**TT Backend**

db.ts

import mongoose from 'mongoose';

const database = {

    config: ()=>{

        const connectOptions = {

            useNewUrlParser: true,

            useUnifiedTopology: true

        }

        mongoose.connect((process.env.MONGODB\_URI || process.env.DEV\_DB), connectOptions as mongoose.ConnectOptions);

        const db = mongoose.connection;

        // tslint:disable-next-line:no-console

        db.on('error', console.error.bind(console, 'connection error:'));

        db.once('open', () => {

            // tslint:disable-next-line:no-console

            console.log('db successful connection');

        });

    }

}

export default database;

**group.ts**

import { NotFoundError } from '../errors/NotFoundError';

import mongoose from 'mongoose';

import { ExistingDocumentError } from '../errors/ExistingDocumentError';

import { BaseError } from '../errors/BaseError';

import { ProfessorModel } from '../models/Professor';

import { GroupModel, StudentModel } from '../models/GroupStudent';

import { NotUpdatedError } from '../errors/NotUpdatedError';

import StudentController from './student';

import { Group } from '../models/Group';

import ProfessorController from './professor';

const groupQuery = '\_id groupName groupCode grade studentsCount students';

const GroupController = {

    async add(groupName: string, grade: number, professorID: string) {

        const existingGroup = await ProfessorModel.findOne({ \_id: professorID, "groups.groupName": groupName });

        if (existingGroup) throw new ExistingDocumentError('Group');

        const newGroup = {

            groupName,

            grade,

            groupCode: 10

        }

        const group = await new GroupModel(newGroup).save();

        const groupData = {

            groupID: group.\_id,

            groupName: group.groupName,

            groupCode: group.groupCode,

            grade: group.grade,

            studentsCount: group.studentsCount

        }

        const pushGroup = await ProfessorModel.updateOne(

            { \_id: professorID },

            { $push: { groups: groupData } }

        );

        if (pushGroup) return group;

        throw new NotFoundError("Professor");

    },

    async getGroupById(id: string) {

        const group = await GroupModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, groupQuery);

        if (group) return group;

        throw new NotFoundError("Group");

    },

    async getGroupByCode(code: number) {

        const group = await GroupModel.findOne({ groupCode: code }, groupQuery);

        if (group) return group;

        throw new NotFoundError("Group");

    },

    isInGroup(group: Group, studentId: string) {

        const studentIndex = group.students.findIndex(student => student.studentID.toString() === studentId);

        return studentIndex !== -1;

    },

    async joinGroup(groupCode: number, idStudent: string) {

        const group = await GroupController.getGroupByCode(groupCode);

        const student = await StudentController.getStudentById(idStudent);

        if (GroupController.isInGroup(group, idStudent))

            throw new BaseError(409, 'Student is already in group');

        if (group && student) {

            const groupStudentData = {

                name: student.username,

                avatarID: student.avatarID,

                studentID: student.\_id,

                highscores: student.highscores

            }

            const pushGroup = await GroupModel.updateOne(

                { \_id: group.id },

                {

                    $push: {

                        ['students']: groupStudentData

                    },

                    $inc: { studentsCount: 1 }

                }

            )

            const groupData = {

                groupName: group.groupName,

                groupID: group.id

            }

            const pushStudent = await StudentModel.updateOne(

                { \_id: student.id },

                {

                    $set: {

                        'group': groupData

                    }

                }

            )

            const professor = await ProfessorModel.findOne({ 'groups.groupCode': groupCode }, '\_id groups');

            const groupIndex = professor.groups.findIndex(groupItem => groupItem.groupCode === group.groupCode);

            professor.groups[groupIndex].studentsCount += 1;

            const pushProfessor = await ProfessorModel.updateOne(

                { \_id: professor.id },

                professor,

            )

            const updatedGroupData = await GroupController.getGroupByCode(groupCode);

            if (pushGroup.modifiedCount > 0 && pushStudent.modifiedCount > 0 && pushProfessor.modifiedCount > 0)

                return updatedGroupData;

            throw new NotUpdatedError("Group");

        }

        throw new NotFoundError("Group or Student");

    },

    async leaveGroup(idStudent: string) {

        const student = await StudentController.getStudentById(idStudent);

        // if (!GroupController.isInGroup(group, idStudent)) throw new BaseError(409, 'Student is not in group');

        if (student && student.group.groupID) {

            const group = await GroupController.getGroupById(student.group.groupID.toString())

            //remove student from group

            const pullGroup = await GroupModel.updateOne(

                { \_id: group.\_id },

                {

                    $pull: {

                        students: { studentID: idStudent }

                    },

                    $inc: { studentsCount: -1 }

                }

            )

            //remove student count from professor's group

            const professor = await ProfessorModel.findOne({ 'groups.groupCode': group.groupCode }, '\_id groups');

            const groupIndex = professor.groups.findIndex(groupItem => groupItem.groupCode === group.groupCode);

            professor.groups[groupIndex].studentsCount -= 1;

            const pullProfessor = await ProfessorModel.updateOne(

                { \_id: professor.id },

                professor,

            )

            //remove group from student

            const pullStudent = await StudentModel.updateOne(

                { \_id: student.id },

                {

                    $set: {

                        group: {}

                    }

                }

            )

            // const updatedGroupData = await GroupController.getGroupById(student.group.groupID.toString());

            if (pullGroup.modifiedCount > 0 && pullStudent.modifiedCount > 0 && pullProfessor.modifiedCount > 0)

                return 'Success';

            throw new NotUpdatedError("Group");

        }

        throw new NotFoundError("Group or Student");

    },

    async deleteGroup(idProfessor: string, groupCode: number) {

        const groups = await ProfessorController.getGroups(idProfessor)

        const group = await GroupController.getGroupByCode(groupCode)

        const students = group.students

        const inProfessor = groups.findIndex(groupAux => groupAux.groupCode === groupCode)

        if (inProfessor !== -1) {

            //delete group from groups

            const deleteGroup = await GroupModel.deleteOne(

                { 'groupCode': groupCode }

            )

            //delete group from professor

            const pullProfessor = await ProfessorModel.updateOne(

                { \_id: new mongoose.Types.ObjectId(idProfessor) },

                {

                    $pull: {

                        groups: { groupCode: groupCode }

                    }

                }

            )

            //delete group from students

            for (const student of students) {

                const pullStudent = await StudentModel.updateOne(

                    { \_id: student.studentID },

                    {

                        $set: {

                            group: {}

                        }

                    }

                )

                if (pullStudent.modifiedCount > 0) console.log('Delete group from student');

