YORK REGION EDUCATIONAL SERVICES (YRES)

Summer Camp Scheduler Web Application

-presented by development team
U of T CSC301 Team 48

CONTENTS

INTRODUCTION

OUR PARTNER

- Our partner is the York Region Educational Services (YRES)
- NPO ID: 796041218 Official website: https://yorkeducation.org/

PROBLEM WE AIM TO SOLVE

- We aim to help with the inaccuracy and time consuming nature of summer camp scheduling

Looking for Campers

2024 Summer Camp (July 8 – Aug. 16, 2024)

Welcome to our exciting summer camp in York Region! Our camp is the ideal destination for families looking for an enriching and affordable summer experience for their children.

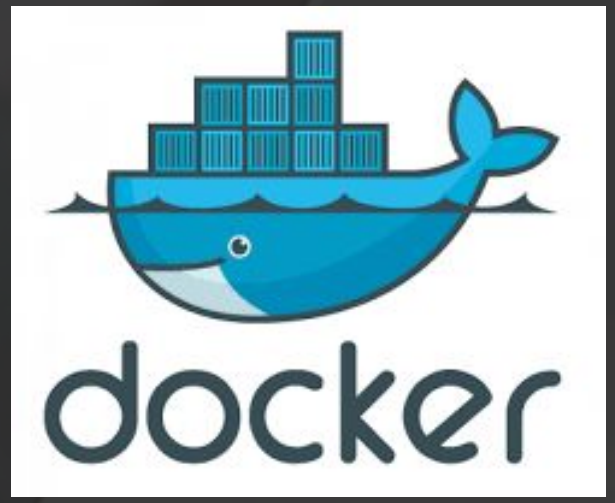
We offer a diverse range of activities, including sports, arts and crafts, STEM

Join us for a summer of fun, friendship, and learning!

