:

User Authentication

Most of the post route is defined in the ModelsRoute.js in a for loop as they have similar format, including the route to post new users to the database.

```
for (const m of models){
    router.post(`/${m}`, validate(zod_schemas[m]),
create_controllers[m]);
}
```

The route to create user would then passes the created user object to the createUser controller in ModelControllers.js:

```
export const createUser = async (req, res) => {
    try {
        console.log("Creating user:", req.body);
        let {username, email, pswd_hash} = req.body;
        pswd_hash = await bcrypt.hash(pswd_hash,10);
        const newUserObj = new Models.User({
            username,
            email,
            pswd_hash: pswd_hash
        });
        await newUserObj.save();
        res.status(201).json(newUserObj);
    } catch (error) {
```

```
            err_500(json);
        }
    };
```

In the controller, the password input is hashed by *bcrypt*, insert to a new user object and save to the database

Login works similar except the controller become loginAttempt from AuthenticationController.js:

```
export const loginAttempt = async (req,res) =>{
    try{
        const {email,password} = req.body;
        const user = await Models.User.findOne({email});
        if (!user) {
            console.log("no user found");
            return res.status(400).json({ error: "User not found"
});
        }
        const match = await bcrypt.compare(password,user.pswd_hash);
        if(!match) return res.status(401).json({error: "Wrong
Password"});
        res.json(user);
    }catch(error){
        err_500(res,error);
    }
};
```

This controller fetches the user object with the email input, compares the hashed password using *bcrypt*. If successful, the controller returns the user object to the App.jsx where it is stored in the state of the component for other components to reference.

Created Users stored in MongoDB with unique email and hashed password:

Data Validation:

As the code snippet from above show, the application uses *zod* to define expected object format:

```
export const UserSchema = z.object({
    username: z.string().min(1),
    email: z.string().email(),
    pswd_hash: z.string().min(1)
});
```

While the validate() function came from ValidationEntry.js:

```
export const validateParams = (schema) => (req,res,next) =>{
    try{
        req.params = schema.parse(req.params);
        next();
    }catch (error){
        return res.status(400).json({errors: error.errors});
    }
};

export const validateQuery = (schema) => (req,res,next) =>{
    try{
        req.query = schema.parse(req.query);
        next();
    }catch(error){
        return res.status(400).json({errors: error.errors});
    }
};

export const validate = (schema) => (req,res,next)=>{
    try{
        req.body = schema.parse(req.body);
        next();
    }catch(error){
        res.status(400).json({errors: error.errors});
    }
};
```

API Integration:

The application support xml file exportation where a collection of user's coursework and the their total weighted grade are shown.:

```
//fetch course work//
        const cw = await CourseWork.find({
            student_id: user_id,
            course_id: course_id
        })
        cw.sort({due_date: 1});


        //writing xml//
        let total = 0;

        let xml_export = `<?xml version="1.0"
encoding="UTF-8"?>\n<coursework student="${studentId}"
course="${courseId}">\n`;

        for (let c in cw){
            xml+=`<item>\n`;
            xml+=`  <title>${c.cw_name}</title>\n`;
            xml+=`  <grade>${c.cw_grade}</grade>\n`;
            xml+=`  <weight>${c.cw_weight}</weight>\n`;
            xml+=`</item>\n`;
            total += c.cw_grade*c.cw_weight;
        }
        xml += `<total>${total}</total>\n`;
        xml += `</coursework>`;

        res.set("Content-Type","application/xml");
        res.send(xml);
```

External API:

In the route file, there is a post route that uses nager API to fetch data of Canadian Holiday, then stores it in the database. These data are used to highlight user schedule when a day in the weekly schedule is a Canadian public holiday:

```
//for holiday//
router.post("/sync",validate(Schemas.HolidaySchema), async (req ,res)=>{
    try{
```

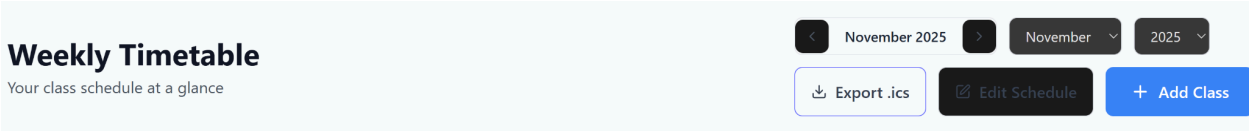
```
        const {year,countryCode} = req.body;

        //retrieving list of holiday from nager//
        const nager_url =
`https://date.nager.at/api/v3/publicholidays/${year}/${countryCode}`;
        const response = await axios.get(nager_url);
        const holidays = response.data;

        let count = 0; //counter of saved entry//

        for(let h of holidays){
            try{
                await Holiday.updateOne(
                    {country_code: countryCode, year, date: h.date},
                    {
                        country_code: countryCode,
                        year,
                        date: h.date,
                        local_name: h.localName,
                        name: h.name,
                        type: h.types?.join(",")?? "Public",
                    },
                    {upsert: true}
                );
```

There are also other routes in the route file that allows the application to update and delete existing database entries.

```
router.put("/course/:courseId",
validateParams(Schemas.makeIdSchema("courseId")),
validate(Schemas.CourseSchema), updateControllers.updateCourse);

//for course work//
router.put("/coursework/:courseWorkId",
validateParams(Schemas.makeIdSchema("courseWorkId")),
validate(Schemas.CourseWorkSchema), updateControllers.updateCourseWork);

//for resource//
```

```
router.put("/resource/:resourceId",
validateParams(Schemas.makeIdSchema("resourceId")),
validate(Schemas.ResourceSchema), updateControllers.updateResource);
. . .
router.delete("/course/:courseId",
validateParams(Schemas.makeIdSchema("courseId")),
deleteControllers.deleteCourse);

//for course work//
router.delete("/coursework/:courseWorkId",
validateParams(Schemas.makeIdSchema("courseWorkId")),
deleteControllers.deleteCourseWork);
```

Data Export:



- Error Handling:

```
const ensureUserRecord = useCallback(
  async ({ name, email }) => {
    if (!email) {
      throw new Error("Email is required");
    }
    try {
      return await fetchUserByEmail(email);
    } catch (error) {
      if (error.response?.status === 404) {
        const payload = {
          username: name || email.split("@")[0],
          email,
          pswd_hash: generatePlaceholderPassword(),
        };
        const createResponse = await axios.post(`${API_BASE_URL}/api/user`, payload);
        return createResponse.data;
      }
      throw error;
    }
  },
  [fetchUserByEmail]
);
```

Conclusion and Reflection:

In this project, we successfully deliver a full-stack student study platform that stores user's data to a database, and uses them to support students in managing their study schedule and improve efficiency. Throughout the project we implement most of the proposed functionality and services, including dynamic form, export web service, API integration, use of Fetch API and many more. These technologies allow us to build seamless communication between the application frontend and backend.

Overall, this project has given us valuable experience on implementing full-stack development principles and unfamiliar backend technologies. The major challenge we face in this project is poor time distribution and frontend structure, as we have underestimated the time for us to build the backend and its connection to the frontend, thus weakening the quality, robustness and security of the connection. For future projects, we would aim to maintain a balanced effort on both frontend and backend.