            }

            const professor = await ProfessorModel.findOne({ \_id: new mongoose.Types.ObjectId(idProfessor) }, '\_id groups');

            if (deleteGroup.deletedCount > 0 && pullProfessor.modifiedCount > 0)

                return professor.groups

        }

        throw new NotFoundError("Group");

    }

};

export default GroupController;

**professor.ts**

import { NotFoundError } from '../errors/NotFoundError';

import mongoose from 'mongoose';

import { Encrypt } from "../util/Encrypt";

import { ExistingDocumentError } from '../errors/ExistingDocumentError';

import { AuthenticationError } from '../errors/AuthenticationError';

import jwt from 'jsonwebtoken';

import { ProfessorModel } from '../models/Professor';

import GroupController from './group';

const professorQuery = '\_id name email groups';

const ProfessorController = {

    async add(name: string, email: string, password: string) {

        const existingProfessor = await ProfessorModel.findOne({ email });

        if (existingProfessor) throw new ExistingDocumentError('Professor');

        const newProfessor = {

            name,

            email,

            password: await Encrypt.cryptPassword(password),

        }

        let professor = await new ProfessorModel(newProfessor).save();

        professor = await ProfessorController.getProfessorById(professor.\_id);

        return professor;

    },

    async getProfessorById(id: string) {

        const professor = await ProfessorModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, professorQuery);

        if (professor) return professor;

        throw new NotFoundError("Professor");

    },

    async isProfessor(id: string) {

        const professor = await ProfessorModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, professorQuery);

        if (professor) return true;

        return false;

    },

    async login(email: string, password: string) {

        const professor = await ProfessorModel.findOne({ email });

        if (!professor) throw new NotFoundError("Professor");

        const validPassword = await Encrypt.comparePassword(password, professor.password);

        if (!validPassword) throw new AuthenticationError();

        const professorData = await ProfessorController.getProfessorById(professor.\_id);

        const token = jwt.sign({

            email: professor.email,

            \_id: professor.\_id

        }, process.env.TOKEN)

        return {

            token,

            professorData

        };

    },

    async getGroups(id: string) {

        const professor = await ProfessorModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, professorQuery);

        const groups = [];

        for (const group of professor.groups) {

            const groupData = await GroupController.getGroupById(group.groupID.toString());

            groups.push(groupData);

        }

        if (professor) return groups;

        throw new NotFoundError("Professor");

    },

};

export default ProfessorController;

**score.ts**

import { mongoose } from "@typegoose/typegoose";

import { NotFoundError } from "../errors/NotFoundError";

import { NotUpdatedError } from "../errors/NotUpdatedError";

import { GroupModel, StudentModel } from "../models/GroupStudent";

import { ScoreModel } from "../models/Score"

const ScoreController = {

    async add(score: number, gameNumber: number, gradeNumber: number, studentID: string) {

        const student = await StudentModel.findById({ \_id: studentID })

        const scoreData = {

            score,

            date: new Date()

        }

        //push score

        const query = `game${gameNumber}.grade${gradeNumber}`

        const pushScore = await ScoreModel.updateOne(

            { \_id: student.scoreID },

            { $push: { [query]: scoreData } }

        );

        //push highscore to student

        const queryHighScore = `highscores.game${gameNumber}.grade${gradeNumber}`

        const pushHighScore = await StudentModel.updateOne(

            { \_id: studentID },

            { $max: { [queryHighScore]: score } },

        );

        if (pushHighScore.modifiedCount > 0) console.log('Update Highscore in Student');

        //push highscore to group

        const queryGroupHighScore = `students.$.highscores.game${gameNumber}.grade${gradeNumber}`

        const groupID = student.group.groupID

        const pushGroup = await GroupModel.updateOne(

            {

                \_id: groupID,

                students: {

                    $elemMatch: {

                        studentID: studentID

                    }

                }

            },

            { $max: { [queryGroupHighScore]: score } }

        )

        if (pushGroup.modifiedCount > 0) console.log('Update Highscore in Group');

        const scoreObject = await ScoreModel.findById({ \_id: student.scoreID })

        if (pushScore.modifiedCount > 0) return scoreObject;

        throw new NotUpdatedError("Score");

    },

    async getScoreById(idScore: string) {

        const score = await ScoreModel.findOne({ \_id: new mongoose.Types.ObjectId(idScore) },)

        if (score) return score

        throw new NotFoundError("Score");

    },

    async resetScore(idStudent: string) {

        const student = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(idStudent) }, 'scoreID group');

        if (student.scoreID) {

            //reset score in Scores

            const score = new ScoreModel()

            const pullScore = await ScoreModel.updateOne(

                { \_id: student.scoreID },

                {

                    $set: {

                        game1: score.game1,

                        game2: score.game2,

                        game3: score.game3

                    }

                }

            )

            //reset highscore in student

            const pullStudent = await StudentModel.updateOne(

                { \_id: new mongoose.Types.ObjectId(idStudent) },

                {

                    $set: {

                        highscores: {}

                    }

                }

            )

            //reset highscores in group

            const groupID = student.group.groupID

            const pullGroup = await GroupModel.updateOne(

                {

                    \_id: groupID,

                    students: {

                        $elemMatch: {

                            studentID: idStudent

                        }

                    }

                },

                {

                    'students.$.highscores': {}

                }

            )

            const updatedScore = await ScoreModel.findOne({ \_id: student.scoreID })

            if (pullScore.modifiedCount > 0 && pullStudent.modifiedCount > 0 && pullGroup.modifiedCount > 0)

                return updatedScore;

            throw new NotUpdatedError("Score");

        }

        throw new NotFoundError("Score");

    }

}

export default ScoreController;

**student.ts**

import { NotFoundError } from '../errors/NotFoundError';

import mongoose from 'mongoose';

import { ScoreModel } from "../models/Score";

import { GroupModel, StudentModel } from "../models/GroupStudent";

import { Encrypt } from "../util/Encrypt";

import { ExistingDocumentError } from '../errors/ExistingDocumentError';

import { AuthenticationError } from '../errors/AuthenticationError';

import jwt from 'jsonwebtoken';

import { GroupData, Highscores } from '../models/Student';

import { NotUpdatedError } from '../errors/NotUpdatedError';

const studentQuery = '\_id name username avatarID group scoreID highscores';

const StudentController = {

    async add(username: string, password: string) {

        const existingStudent = await StudentModel.findOne({ username });

        if (existingStudent) throw new ExistingDocumentError('Student');

        const score = await new ScoreModel().save();

        const group = new GroupData()

        const highscores = new Highscores()

        const newStudent = {

            username,

            password: await Encrypt.cryptPassword(password),

            avatarID: 0,

            scoreID: score.\_id,

            group,

            highscores

        }

        let student = await new StudentModel(newStudent).save();

        student = await StudentController.getStudentById(student.\_id);

        return student;