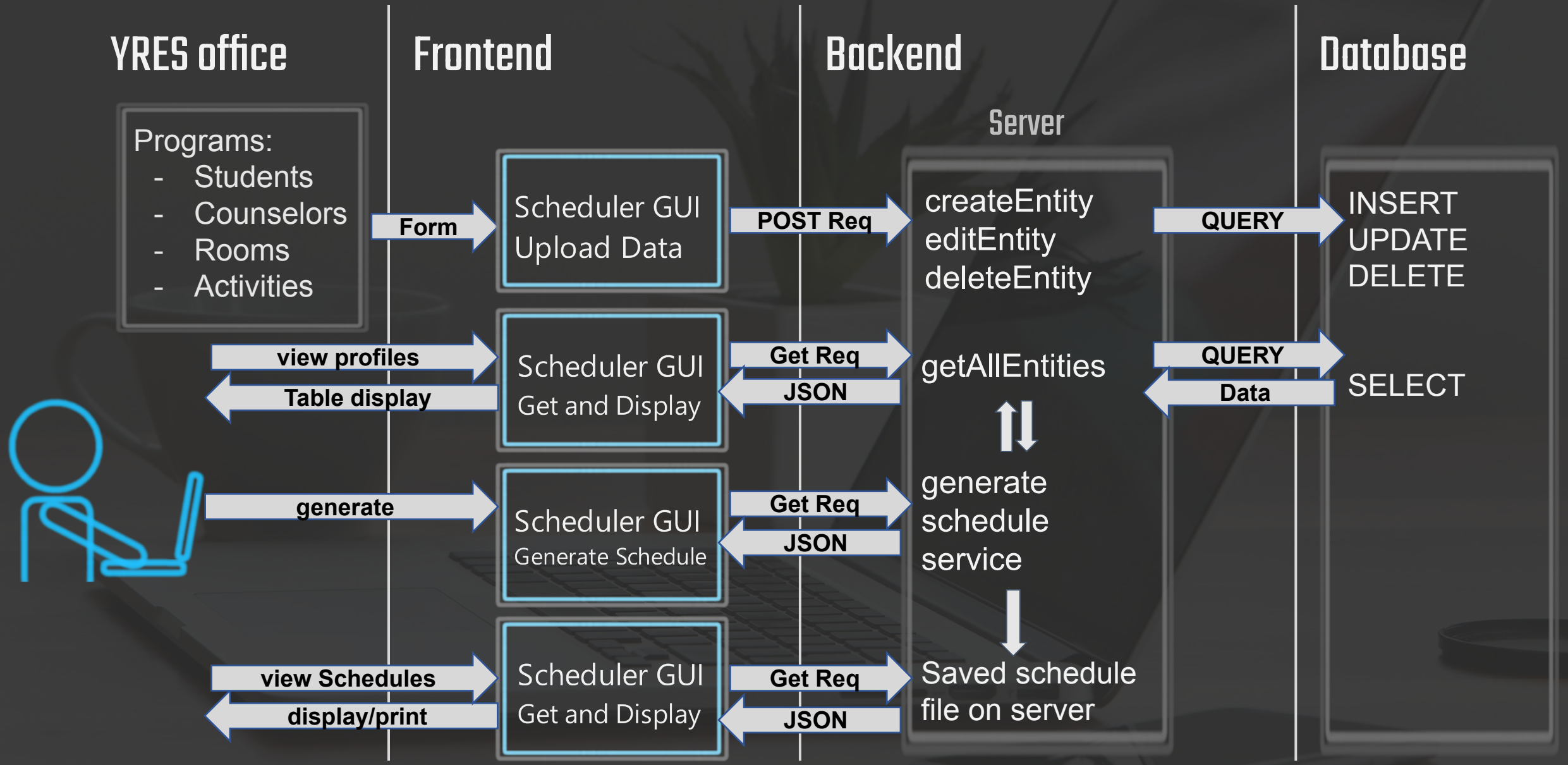
Register Now

ARCHITECTURE AND TECHNICAL DISCUSSION

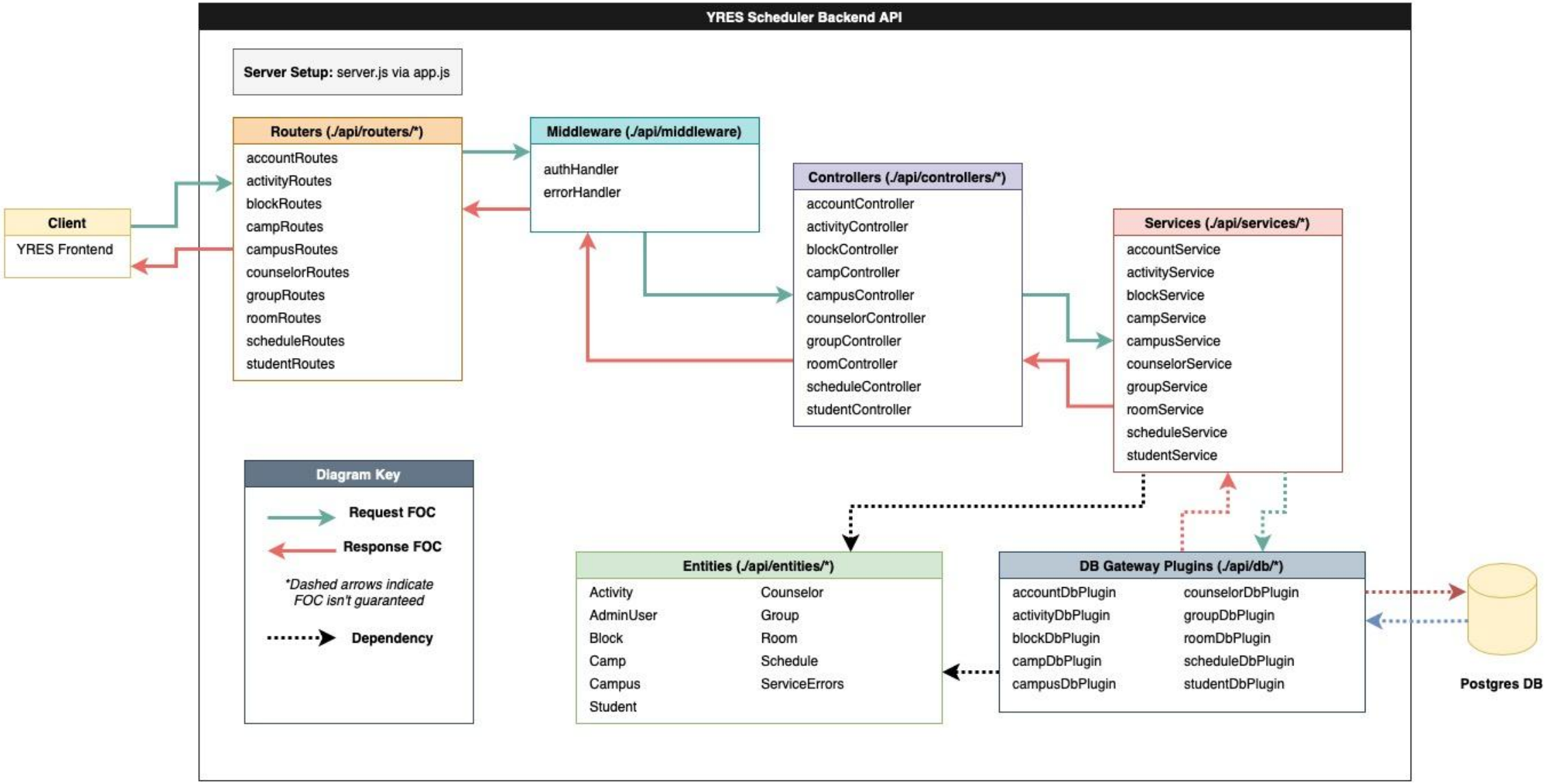
ARCHITECTURE - TECH STACK



ARCHITECTURE - WORKFLOW

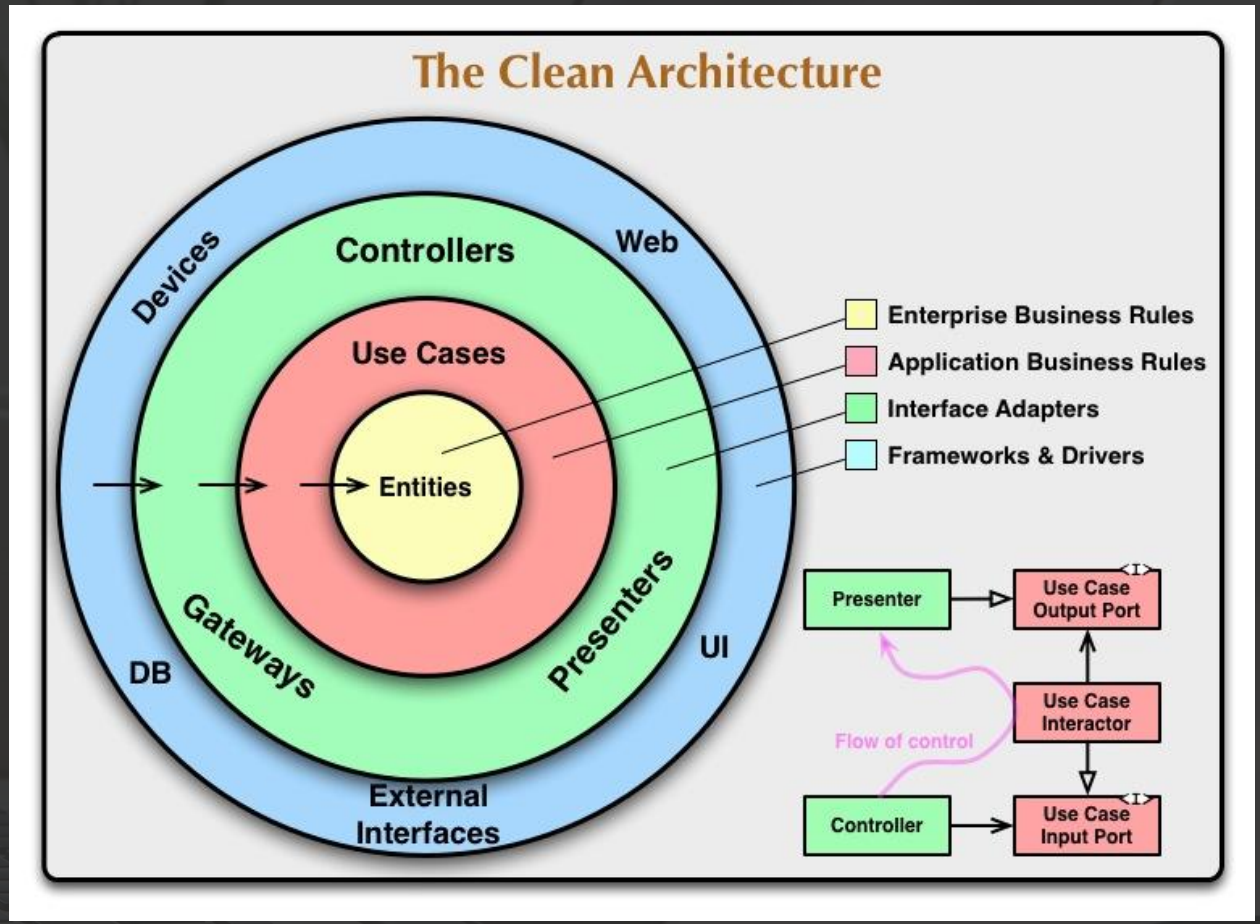


ARCHITECTURE - BACKEND API



ARCHITECTURE - CLEAN & SERVICE-BASED

- *Backend API loosely follows a CLEAN & service-based architecture, with some differences.*
- *Related use cases are grouped, most commonly be entity (e.g. student service, camp service etc).*
- *Controllers also serve as presenters in the interface adapters layer.*
- *Violation: DB gateways are accessed directly by services in the use case layer without dependency inversion.*
- *We have deemed this to be ok since no circular dependencies arise and the parameters/responses are implementation non-specific.*



ARCHITECTURE - BACKEND API ROUTERS

For Example: Student Service Router

```
const studentController = require('../controllers/studentController');
const logger = require('../../logger');
const auth = require('../middleware/authHandler');

const studentRoutes = (app) => {
    /**
     * Route to get all students.
     * @name GET /student/all/
     * @function
     * @memberof module:routes/studentRoutes
     * @param {Object} req - The Express request object.
     * @param {Object} res - The Express response object.
     * @returns {Promise} A Promise that resolves to the result of the get
     */
    app.get('/student/all/', auth, async (req, res) => {
        logger.info(`GET /student/all/`);
        const resp = await studentController.getAllStudents(req, res);
        res.send(resp);
    })
```

...

**Many more endpoints in rest of file*

- *Specifies the routes for API endpoints (e.g. GET /student/all/).*
- *Specify additional middleware for a route if necessary (e.g. auth middleware if route requires authentication).*
- *Calls the corresponding controller operation and then sends its response.*

ARCHITECTURE - BACKEND API CONTROLLERS

For Example: Student Service Controller

```
const studentService = require('../services/studentService');
const {StudentServiceError, STATUS_CODES} = require('../entities/ServiceErrors');
const logger = require('../../logger');

/**
 * Retrieves all students.
 * @param {Object} req - The request object.
 * @param {Object} res - The response object.
 * @returns {Object} - An object containing an array of students with friend and enemy IDs.
 */
async function getAllStudents(req, res) {

    const all_students = await studentService.getAllStudents();

    res.status(STATUS_CODES.SUCCESS);

    return {
        students: all_students.map((student) => {
            return {
                ...student,
                friend_ids: student.getFriendIds(),
                enemy_ids: student.getEnemyIds()
            };
        })
    };

}
```

...

**Many more service functions in rest of file*

- *Responsible for unpacking and validating variables from request object (i.e. query parameters and request body).*
- *Calls the corresponding service function.*
- *If an error is not thrown, the service function is successful and so the response object status code is set accordingly.*
- *Finally, the body for the response object is prepared and returned to the router.*

ARCHITECTURE - BACKEND API SERVICES

For Example: Student Service

```
const db = require('../db/studentDbPlugin');
const {StudentServiceError, STATUS_CODES} = require('../entities/ServiceErrors');

/**
 * Retrieves all students from the database.
 *
 * @returns {Array<Student>} An array of student objects.
 */
async function getAllStudents() {
    try {
        return await db.getAllStudents();
    } catch(err) {
        throw new StudentServiceError(
            `DB Operation Failure: ${err}`,
            STATUS_CODES.FAILED
        );
    }
}
```

...