    },

    async getStudentById(id: string) {

        const student = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, studentQuery);

        if (student) return student;

        throw new NotFoundError("Student");

    },

    async isStudent(id: string) {

        const student = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(id) }, studentQuery);

        if (student) return true;

        return false;

    },

    async login(username: string, password: string) {

        const student = await StudentModel.findOne({ username });

        if (!student) throw new NotFoundError("Student");

        const validPassword = await Encrypt.comparePassword(password, student.password);

        if (!validPassword) throw new AuthenticationError();

        const studentData = await StudentController.getStudentById(student.\_id);

        const token = jwt.sign({

            username: student.username,

            \_id: student.\_id

        }, process.env.TOKEN)

        return {

            token,

            studentData

        };

    },

    async getScores(idStudent: string) {

        const student = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(idStudent) }, 'scoreID');

        if (student) {

            const score = await ScoreModel.findOne({ \_id: student.scoreID })

            if (score) return score

        }

        throw new NotFoundError("Scores");

    },

    async updateAvatar(idAvatar: number, idStudent: string) {

        const student = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(idStudent) }, 'group');

        if (student.group == undefined) throw new NotUpdatedError("Avatar");

        if (idAvatar >= 0 && idAvatar <= 7) {

            //update avatar in student

            const pullStudent = await StudentModel.updateOne(

                { \_id: new mongoose.Types.ObjectId(idStudent) },

                {

                    $set: {

                        avatarID: idAvatar

                    }

                }

            )

            //update avatar in student group

            const pullStudentGroup = await GroupModel.updateOne(

                {

                    \_id: student.group.groupID,

                    students: {

                        $elemMatch: {

                            studentID: student.\_id

                        }

                    }

                },

                {

                    'students.$.avatarID': idAvatar

                }

            )

            const updateStudent = await StudentModel.findOne({ \_id: new mongoose.Types.ObjectId(idStudent) })

            if (pullStudent.modifiedCount > 0 && pullStudentGroup.modifiedCount > 0) return updateStudent;

            throw new NotUpdatedError("Avatar");

        }

        throw new NotUpdatedError("Avatar");

    }

};

export default StudentController;

**AuthenticationError**.ts

import { BaseError } from "./BaseError";

export class AuthenticationError extends BaseError {

    constructor() {

      super(401, `Auth failed`);

    }

}

**BaseError.ts**

export class BaseError extends Error {

    statusCode: number;

    constructor(statusCode: number, message: string) {

      super(message);

      Object.setPrototypeOf(this, new.target.prototype);

      this.name = Error.name;

      this.statusCode = statusCode;

      Error.captureStackTrace(this);

    }

  }

**ExistingDocumentError.ts**

import { BaseError } from "./BaseError";

export class ExistingDocumentError extends BaseError {

    documentName: string;

    constructor(documentName: string) {

      super(409, `${documentName} already exist`);

      this.documentName = documentName;

    }

  }

**NotFoundError.ts**

import { BaseError } from "./BaseError";

export class NotFoundError extends BaseError {

    propertyName: string;

    constructor(propertyName: string) {

      super(404, `${propertyName} not found.`);

      this.propertyName = propertyName;

    }

  }

**NotUpdatedError.ts**

import { BaseError } from "./BaseError";

export class NotUpdatedError extends BaseError {

    propertyName: string;

    constructor(propertyName: string) {

        super(404, `${propertyName} not updated.`);

        this.propertyName = propertyName;

    }

}

**group.ts**

import express from 'express';

import { Request, Response, NextFunction } from "express";

import asyncErrorHandler from '../util/asyncErrorHandler';

import GroupController from '../controllers/group';

import jwt from 'jsonwebtoken';

import ProfessorController from '../controllers/professor';

import StudentController from '../controllers/student';

const router = express.Router();

// Add Group

router.post('/', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.groupName)

        return res.status(400).json("Missing group name");

    if (!data.grade)

        return res.status(400).json("Missing group grade");

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const professorData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isProfessor = await ProfessorController.isProfessor(professorData.\_id);

        if (isProfessor) {

            // add group

            const response = await GroupController.add(data.groupName, data.grade, professorData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

// Get Group

router.get('/:id', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.params;

    if (!data)

        return res.status(400).json("Missing params");

    if (!data.id)

        return res.status(400).json("Missing id");

    const response = await GroupController.getGroupById(data.id)

    if (response) return res.status(200).send(response);

}));

//Delete Group

router.post('/delete', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.groupCode)

        return res.status(400).json("Missing groupCode");

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const professorData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isProfessor = await ProfessorController.isProfessor(professorData.\_id);

        if (isProfessor) {

            // delete group

            const response = await GroupController.deleteGroup(professorData.\_id, data.groupCode)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

// Join Group

router.post('/join', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.groupCode)

        return res.status(400).json("Missing groupCode");

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const studentData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isStudent = await StudentController.isStudent(studentData.\_id);

        if (isStudent) {

            // join group

            const response = await GroupController.joinGroup(data.groupCode, studentData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

//Leave Group

router.post('/leave', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const studentData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isStudent = await StudentController.isStudent(studentData.\_id);

        if (isStudent) {

            // leave group

            const response = await GroupController.leaveGroup(studentData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

export default router;

logout.ts

import express from 'express';

import { Request, Response, NextFunction } from "express";

import asyncErrorHandler from '../util/asyncErrorHandler';

const router = express.Router();

//Logout

router.post('/', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Not logged')

        return res.status(200).cookie('token', null, {

            secure: true,

            httpOnly: true,

            maxAge: -1,

            sameSite: 'none'

        }).json('Logged out');

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

export default router;

professor.ts

import express from 'express';

import { Request, Response, NextFunction } from "express";

import asyncErrorHandler from '../util/asyncErrorHandler';

import ProfessorController from '../controllers/professor';

import jwt from 'jsonwebtoken';

const router = express.Router();

// Add Professor

router.post('/', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.name)

        return res.status(400).json("Missing username");

    if (!data.email)

        return res.status(400).json("Missing email");

    if (!data.password)

        return res.status(400).json("Missing password");

    const response = await ProfessorController.add(data.name, data.email, data.password);

    const { token, professorData } = await ProfessorController.login(data.email, data.password);

    if (professorData) return res.status(200).cookie('token', token, {

        secure: true, // set to true if you're using https

        httpOnly: true,

        sameSite: 'none'

    }).send(professorData);

}));

// Login Professor

router.post('/login', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.email)

        return res.status(400).json("Missing email");

    if (!data.password)

        return res.status(400).json("Missing password");

    const { token, professorData } = await ProfessorController.login(data.email, data.password)

    if (professorData) return res.status(200).cookie('token', token, {

        secure: true, // set to true if you're using https

        httpOnly: true,

        sameSite: 'none'

    }).send(professorData);

}));

//Get Professor Groups

router.get('/groups', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const professorData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isProfessor = await ProfessorController.isProfessor(professorData.\_id);

        if (isProfessor) {

            // get groups

            const response = await ProfessorController.getGroups(professorData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

// Get Professor

router.get('/:id', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.params;

    if (!data)

        return res.status(400).json("Missing params");

    if (!data.id)

        return res.status(400).json("Missing id");

    const response = await ProfessorController.getProfessorById(data.id)

    if (response) return res.status(200).send(response);

}));

export default router;

routes.ts

import express from 'express';

import student from './student';

import professor from './professor';

import group from './group';

import score from './score'

import logout from './logout'

const router = express.Router();

router.use('/student', student);

router.use('/professor', professor);

router.use('/group', group);

router.use('/score', score);

router.use('/logout', logout);

export default router

score.ts

import express from 'express';

import { Request, Response, NextFunction } from "express";

import asyncErrorHandler from '../util/asyncErrorHandler';

import jwt from 'jsonwebtoken';

import StudentController from '../controllers/student';

import ScoreController from '../controllers/score';

const router = express.Router();

router.post('/', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.score)

        return res.status(400).json("Missing score");

    if (!data.gameNumber)

        return res.status(400).json("Missing gameNumber");

    if (!data.gradeNumber)

        return res.status(400).json("Missing gradeNumber");

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const studentData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isStudent = await StudentController.isStudent(studentData.\_id);

        if (isStudent) {

            // add score

            const response = await ScoreController.add(data.score, data.gameNumber, data.gradeNumber, studentData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

router.post('/reset', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const token = req.cookies.token || '';

    try {

        if (!token) return res.status(401).json('Login required')

        const studentData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isStudent = await StudentController.isStudent(studentData.\_id);

        if (isStudent) {

            // reset score

            const response = await ScoreController.resetScore(studentData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

export default router;

student.ts

import express from 'express';

import { Request, Response, NextFunction } from "express";

import asyncErrorHandler from '../util/asyncErrorHandler';

import StudentController from '../controllers/student';

import jwt from 'jsonwebtoken';

const router = express.Router();

//Get Student Scores

router.get('/scores/:id', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.params;

    if (!data)

        return res.status(400).json("Missing params");

    if (!data.id)

        return res.status(400).json("Missing id student");

    // get scores

    const response = await StudentController.getScores(data.id)

    if (response) return res.status(200).send(response);

}));

// Get Student

router.get('/:id', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.params;

    if (!data)

        return res.status(400).json("Missing params");

    if (!data.id)

        return res.status(400).json("Missing id");

    const response = await StudentController.getStudentById(data.id);

    if (response) return res.status(200).send(response);

}));

// Add Student

router.post('/', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.username)

        return res.status(400).json("Missing username");

    if (!data.password)

        return res.status(400).json("Missing password");

    const response = await StudentController.add(req.body.username, req.body.password);

    const { token, studentData } = await StudentController.login(data.username, data.password);

    if (studentData) return res.status(200).cookie('token', token, {

        secure: true, // set to true if you're using https

        httpOnly: true,

        sameSite: 'none'

    }).send(studentData);

}));

// Login Student

router.post('/login', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.username)

        return res.status(400).json("Missing username");

    if (!data.password)

        return res.status(400).json("Missing password");

    const { token, studentData } = await StudentController.login(data.username, data.password)

    if (studentData) return res.status(200).cookie('token', token, {

        secure: true, // set to true if you're using https

        httpOnly: true,

        sameSite: 'none'

    }).send(studentData);

}));

// Update Student Avatar

router.post('/avatar', asyncErrorHandler(async (req: Request, res: Response, next: NextFunction) => {

    const data = req.body;

    const token = req.cookies.token || '';

    if (!data)

        return res.status(400).json("Missing request body");

    if (!data.avatarID)

        return res.status(400).json("Missing avatarID");

    try {

        if (!token) return res.status(401).json('Login required')

        const studentData: any = jwt.verify(token, process.env.TOKEN);

        // verify role

        const isStudent = await StudentController.isStudent(studentData.\_id);

        if (isStudent) {

            // update avatar

            const response = await StudentController.updateAvatar(data.avatarID, studentData.\_id)

            if (response) return res.status(200).send(response);

        } else {

            return res.status(403).json('Denied Access');

        }

    } catch (err) {

        return res.status(500).json(err.toString());

    }

}));

export default router;

TT Frontend

[id].tsx

import { NextPage, NextPageContext } from 'next';

import React, { useState } from 'react';

import 'animate.css';

import AuthLayout from '../../components/Layout/AuthLayout';

import axios from 'axios';

import { GameScore, Group } from '../../types';

import LoadingPage from '../../components/Layout/LoadingPage';

import ReturnButton from '../../components/General/ReturnButton';

import PageTitle from '../../components/General/PageTitle';

import { Icon } from '@iconify/react';

import { toast } from 'react-toastify';

import StudentInfoRow from '../../components/Group/UserInfoRow';

import Select from '../../components/General/Select';

import Input from '../../components/General/Input';

import StudentInfoCard from '../../components/Group/UserInfoCard';

type PageProps = {

  group: Group

}

const GroupPage: NextPage<PageProps> = ({ group }: PageProps) => {

  const [selectedFilter, setSelectedFilter] = useState(0);

  // const filterOptions = ['Score Promediado', 'Carreritas', 'Chocolatoso', 'Espiritus Chocarreros'];

  const filterOptions = ['Carreritas', 'Chocolatoso', 'Espiritus Chocarreros', 'Pizzeria'];

  const [busqueda, setBusqueda] = useState("")

  if (!group) {

    return <LoadingPage />

  }

  const copyID = (e: any) => {

    e.preventDefault();

    if (group) {

      navigator.clipboard.writeText(group.groupCode + "");

      toast.success("ID Copiado!");