**Many more service functions in rest of file*

- *Implements the use case logic for an API call.*
- *Call functions in corresponding DB gateway plugin where necessary in order to manage and query persistent storage.*
- *Throws service-specific custom errors with appropriate status codes and messages.*
- *Returns data or a status flag to the controller.*

ARCHITECTURE - BACKEND API ENTITIES

For Example: Student Entity

```
module.exports = class Student {
    /**
     * Create a new student profile.
     *
     * @param {number} student_id - The unique ID of this student.
     * @param {number} student_ui_id - The unique ID of this student for the UI
     * @param {string} lastname
     * @param {string} firstname
     * @param {number} age
     * @param {string} sex
     * @param {number} campus_id  -- Campus id that the student belongs to
     * @param {Set<string>} friend_ids - A set of student IDs that this student prefers to work with.
     * @param {Set<string>} enemy_ids - A set of student IDs that this student doesn't want to work with.
    */
    constructor(
        student_id,
        student_ui_id,
        lastname,
        firstname,
        age,
        sex,
        campus_id,
        friend_ids,
        enemy_ids) {

        this._student_id = student_id;
        this._student_ui_id = student_ui_id;
        this.lastname = lastname;
        this.firstname = firstname;
        this.age = age;
        this.sex = sex;
        this.campus_id = campus_id;
        this.friend_ids = friend_ids;
        this.enemy_ids = enemy_ids;
    }
```

...

**Class Definition continues in rest of file*

- *Defines attributes and behaviour for an entity involved in application use cases.*
- *Often corresponds to an entity within the DB (although structured differently due to schema restrictions).*
- *Provides common data structures to be used across services and use cases.*

ARCHITECTURE - BACKEND API DB GATEWAY PLUGINS

For Example: Student DB Plugin

```
/**
 * Retrieves all students from the database and maps them to Student objects.
 *
 * @returns {Array<Student>} An array of Student objects.
 */
async function getAllStudents() {
    const queryGetAllStudents = `
        SELECT
            S.student_id,
            S.student_ui_id,
            S.firstname,
            S.lastname,
            S.campus_id,
            S.age,
            S.sex
        FROM
            Student S;
    `;
    const functionName = getAllStudents.name; // Get the name of the current function for logging purposes
    logger.debug(`Function ${functionName}: Getting all students in the studentDbPlugin`);
    var students;
    try {
        const result = await client.query(queryGetAllStudents);

        if (result && result.rowCount > 0) {
            const rows = result.rows;

            students = rows.map(mapRowToStudent);

            for (var i=0; i<students.length; i++) {
                students[i] = await getFriendPreferencesAndCategorize(students[i]);
            }

            return students;
        } else {
            return [];
        }
    } catch (err) {
        throw new Error(err);
    }
}
```

...

**Many more DB gateway functions in rest of file*

- *Uses a connection pool to carry out PSQL operations such as INSERT, DELETE and SELECT.*
- *Formats raw DB responses into the API entities which are then returned to the service logic.*

ARCHITECTURE - BACKEND API ERROR HANDLING

Error Handling Middleware:

```
const DEFAULT_STATUS_CODE = 500;

const errorHandler = (err, req, res, next) => {
    const status_code = err.status_code || DEFAULT_STATUS_CODE;
    res.status(status_code).send(
        {
            error: err.name,
            message: err.message,
            trace: err.stack
        }
    );
    next();

}

module.exports = errorHandler;
```

Throwing Custom Error in Service:

```
if (await db.existsUser(username)) {
    throw new AccountServiceError(
        `User already exists for username '${username}'`,
        STATUS_CODES.CONFLICT
    );
}
```

Informative Error Response:

Body 409 Conflict 53 ms 1.21 KB Save as Example

Pretty | Raw | Preview | Visualize | JSON


```
{
    "error": "AccountServiceError",
    "message": "User already exists for username 'admin'",
    "trace": "AccountServiceError: User already exists for username 'admin'\n
        at Object.signup (/Users/harvey.donnelly/Desktop/
        project-48-yorkregioneducationalservices-T/deliverables/yres_scheduler/
        yres_scheduler_backend/api/services/accountService.js:73:15)\n    at
        process.processTicksAndRejections (node:internal/process/
        task_queues:95:5)\n    at async Object.signup (/Users/harvey.donnelly/
        Desktop/project-48-yorkregioneducationalservices-T/deliverables/
        yres_scheduler/yres_scheduler_backend/api/controllers/accountController.
        js:62:19)\n    at async /Users/harvey.donnelly/Desktop/
        project-48-yorkregioneducationalservices-T/deliverables/yres_scheduler/
        yres_scheduler_backend/api/routes/accountRoutes.js:36:18"
}
```

ARCHITECTURE - BACKEND API AUTHENTICATION

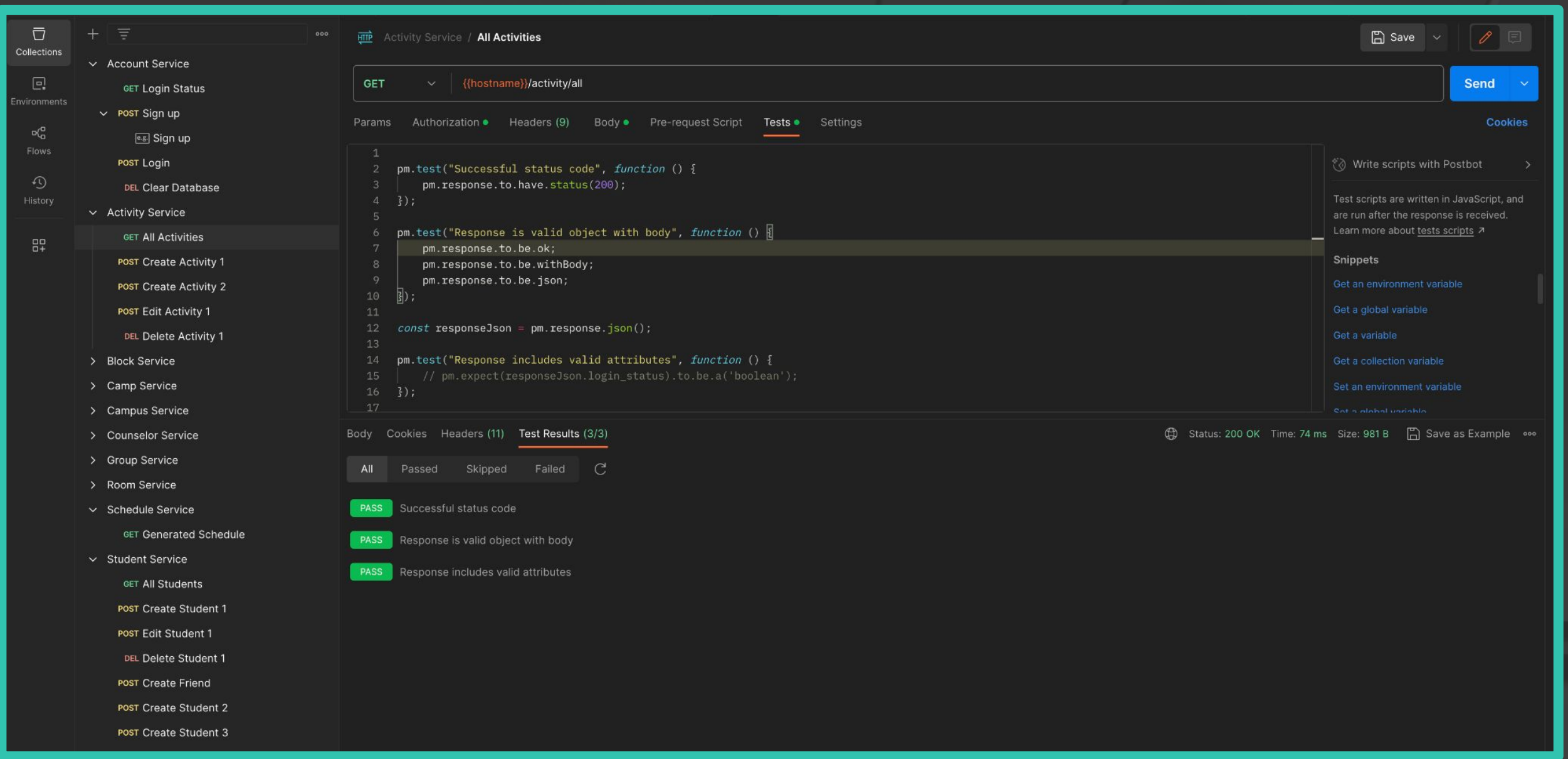
Protected requests require valid JWT in request header, otherwise they are blocked by Auth middleware.