    }

  };

  const gradeGroup = "grade" + group.grade?.toString() as keyof GameScore

  const renderFilter = () => {

    if (selectedFilter == 0) {

      group.students?.sort((a: any, b: any) => {

        if (a.highscores.game1 && b.highscores.game1)

          return b.highscores.game1[gradeGroup] - a.highscores.game1[gradeGroup]

        else return -1

      })

    } else if (selectedFilter == 1) {

      group.students?.sort((a: any, b: any) => {

        if (a.highscores.game2 && b.highscores.game2)

          return b.highscores.game2[gradeGroup] - a.highscores.game2[gradeGroup]

        else return -1

      })

    } else if (selectedFilter == 2) {

      group.students?.sort((a: any, b: any) => {

        if (a.highscores.game3 && b.highscores.game3)

          return b.highscores.game3[gradeGroup] - a.highscores.game3[gradeGroup]

        else return 0

      })

    } else if (selectedFilter == 3) {

      group.students?.sort((a: any, b: any) => {

        if (a.highscores.game4 && b.highscores.game4)

          return b.highscores.game4[gradeGroup] - a.highscores.game4[gradeGroup]

        else return 0

      })

    }

    return (

      busqueda == "" ?

        group.students?.map((student, i) => (

          <StudentInfoRow student={student} position={i} key={student.studentID} grade={group.grade} />

        ))

        :

        group.students?.filter(student => student.name?.includes(busqueda)).map((student, i) => (

          <StudentInfoRow student={student} position={i} key={student.studentID} grade={group.grade} />

        ))

    )

  }

  return (

    <AuthLayout>

      <div className='flex flex-col bg-matiliztli bg-center bg-cover text-white h-screen w-full overflow-y-scroll'>

        <div className='mx-6 flex flex-col items-start text-lg

        lg:mx-56 md:mx-40 lg:max-w-full'>

          <ReturnButton />

          <PageTitle>

            {group.groupName}

          </PageTitle>

          <div className='flex flex-col space-y-1'>

            <button

              className='flex flex-row items-center space-x-3 text-2xl -mt-3 mb-2'

              onClick={copyID}

            >

              <span>

                Código: {group.groupCode}

              </span>

              <div className='hover:scale-110 transition-transform'>

                <Icon icon={'akar-icons:copy'} width={25} />

              </div>

            </button>

            <span>

              Grado: {group.grade}°

            </span>

            <span>

              Estudiantes: {group.studentsCount}

            </span>

            <span>

              ID: {group.groupID}

            </span>

          </div>

          <div className='flex flex-col w-full mt-2

          lg:justify-betweeen lg:flex-row'>

            <div className='flex items-center space-x-3 '>

              <Input placeholder='Nombre de alumno' value={busqueda} onChange={setBusqueda} type='text' className='my-3' />

              <Icon icon={'akar-icons:search'} width={40} />

            </div>

            <div className='w-full flex flex-row space-x-4 justify-end items-center'>

              <span>

                Ordenar por:

              </span>

              <Select

                className='w-60'

                selectedOption={selectedFilter}

                options={filterOptions}

                onChangeSelection={setSelectedFilter}

                defaultText='Score Promedio'

              />

            </div>

          </div>

          <div className='w-full mt-10 hidden  lg:flex'>

            <table className='w-full'>

              <tbody className='rounded-full overflow-hidden w-full'>

                <tr>

                  <th className='font-normal'>Puesto</th>

                  <th className='font-normal'>Avatar</th>

                  <th className='font-normal'>Nombre</th>

                  <th className='font-normal'>Carreritas</th>

                  <th className='font-normal'>Chocolatoso</th>

                  <th className='font-normal'>Espiritus Chocarreros</th>

                  <th className='font-normal pl-2'>Pizzeria</th>

                </tr>

                {renderFilter()}

              </tbody>

            </table>

          </div>

          <div className='w-full mt-10 flex flex-col lg:hidden'>

            {group.students?.map((student, i) => (

              <StudentInfoCard position={i} student={student} key={student.studentID} />

            ))}

          </div>

        </div>

      </div>

    </AuthLayout>

  )

}

/\*

 hacer fetch de la info del grupo con la API en un useEffect

\*/

// This gets called on every request

export async function getServerSideProps(context: NextPageContext) {

  const groupID = context.query.id as string;

  const API\_URL = process.env.NEXT\_PUBLIC\_API\_URL;

  const getGroup = async (id: string) => {

    return axios.get(`${API\_URL}/group/${id}`, {

      withCredentials: true

    }).then((res) => {

      const data = res.data;

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const group = await getGroup(groupID);

  group.groupID = group.\_id;

  // Pass data to the page via props

  return { props: { group } }

}

export default GroupPage;

[id].tsx //StudentPage

import { NextPage, NextPageContext } from 'next';

import React, { useEffect, useState } from 'react';

import 'animate.css';

import AuthLayout from '../../components/Layout/AuthLayout';

import axios from 'axios';

import { Grade, Group, Score, Scores, Student } from '../../types';

import LoadingPage from '../../components/Layout/LoadingPage';

import ReturnButton from '../../components/General/ReturnButton';

import PageTitle from '../../components/General/PageTitle';

import { Icon } from '@iconify/react';

import Image from 'next/image';

import { URLSearchParams } from 'url';

import ChartScores from '../../components/General/ChartScores';

import Select from '../../components/General/Select';

type PageProps = {

  student: Student

  group: Group

  scores: Scores

}

const StudentPage: NextPage<PageProps> = ({ student, group, scores }: PageProps) => {

  // console.log(scores);

  const [gradeStudent, setGradeStudent] = useState<Grade>("grade4")

  const [selectedGrade, setSelectedGrade] = useState(0)

  const filterOptions = ['4° Grado', '5° Grado', '6° Grado'];

  useEffect(() => {

    if (group.grade == 4) {

      setGradeStudent("grade4")

      setSelectedGrade(0)

    } else if (group.grade == 5) {

      setGradeStudent("grade5")

      setSelectedGrade(1)

    } else if (group.grade == 6) {

      setGradeStudent("grade6")

      setSelectedGrade(2)

    }

  }, [group.grade])

  useEffect(() => {

    if (selectedGrade == 0) setGradeStudent("grade4")

    if (selectedGrade == 1) setGradeStudent("grade5")

    if (selectedGrade == 2) setGradeStudent("grade6")

  }, [selectedGrade])

  const copyID = (e: any) => {

    /\*

      e.preventDefault();

      if (group) {

        navigator.clipboard.writeText(group.groupCode+"");

        toast.success("ID Copiado!");

      }

    \*/

  };

  const background = [

    'bg-table-purple-1',

    'bg-table-purple-2',

    'bg-table-purple-3'