Snippet from Auth Middleware:

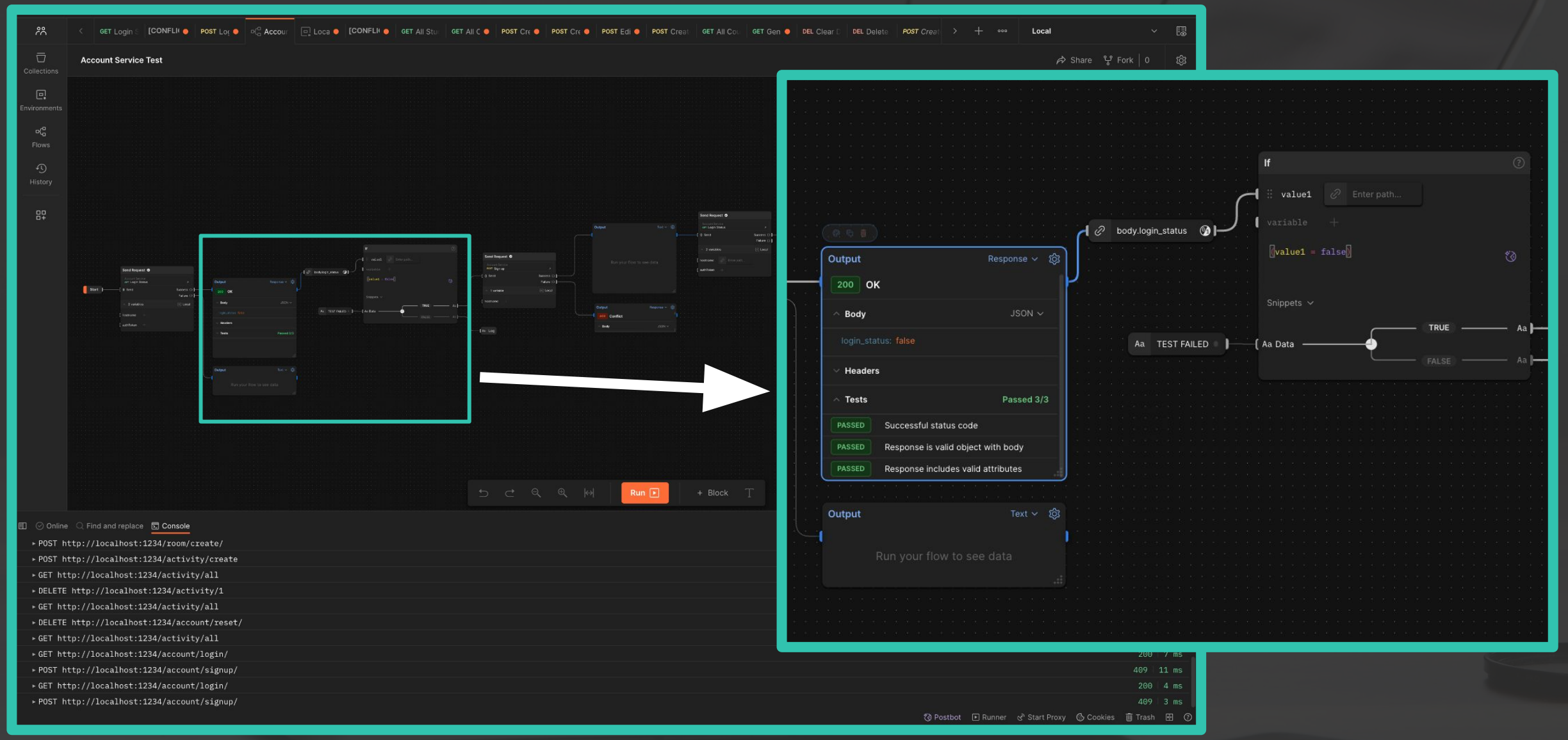
```
if (typeof header !== "undefined") {
    const token = header.split(" ")[1];
    req.token = token;

    jwt.verify(req.token, AUTH_TOKEN_SECRET, (err, data) => {
        if (err) {
            res.sendStatus(STATUS_CODES.FORBIDDEN);
        } else {
            next();
        }
    });
} else {
    res.sendStatus(STATUS_CODES.FORBIDDEN);
}
```

A valid token can be generated via POST /account/login with a valid username and password combination.