  ];

  const convertDate = (timeStr: string) => {

    //timeStr = "2010-01-13T18:31:16Z";

    var date = new Date(timeStr);

    var day = date.getDate();

    var year = date.getFullYear();

    var month = date.getMonth() + 1;

    var dateStr = day + "/" + month + "/" + year;

    return dateStr;

  };

  if (!student) {

    return <LoadingPage />

  }

  if (!group) {

    return <LoadingPage />

  }

  if (!scores) {

    return <LoadingPage />

  }

  return (

    <AuthLayout>

      <div className='flex flex-col bg-matiliztli bg-center bg-cover text-white h-screen w-full overflow-y-scroll'>

        <div className='mx-6 flex flex-col items-start text-lg

        lg:mx-56 md:mx-40 lg:max-w-full'>

          <ReturnButton />

          <div className='rounded-full overflow-hidden w-32 h-32 relative mt-5'>

            <Image alt='avatar'

              src={`/static/avatars/${student.avatarID}.jpg`}

              layout='fill'

              objectFit='contain'>

            </Image>

          </div>

          <PageTitle>

            {student.username}

          </PageTitle>

          <div className='flex flex-col space-y-3'>

            <button

              className='flex flex-row items-center space-x-3 text-2xl -mt-3 mb-2'

              onClick={copyID}

            >

              <span>

                Código grupo: {group.groupCode}

              </span>

              <div className='hover:scale-110 transition-transform'>

                <Icon icon={'akar-icons:copy'} width={25} />

              </div>

            </button>

            <span>

              Grupo: {student.group?.groupName || '-'}

            </span>

            <span>

              Grado: {group.grade}°

            </span>

            <span>

              ID del alumno: {student.studentID}

            </span>

            <div className='w-full flex flex-row space-x-4 items-center'>

              <span>

                Grado:

              </span>

              <Select

                className='w-60'

                selectedOption={selectedGrade}

                options={filterOptions}

                onChangeSelection={setSelectedGrade}

                defaultText='Score Promedio'

              />

            </div>

          </div>

          <h1 className='mt-5 mb-3 font-bold text-xl'>

            Carreritas

          </h1>

          <div className='flex flex-col w-full lg:flex-row mb-10'>

            <table className='w-full'>

              <tbody className='rounded-full overflow-hidden w-full'>

                <tr>

                  <th className='font-normal'>Score</th>

                  <th className='font-normal'>Fecha</th>

                </tr>

                {

                  scores.game1[gradeStudent].reverse().map((score: any, i: number) => (

                    <tr className={`${i <= 2 ? background[i] : 'bg-white bg-opacity-40'} w-full

                      cursor-pointer hover:bg-main-dark transition-colors duration-300`}

                      key={i}>

                      <th className='py-4'>{score.score}</th>

                      <th>{convertDate(score.date)}</th>

                    </tr>

                  ))

                }

              </tbody>

            </table>

            <div className='w-full flex justify-center mt-7 lg:mx-5'>

              <ChartScores scores={scores.game1[gradeStudent].reverse().map((score: Score) => ({

                date: convertDate(score.date as string),

                score: score.score

              }))} />

            </div>

          </div>

          <h1 className='mt-5 mb-3 font-bold text-xl'>

            Chocolatoso

          </h1>

          <div className='flex flex-col w-full lg:flex-row mb-10'>

            <table className='w-full'>

              <tbody className='rounded-full overflow-hidden w-full'>

                <tr>

                  <th className='font-normal'>Score</th>

                  <th className='font-normal'>Fecha</th>

                </tr>

                {

                  scores.game2[gradeStudent].reverse().map((score: any, i: number) => (

                    <tr className={`${i <= 2 ? background[i] : 'bg-white bg-opacity-40'} w-full

                      cursor-pointer hover:bg-main-dark transition-colors duration-300`}

                      key={i}>

                      <th className='py-4'>{score.score}</th>

                      <th>{convertDate(score.date)}</th>

                    </tr>

                  ))

                }

              </tbody>

            </table>

            <div className='w-full flex justify-center mt-7 lg:mx-5'>

              <ChartScores scores={scores.game2[gradeStudent].reverse().map((score: Score) => ({

                date: convertDate(score.date as string),

                score: score.score

              }))} />

            </div>

          </div>

          <h1 className='mt-5 mb-3 font-bold text-xl'>

            Espíritus Chocarreros

          </h1>

          <div className='flex flex-col w-full lg:flex-row mb-10'>

            <table className='w-full'>

              <tbody className='rounded-full overflow-hidden w-full'>

                <tr>

                  <th className='font-normal'>Score</th>

                  <th className='font-normal'>Fecha</th>

                </tr>

                {

                  scores.game3[gradeStudent].reverse().map((score: any, i: number) => (

                    <tr className={`${i <= 2 ? background[i] : 'bg-white bg-opacity-40'} w-full

                      cursor-pointer hover:bg-main-dark transition-colors duration-300`}

                      key={i}>

                      <th className='py-4'>{score.score}</th>

                      <th>{convertDate(score.date)}</th>

                    </tr>

                  ))

                }

              </tbody>

            </table>

            <div className='w-full flex justify-center mt-7 lg:mx-5'>

              <ChartScores scores={scores.game3[gradeStudent].reverse().map((score: Score) => ({

                date: convertDate(score.date as string),

                score: score.score

              }))} />

            </div>

          </div>

          <h1 className='mt-5 mb-3 font-bold text-xl'>

            Pizzeria

          </h1>

          <div className='flex flex-col w-full lg:flex-row mb-10'>

            <table className='w-full'>

              <tbody className='rounded-full overflow-hidden w-full'>

                <tr>

                  <th className='font-normal'>Score</th>

                  <th className='font-normal'>Fecha</th>

                </tr>

                {

                  scores.game4[gradeStudent].reverse().map((score: any, i: number) => (

                    <tr className={`${i <= 2 ? background[i] : 'bg-white bg-opacity-40'} w-full

                      cursor-pointer hover:bg-main-dark transition-colors duration-300`}

                      key={i}>

                      <th className='py-4'>{score.score}</th>

                      <th>{convertDate(score.date)}</th>

                    </tr>

                  ))

                }

              </tbody>

            </table>

            <div className='w-full flex justify-center mt-7 lg:mx-5'>

              <ChartScores scores={scores.game4[gradeStudent].reverse().map((score: Score) => ({

                date: convertDate(score.date as string),

                score: score.score

              }))} />

            </div>

          </div>

        </div>

      </div>

    </AuthLayout>

  )