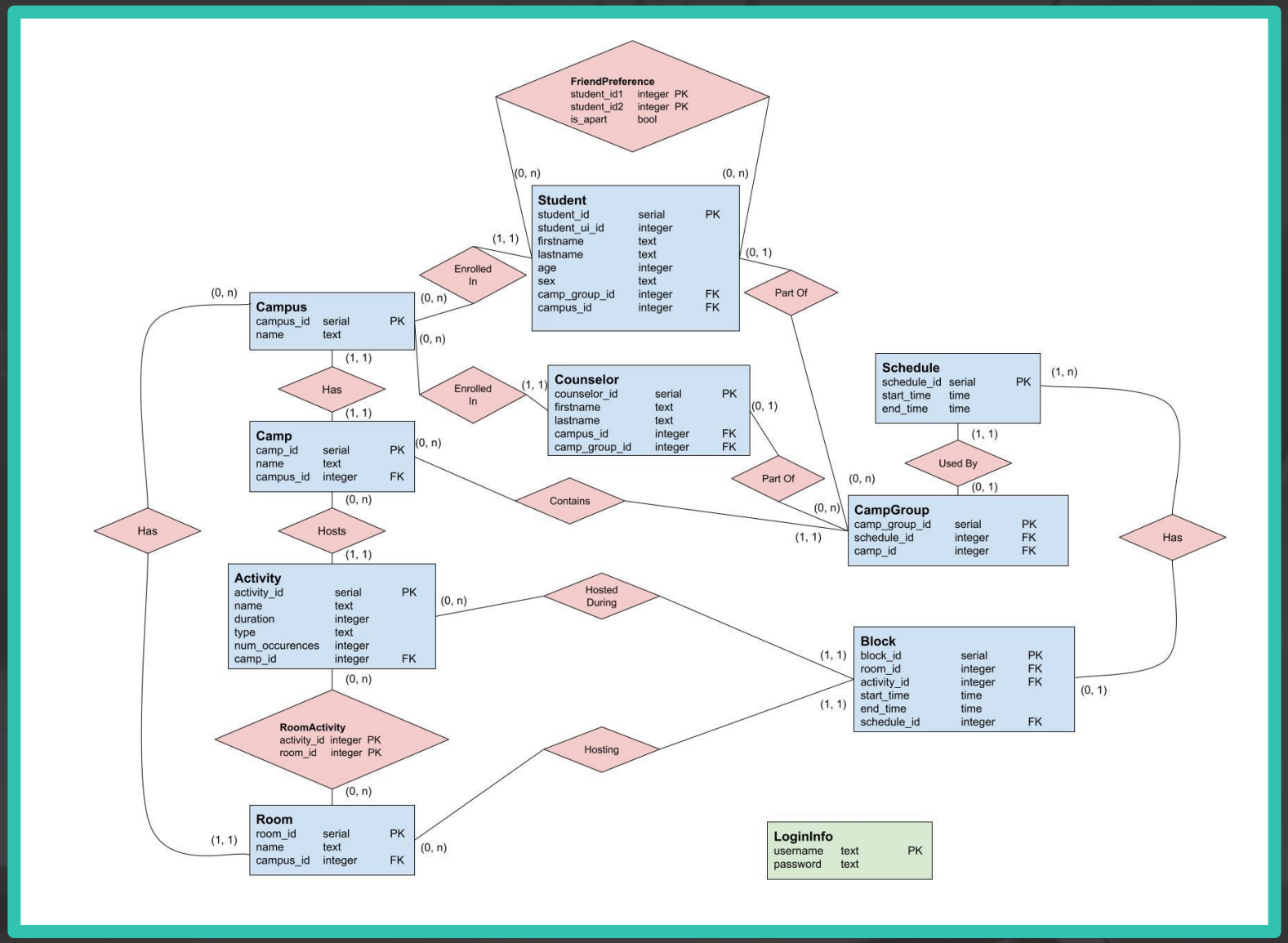
TECHNICAL DISCUSSION - POSTMAN COLLECTION & TESTING



TECHNICAL DISCUSSION – POSTMAN FLOWS & SERVICE TESTING



ARCHITECTURE - DATA



ARCHITECTURE - DATA

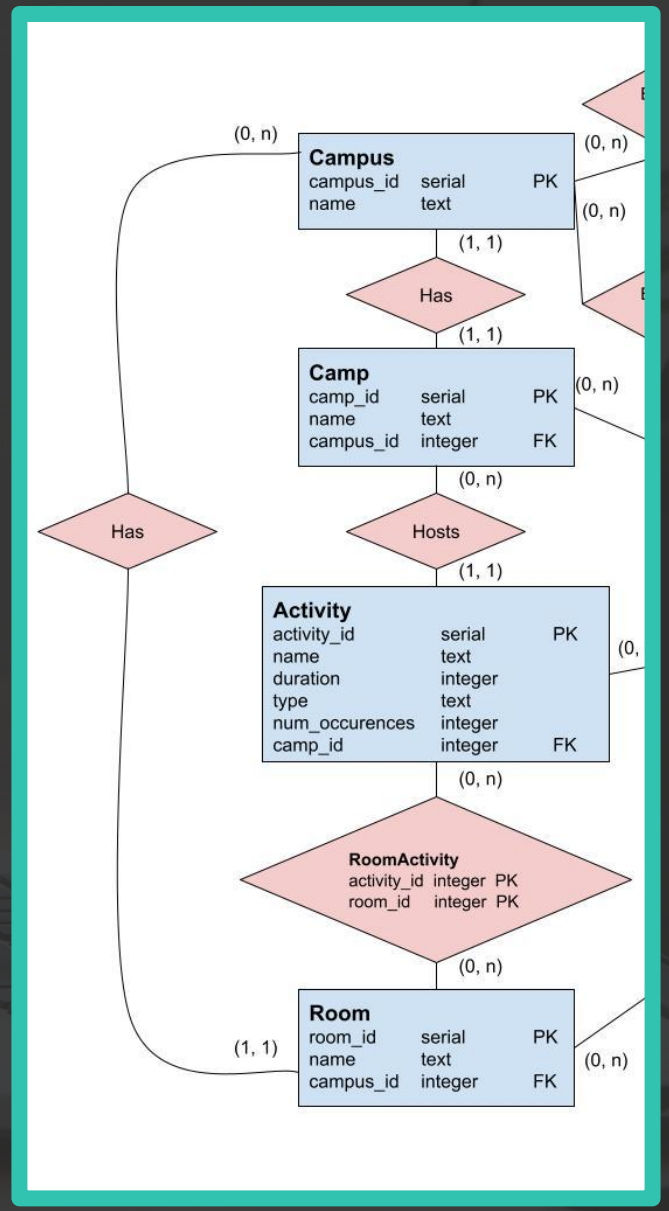
Campus

- Physical campsite
- One core instance
- Contains name

Camp

- Camp program entity
- One core instance
- Contains name, campus

```
-- Represents a campus with rooms for camps to take place
-- Columns:
    -- campus_id : The auto generated unique ID
    -- name : <UI> The name of the campus
create table Campus (
    campus_id serial primary key,
    name text not null
);


-- Represents a camp, i.e. a classification of groups (based on program type and/or student age)
-- Columns:
    -- camp_id : The auto generated unique ID
    -- name : <UI> The name of the camp
    -- campus_id : Foredign key constraint
create table Camp (
    camp_id serial primary key,
    name text not null,
    campus_id integer references Campus
);
```




```
-- Intermediary table for many-to-many relationship between Rooms and Activities.
-- Columns:
    -- room_id : The id of the room
    -- activty_id : The id of the activity
create table RoomActivity (
    activity_id integer references Activity on delete cascade,
    room_id integer references Room on delete cascade,
    primary key (activity_id, room_id)
);
```

Activity

- Class/activity/other timed event
- Contains name, duration,
- type (“filler”, “class”, “study”),
- number of occurrences, camp

Room

- Room inside a building
- Contains name, campus

RoomActivity

- Intermediary for many-to-many relationship

ARCHITECTURE - DATA

Student

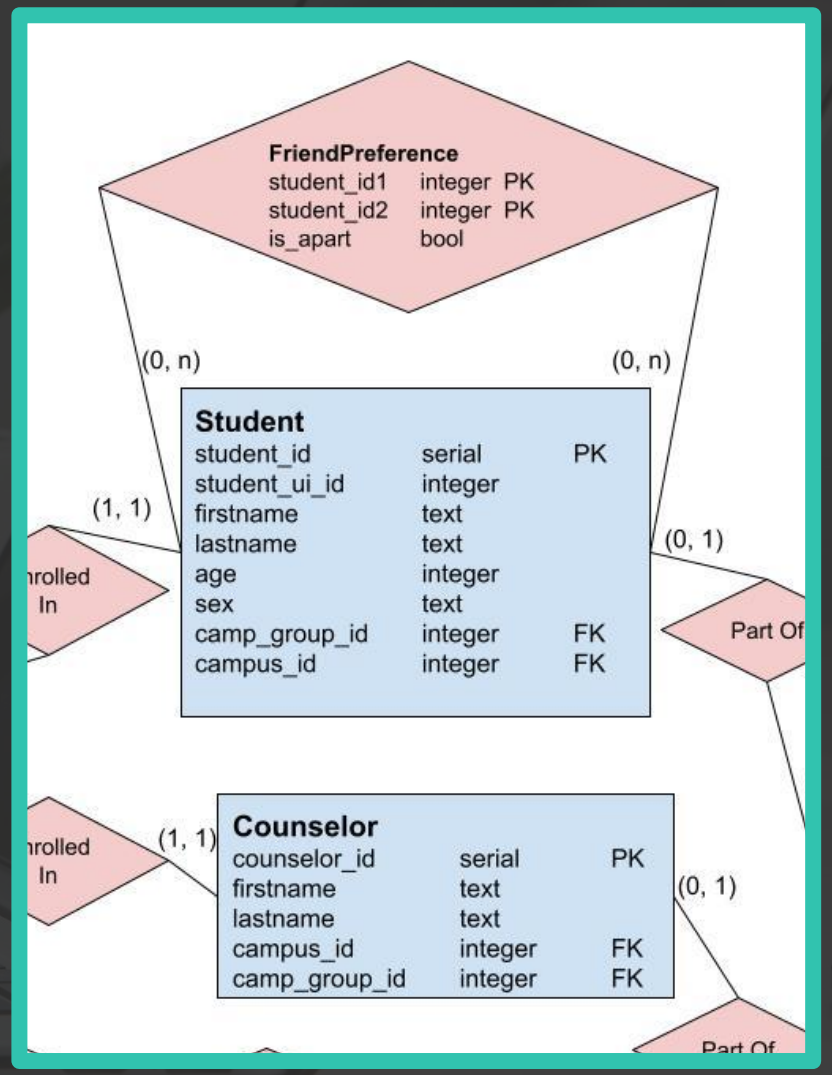
- Camp participant
- Contains student number, name, age,
- sex, group, campus

Counselor

- Camp staff member
- Contains name, campus, group

FriendPreference

- Intermediary for recursive relationship
- Students to keep together/apart




```sql
-- The friend preferences for all the students.
-- Columns:
    -- student_id1 : The student id of the first friend
    -- student_id2 : The student id of the second friend
    -- is_apart : If the students should be apart
create table FriendPreference (
    student_id1 integer not null references Student(student_id) on delete cascade,
    student_id2 integer not null references Student(student_id) on delete cascade,
    is_apart bool not null,
    primary key (student_id1, student_id2), -- Primary key
    check (student_id1 > student_id2) -- Make sure they are not duplicates
);
```

ARCHITECTURE - DATA

Schedule

- Timetable for students/counselors
- Contains start/end times

Block

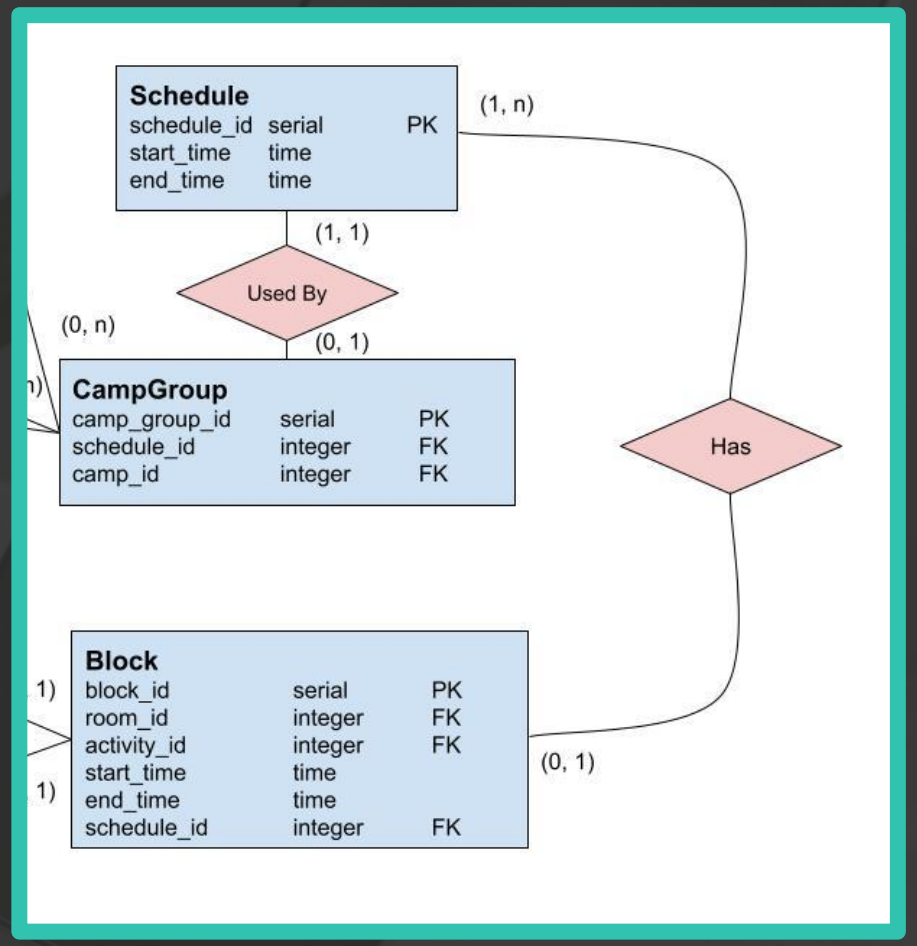
- Schedule Block entity
- Contains room, activity,
- start/end times, schedule

CampGroup

- Group of students and counselors
- Contains schedule, camp

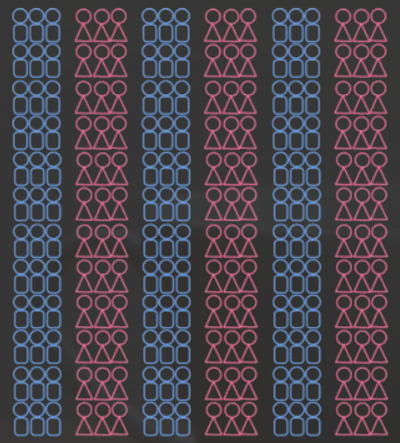
LoginInfo

- Authentication entity
- username and password



TECHNICAL DISCUSSION - GROUPING ALGORITHM

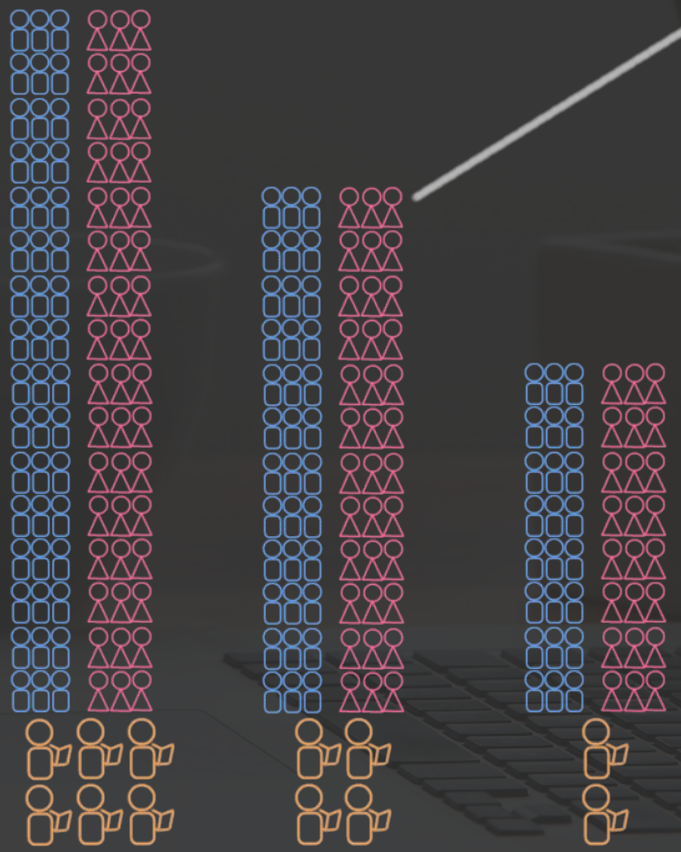
Parameters



List of all Students

List of all Counselors

Divide students according to their registered camp type.
Allocate counselors to each camp type depending on number of students.



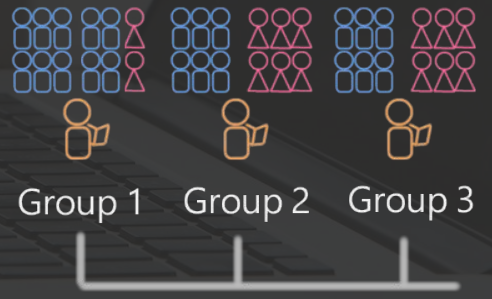
Students with friend preferences

(students that want to be in the same group)

Other students

(Arranged by gender ratio, to ensure even distribution)

Prioritize those with friend preferences.
Group them first.
Then assign the others randomly with best possible even distribution of gender.



Return

- Camp Type 1
  - Group 1
    - Students
    - Counselors
    - Empty Schedule [ ]
  - Group 2
  - More groups...
- Camp Type 2
  - More groups...
- More camps...

TECHNICAL DISCUSSION - SCHEDULING ALGORITHM

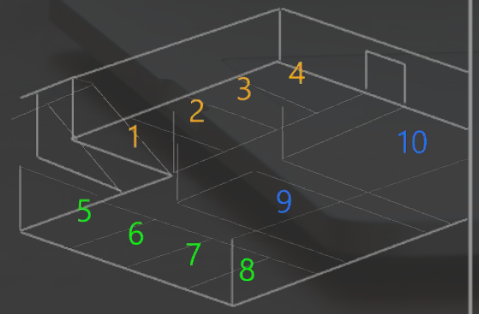
Parameters:
Returned list from
Grouping algorithm

- Camp Type 1
  - Group 1
    - Students
    - Counselors
    - Empty Schedule [ ]
  - ...
- ...

Math Class - 8 times - 1h
P.E. - 2 times - 2h
French Class - 5 times - 1h

...

Activities



Rooms

First separate activities
for each camp type.

Age 5-12	Age 12-18
Math 1	Math 2
English 1	English 2
Art 1	Physics
Music 1	Biology
Science	P.E.
...	...

... Camps

Wrap activities into blocks
based on their number of
occurences.

Age 5-12	Age 12-18
Art 1	P.E.
Art 1	P.E.
Math 1	P.E.
Math 1	English 2
Math 1	English 2
Math 1	English 2
...	...

... Camps

The list is sorted so that
longer activities are placed
at the front.

Number of days...

undefined	undefined
undefined	undefined
undefined	undefined
undefined	undefined
undefined	undefined

Initialization

Available rooms
at all time slot
Size = Schedule size

[1,..10]	[1,..10]
[1,..10]	[1,..10]
[1,..10]	[1,..10]
[1,..10]	[1,..10]

Initialization

For a limited number of attempts:
- For each block of each group from each camp:
- - Try inserting that block into the group's schedule
- - If an insertion is impossible: start anew
- If all blocks are inserted: return

If attempts are exhausted: fail

E.g. Try inserting a P.E. course, which takes 2 hours.
At a random slot with indices <[0, Days), [0, Hours - 2)>
say <0, 0>

undefined ✓	undefined
undefined ✓	undefined
undefined	undefined
undefined	undefined
undefined	undefined

OK

undefined ✓	undefined
Math 2 ✗	undefined
undefined	undefined
undefined	undefined
undefined	undefined

Fail — Retry with new indices

P.E. can only take place in room 9, 10

[5, 8, 9]	...
[4,..9, 10]	...

OK

[5, 6, 8] ✗	...
[4,..9, 10]	...

Fail

[5, 8, 9]	P.E.
[4,..9, 10]	P.E.