}

/\*

 hacer fetch de la info del grupo con la API en un useEffect

\*/

// This gets called on every request

export async function getServerSideProps(context: NextPageContext) {

  const studentID = context.query.id as string;

  const API\_URL = process.env.NEXT\_PUBLIC\_API\_URL;

  const getStudentInfo = async (id: string) => {

    return axios.get(`${API\_URL}/student/${id}`).then((res) => {

      const data = res.data;

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const getGroup = async (id: string) => {

    return axios.get(`${API\_URL}/group/${id}`).then((res) => {

      const data = res.data;

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const getScores = async (id: string) => {

    const params = new URLSearchParams();

    return axios.get(`${API\_URL}/student/scores/${id}`).then((res) => {

      const data = res.data;

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const student = await getStudentInfo(studentID).catch((e) => null);

  const group = await getGroup(student.group.groupID).catch((e) => null)

  const scores = await getScores(studentID).catch((e) => null);

  student.studentID = student.\_id;

  // Pass data to the page via props

  return { props: { student, group, scores } }

}

export default StudentPage;

\_app.tsx

import '../styles/globals.css'

import type { AppProps } from 'next/app'

import { ProfessorInfoContextProvider } from '../contexts/ProfessorInfoContext'

function MyApp({ Component, pageProps }: AppProps) {

  return (

    <ProfessorInfoContextProvider>

      <Component {...pageProps} />

    </ProfessorInfoContextProvider>

  )

}

export default MyApp

components.tsx

import type { NextPage } from 'next'

import { useState } from 'react'

import Button from '../components/General/Button'

import Checkbox from '../components/General/Checkbox'

import Dropdown from '../components/General/Dropdown'

import Input from '../components/General/Input'

import PageLayout from '../components/Layout/PageLayout'

import Select from '../components/General/Select'

import CreatedGroupModal from '../components/Modals/CreatedGroupModal'

const ComponentsPage: NextPage = () => {

    const [showModal, setShowModal] = useState(false);

    const [selectedOption, setSelectedOption] = useState(-1);

    const [testValue, setTestValue] = useState('');

    const [selectedCheckbox, setSelectedCheckbox] = useState(false);

  return (

        <PageLayout>

            <div className={"flex flex-col p-5 space-y-4 text-white max-w-xs"}>

                <div>

                    <span>Button</span>

                    <div className='flex flex-col space-y-10 py-10 max-w-lg'>

                        <Button label='Blue Button' />

                        <Button label='Pink Button' color='pink' />

                    </div>

                </div>

                <div>

                    <span>Checkbox</span>

                    <div >

                        <Checkbox onClick={()=>setSelectedCheckbox(!selectedCheckbox)} selected={selectedCheckbox} />

                    </div>

                </div>

                <div>

                    <span>Dropdown</span>

                    <div >

                        <Dropdown title='dropdown title' answer='answer test' />

                    </div>

                </div>

                <div>

                    <span>Input</span>

                    <div className='borde border-white'>

                        <Input value={testValue} onChange={setTestValue} placeholder='adsf' />

                    </div>

                </div>

                <div>

                    <span>Select</span>

                    <div className=''>

                        <Select options={['opt1', 'opt2', 'opt3']} defaultText='Selecciona el grado'

                        onChangeSelection={setSelectedOption} selectedOption={selectedOption}/>

                    </div>

                </div>

                <div>

                    <span>Modal</span>

                    <div className=''>

                        <CreatedGroupModal isOpen={showModal} closeModal={()=>setShowModal(false)} />

                        <Button label='View Modal' onClick={()=>setShowModal(true)} />

                    </div>

                </div>

            </div>

        </PageLayout>

  )

}

export default ComponentsPage

create-account.tsx

import { NextPage } from 'next'

import Image from 'next/image'

import Router from 'next/router'

import React, { useState } from 'react'

import Button from '../components/General/Button'

import Input from '../components/General/Input'

import PageLayout from '../components/Layout/PageLayout'

import 'animate.css';

import useAPI from '../hooks/useAPI'

import { toast } from 'react-toastify'

const CreateAccount: NextPage = () => {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const {

    register

  } = useAPI();

  const login = () =>{

    Router.push('login');

  };

  const registerHandler = async () =>{

    if (name==null || name=='') {

      toast.error('ingresa el nombre')

      return;

    }

    if (email==null || email=='') {

      toast.error('ingresa el email')

      return;

    }

    register(name, email, password).then((data)=>{

      Router.push('my-groups');

    }).catch(e=>{

      toast.error(e);

    });

  };

  return (

  <PageLayout>

    <div className='flex flex-col md:flex-row bg-main-dark h-screen'>

      <div className='w-full flex flex-col items-center justify-center bg-matiliztli bg-center bg-cover text-white h-52

      md:h-screen md:w-1/2 md:justify-start'>

        <div className='w-4/5 h-20 relative

        md:h-28 md:mt-24 md:mb-10

        animate\_\_animated animate\_\_fadeInUp'>

          <Image src={'/static/logo.svg'} alt='Matiliztli logo' layout='fill' objectFit='contain' />

        </div>

        <p className='text-center font-medium px-10

        md:text-2xl md:w-1/2

        animate\_\_animated animate\_\_fadeInUp animate\_\_delay-1s'>

          Regístrate gratis para comenzar a administrar un grupo.

        </p>

      </div>

      <div className='w-full px-10 bg-main-dark h-full text-white flex flex-col items-center text-center

      md:w-1/2'>

        <p className='text-center text-3xl font-bold my-10

        md:text-4xl'>

          Crea tu cuenta

        </p>

        <div className='justify-center w-full

        md:w-2/3'>

          <Input placeholder='Nombre' value={name} onChange={setName} className='my-2'/>

          <Input placeholder='Email' value={email} onChange={setEmail} className='my-2'/>

          <Input placeholder='Contraseña' type='password' value={password} onChange={setPassword} className='my-3'/>

          <Button label='Registrarse' className='my-2'

          onClick={registerHandler}/>

          <p className='text-center font-black underline my-5 hover:cursor-pointer'

          onClick={login}>

            Ya tengo una cuenta

          </p>

        </div>

      </div>

    </div>

  </PageLayout>

  )