→ Go on to the next block

Return if no more blocks ☐

TECHNICAL DISCUSSION - Logging

```
// Create a custom format for log entries
const logger = winston.createLogger({
  level: 'http',
  format: combine(
    timestamp({
      format: 'YYYY-MM-DD hh:mm:ss.SSS A',
    }),
    json()
  ),
  transports:  [
    new winston.transports.File({
      filename: './logging/all-backend-logs.log',
    }),
    new winston.transports.File({
      filename: './logging/backend-http.log',
      level: 'http',
      format: combine(httpFilter(), timestamp(), json()),
    }),
    new winston.transports.File({
      filename: './logging/backend-error.log',
      level: 'error',
      format: combine(errorFilter(), timestamp(), json()),
    }),
    new winston.transports.File({
      filename: './logging/backend-info.log',
      level: 'info',
      format: combine(infoFilter(), timestamp(), json()),
    }),
    new winston.transports.File({
        filename: './logging/backend-debug.log',
        level: 'debug',
        format: combine(debugFilter(), timestamp(), json()),
      }),
  ],
});
```

all-backend-logs.log

deliverables > yres_scheduler > yres_scheduler_backend > logging > all-backend-logs.log


```
2920  {"level":"info","message":"server is running on port 1234","timestamp":"2023-11-30 06:33:01.980 PM"}
2921  {"level":"info","message":"Server is ready!","timestamp":"2023-11-30 06:33:02.318 PM"}
2922  {"level":"http","message":"GET /account/reset/ 404 34 - 1.751 ms","timestamp":"2023-11-30 06:33:13.374 PM"}
2923  {"level":"http","message":"GET /account/reset/ 404 34 - 3.854 ms","timestamp":"2023-11-30 06:34:01.156 PM"}
2924  {"level":"http","message":"DELETE /account/reset/ 200 20 - 871.355 ms","timestamp":"2023-11-30 06:34:25.709 PM"}
2925  {"level":"http","message":"DELETE /account/reset/ 200 20 - 575.583 ms","timestamp":"2023-11-30 06:34:53.474 PM"}
2926  {"level":"http","message":"DELETE /account/reset/ 200 20 - 570.769 ms","timestamp":"2023-11-30 06:34:54.767 PM"}
2927  {"level":"http","message":"DELETE /account/reset/ 200 20 - 456.259 ms","timestamp":"2023-11-30 06:34:55.735 PM"}
2928
```

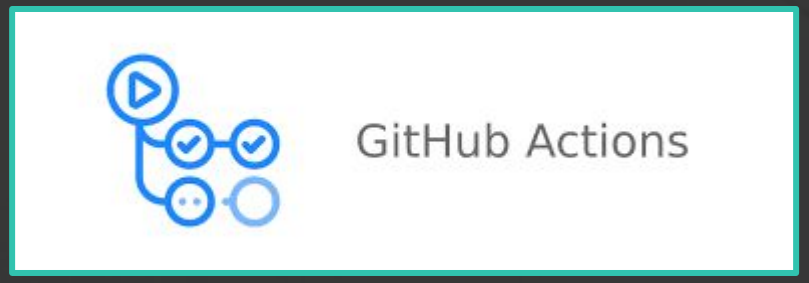

```
const functionName = createStudent.name; // Get the name of the current function for logging purposes      You,
logger.debug(`Function ${functionName}: Creating student in the studentDbPlugin`, { student_ui_id });

try {
```

- logging
  - all-backend-logs.log
  - backend-debug.log
  - backend-error.log
  - backend-http.log
  - backend-info.log


```
{"level":"debug","message":"Function createStudent: Creating student in the studentDbPlugin","student_ui_id":"10101","timestamp":"2023-12-01 01:31:29.444
 AM"}
{"level":"debug","message":"Function getAllStudents: Getting all students in the studentDbPlugin","timestamp":"2023-12-01 01:31:29.522 AM"}
{"level":"debug","message":"Function getAllCounselors: Getting all counselors from the database.","timestamp":"2023-12-01 01:31:29.607 AM"}
ubuntu@ip-172-31-41-128:~/project-48-yorkregioneducationalservices-T/deliverables/yres_scheduler/yres_scheduler_backend/logging$
```

PROCESS, DEPLOYMENT, ACCESSING

CI/CD



CI/CD

Backend CI

Backend Continous Deployment

Docker CD Database Pipeline

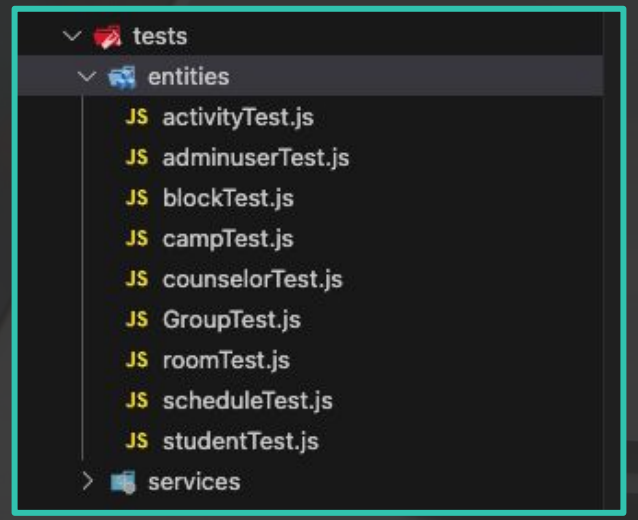
Frontend CI/CD

619 workflow runs — Event ▾ Status ▾ Branch ▾ Actor ▾

Update deployDB.yml **Backend CI** #225: Commit ada1e8f pushed by EricKarpovits	main	2 hours ago 33s
Update deployDB.yml **Docker CD Database Pipeline** #70: Commit ada1e8f pushed by EricKarpovits	main	2 hours ago 4m 35s
Update deployDB.yml **Backend Continous Deployment** #109: Commit ada1e8f pushed by EricKarpovits	main	2 hours ago 2m 38s

```yaml
You, 3 weeks ago | 1 author (You)

name: Backend CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [18.x]

    steps:
    - uses: actions/checkout@v3

    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3      You, 3 weeks ago • Commiting most recent changes
      with:
        node-version: ${{ matrix.node-version }}

    - name: Install Dependencies
      run: npm install
      working-directory: deliverables/yres_scheduler/yres_scheduler_backend

    - name: Run Node.js Server
      run: |
        node server.js &
          sleep 5  # Allow some time for the server to start and make sure nothing fails
      working-directory: deliverables/yres_scheduler/yres_scheduler_backend

    - name: Run all the backend tests
      run: npm test
      working-directory: deliverables/yres_scheduler/yres_scheduler_backend
```

- Set up job
- Run actions/checkout@v3
- Use Node.js 18.x
- Install Dependencies
- Run Node.js Server
- Run all the backend tests
- Post Use Node.js 18.x
- Post Run actions/checkout@v3
- Complete job



DB Deployment

all-backend-logs.log | backend-debug.log | docker-compose.yml | app.js

eliverables > yres_scheduler > yres_scheduler_database > docker-compose.yml

You, last week | 1 author (You)

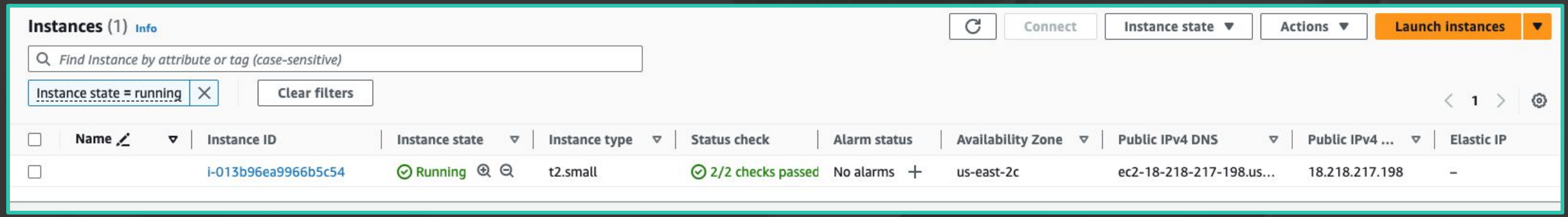
```yaml
version: "3"
services:
  db:
    image: postgres:latest
    container_name: db-container
    environment:
      POSTGRES_PASSWORD: csc301
      POSTGRES_USER: yres
      POSTGRES_DB: yres_db
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./yres_schema.sql:/docker-entrypoint-initdb.d/yres_schema.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U yres -d yres_db"]
      interval: 1s
      timeout: 5s
      retries: 5

  testing_db:
    image: postgres:latest
    container_name: testing-db-container
    environment:
      POSTGRES_PASSWORD: csc301
      POSTGRES_USER: yres
      POSTGRES_DB: testing_db # Database name for testing instance
    ports:
      - "5500:5432" # Different port mapping for testing instance
    volumes:
      - postgres-data-testing:/var/lib/postgresql/data
      - ./yres_schema.sql:/docker-entrypoint-initdb.d/yres_schema.sql # Shared SQL sc
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U yres -d testing_db"]
      interval: 1s
      timeout: 5s
      retries: 5

volumes:
  postgres-data:      You, last week • Adding another dockerfile compose change
  postgres-data-testing: # Volume for testing instance data
```

all-backend-logs.log | backend-debug.log | deployDB.yml 9+ | app.js | jsonToFile.js

.github > workflows > deployDB.yml

```yaml
jobs:
  transfer-and-deploy:      You, 3 weeks ago • Fixed the unit test and updated the CD workflow
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Transfer code to EC2 instance
        uses: appleboy/scp-action@master
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          source: .
          target: /home/ubuntu/yres/database

      - name: Deploy to EC2 instance
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          script: |
            # Change directory to the database directory.
            cd /home/ubuntu/yres/database/deliverables/yres_scheduler/yres_scheduler_database/
            # Stop and remove any existing Docker containers.
            sudo docker-compose down --volumes --remove-orphans || true
            # Start the Docker Compose stack in detached mode.
            sudo docker-compose up -d

      - name: Wait for Deployment to Complete
        # Wait for 3 seconds to allow the deployment to complete.
        run: sleep 3s

      - name: Test PostgreSQL Connection
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          script: |
            # Replace the following with your PostgreSQL connection parameters.
            PSQL_HOST=ec2-18-218-217-198.us-east-2.compute.amazonaws.com
            PSQL_PORT=5432
            PSQL_USER=yres
            PSQL_DB=yres_db

            # Test the PostgreSQL connection.
            if PGPASSWORD=${{ secrets.PG_PASSWORD }} psql -h $PSQL_HOST -p $PSQL_PORT -U $PSQL_USER -d $PSQL_DB -c "SELECT 1"; then
              echo "PostgreSQL connection successful!"
            else
              echo "PostgreSQL connection failed."
              exit 1
            fi
```

EC2


Instances (1) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
Instance state = running
Clear filters
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
i-013b96ea9966b5c54
Running
t2.small
2/2 checks passed
No alarms
us-east-2c
ec2-18-218-217-198.us...
18.218.217.198
–

Product Handover

Code | Blame | 6 lines (4 loc) · 290 Bytes | Your organization can pay for GitHub Copilot | Raw

```
1  Notes of Discussion of Project Handover:
2
3  Partner will attain our code by simply forking the repo. In order to deploy
4  their forked project on their own EC2 instance, we will provide in depth deployment instructions, as well as addition resources to help
5  their team get the product going.
```

REFLECTION AND LEARNINGS

REFLECTION

- We learned about working with and designing a product for a prospective “client”

Question regarding Prototype

hugo quan <hugo.quan@upluseducation.ca>
Sun 9/24/2023 2:44 PM

To: Max Xu <max.xu@mail.utoronto.ca>
Cc: Celina Yueh <celina.yueh@upluseducation.ca>

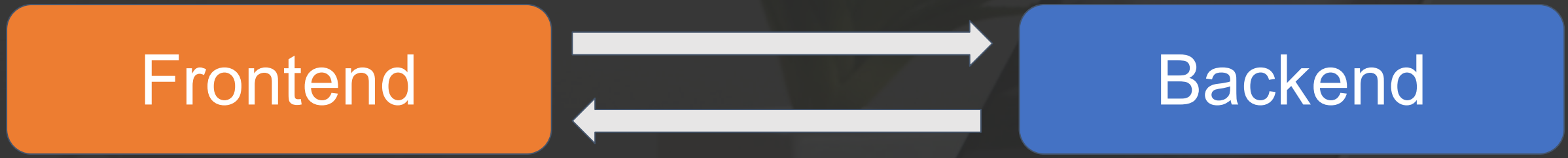
> You don't often get email from hugo.quan@upluseducati

Hey Max it is Hugo Quan here, I was the CO-OP
remembered you have provided the prototype sl
the prototype with the questions so that I could

Kind regards, Hugo Quan
437-989-1833

REFLECTION

- We learned about the importance of communication between the frontend and backend teams

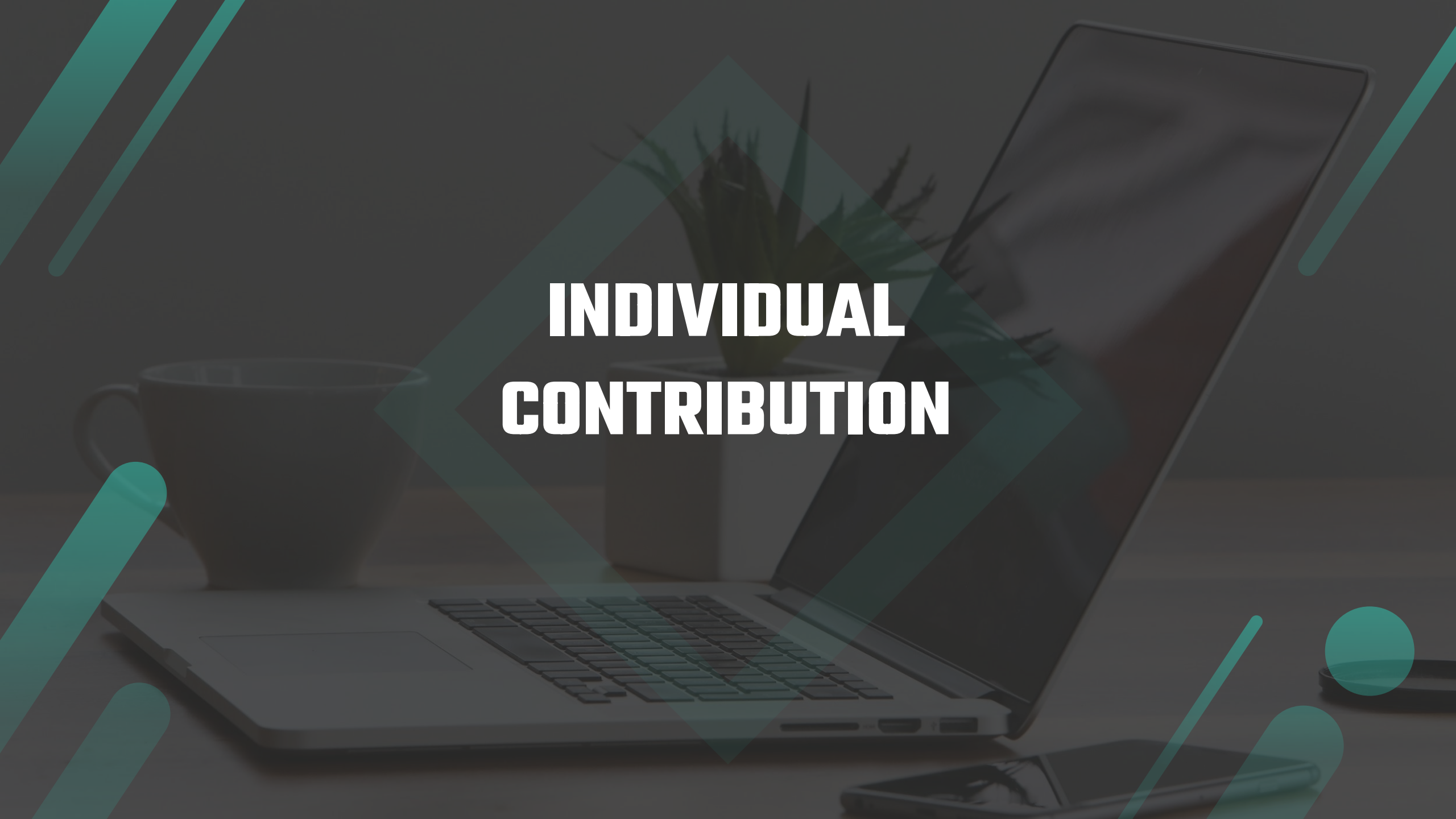


EK Yesterday at 12:00 AM
y'all dont add groups
we do

Max Xu Yesterday at 12:01 AM
Ok Gabriel talked to Harvey and I think that's where this confusion came from

Gabriel Yesterday at 12:02 AM
Yeah, during the update meeting with Harvey on Friday, I was told Backend decided create groups -> user can edit groups -> generate schedule

EK Yesterday at 12:02 AM
nah thats way to complex for tmr



INDIVIDUAL CONTRIBUTION

INDIVIDUAL CONTRIBUTION

- Peifeng Zhang: Backend, designing and testing the grouping and scheduling algorithm.
- Marc Grigoriu: Backend, DB, writing and debugging basic entity interaction functions, writing Postman test flows.
- Harvey Donnelly: Backend, auth middleware/account service, error handling, architecture/planning.
- Max Xu: Frontend, modals for editing and adding, refactoring for integration, error handling, creating reusable frontend components, and frontend styling
- Ewan Jordan: Frontend, error handling, profile page design and logic.
- Gabriel Anover: Frontend, floorplan, schedules timetable, refactoring, frontend organization
- Eric: Backend, CI/CD, Docker, DB, entities, refactoring, deployment on EC2, logging, integrating, scheduling.


PRESENTATION


THANK YOU!