}

export default CreateAccount;

create-group.tsx

import {NextPage} from 'next';

import Router from 'next/router';

import React, {useState} from 'react';

import Button from '../components/General/Button';

import 'animate.css';

import Input from '../components/General/Input';

import Select from '../components/General/Select';

import ReturnButton from '../components/General/ReturnButton';

import CreatedGroupModal from '../components/Modals/CreatedGroupModal'

import AuthLayout from '../components/Layout/AuthLayout'

import useAPI from '../hooks/useAPI';

import {toast} from 'react-toastify';

import {Group} from '../types';

import {useProfessorInfo} from '../contexts/ProfessorInfoContext';

const CreateGroup: NextPage = () => {

  const {refreshInfo} = useProfessorInfo();

  const {addGroup} = useAPI();

  const [createdGroup, setCreatedGroup] = useState<Group>();

  const [groupName, setGroupName] = useState('');

  const [grade, setGrade] = useState(0);

  const [showModal, setShowModal] = useState(false);

  const addGroupHandler = ()=>{

    if (grade==0) {

      toast.error("Elige un grado válido");

      return;

    }

    addGroup(groupName, (grade+3).toString()).then((data)=>{

      refreshInfo();

      setCreatedGroup(data as Group);

      setShowModal(true);

      setGroupName('');

      setGrade(0);

    });

  }

  const logout = () =>{

    Router.push('home');

  };

  return (

  <AuthLayout>

    <div className='flex flex-col bg-matiliztli bg-center bg-cover text-white h-screen w-full'>

      <div className='mx-6 flex flex-col items-center

        lg:mx-56 md:mx-40 lg:max-w-full'>

        <ReturnButton href={'/my-groups'} />

        <p className='text-center text-3xl font-bold mb-10 mt-5

        md:text-4xl

        animate\_\_animated animate\_\_fadeInUp'>

          Crear nuevo grupo

        </p>

        <div className='w-full lg:w-1/3'>

          <Input

            value={groupName}

            onChange={setGroupName}

            placeholder={'Nombre o apodo del grupo'}

            className='my-2 '

          />

          <Select

            defaultText='Selecciona el grado'

            className='my-2 '

            options={['Selecciona el grado','4°','5°','6°']}

            selectedOption={grade}

            onChangeSelection={setGrade}

          />

          <CreatedGroupModal

            isOpen={showModal}

            closeModal={()=>setShowModal(false)}

            group={createdGroup}

          />

          <Button label='Crear grupo' className='my-2' onClick={addGroupHandler}/>

          <Button label='Ver mis grupos' className='my-2' color='pink'

            onClick={()=>{

              Router.push('/my-groups');

            }}/>

        </div>

      </div>

    </div>

  </AuthLayout>

  )

}

export default CreateGroup;

home.tsx

import type { NextPage } from 'next'

import PageLayout from '../components/Layout/PageLayout';

import Button from '../components/General/Button';

import Image from 'next/image';

import Router from 'next/router';

import 'animate.css';

import { useProfessorInfo } from '../contexts/ProfessorInfoContext';

const Home: NextPage = () => {

  const { loggedIn } = useProfessorInfo();

  const login = () => {

    Router.push('login');

  };

  const createAccount = () => {

    Router.push('create-account');

  };

  const downloadFile = () => {

    window.open('/static/Matiliztli-Final.apk')

  }

  return (

    <PageLayout>

      <div className='flex flex-col items-center bg-matiliztli bg-center bg-cover text-white h-screen'>

        <div className='w-4/5 h-20 relative

        md:h-28 md:mt-24 md:mb-10

        animate\_\_animated animate\_\_fadeInUp'>

          <Image src={'/static/logo.svg'} alt='Matiliztli logo' layout='fill' objectFit='contain' />

        </div>

        <p className='w-2/3 text-center md:text-2xl md:pb-10

        animate\_\_animated animate\_\_fadeInUp animate\_\_delay-1s'>

          Una forma de apoyo en las matemáticas jugando.

        </p>

        <Button label='¡Descarga la app!' color='pink' className='m-4 w-60

        md:mb-32'

          onClick={downloadFile}

        />

      </div>

      <div className='flex flex-col bg-main-dark

      md:flex-row'>

        <div className='w-full text-white p-8

        md:w-1/2 md:p-14'>

          <h1 className='font-black text-2xl mb-4

          md:text-4xl'>

            ¿Qué es Matiliztli?

          </h1>

          <p className='md:text-lg mb-5'>

            Matiliztli es una app de apoyo que ayuda a reforzar los conocimientos

            de matemáticas, dirigida a estudiantes de 4to a 6to grado de primaria.

          </p>

          <p className='md:text-lg mb-5'>

            Desde esta página web puedes crear un grupo para monitorear el progreso

            de un grupo de alumnos que utilicen esta app.

          </p>

          <div className='flex flex-col items-center

          md:flex-row'>

            {!loggedIn &&

              <>

                <Button label='Registrarse' color='pink' className='m-4'

                  onClick={createAccount} />

                <Button label='Iniciar Sesión' className='m-4'

                  onClick={login} />

              </>

            }

          </div>

        </div>

        <div className='w-full min-h-full md:w-1/2 relative'>

          <Image src={'/static/picture1.png'} alt='Student girl' layout='fill' objectFit='cover' />

        </div>

      </div>

    </PageLayout>

  )

}

export default Home

login.tsx

import { NextPage } from 'next'

import Image from 'next/image'

import Router from 'next/router'

import React, { useState } from 'react'

import Button from '../components/General/Button'

import Input from '../components/General/Input'

import PageLayout from '../components/Layout/PageLayout'

import 'animate.css';

import useAPI from '../hooks/useAPI'

import { toast } from 'react-toastify'

const Login: NextPage = () => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const {

    login

  } = useAPI();

  const createAccount = () =>{

    Router.push('create-account');

  };

  const loginHandler = async () =>{

    if (email==null || email=='') {

      toast.error('ingresa el email')

      return;

    }

    login(email, password).then((data)=>{

      Router.push('my-groups');

    }).catch(e=>{

      toast.error(e);

    });

  };

  return (

  <PageLayout>

    <div className='flex flex-col md:flex-row bg-main-dark h-screen'>

      <div className='w-full flex flex-col items-center justify-center bg-matiliztli bg-center bg-cover text-white h-52

      md:h-screen md:w-1/2 md:justify-start'>

        <div className='w-4/5 h-20 relative

        md:h-28 md:mt-24 md:mb-10

        animate\_\_animated animate\_\_fadeInUp'>

          <Image src={'/static/logo.svg'} alt='Matiliztli logo' layout='fill' objectFit='contain' />

        </div>

        <p className='text-center font-medium px-10

        md:text-2xl md:w-1/2

        animate\_\_animated animate\_\_fadeInUp animate\_\_delay-1s'>

          ¡Bienvenido(a) de nuevo!<br/>Tu grupo te está esperando.

        </p>

      </div>

      <div className='w-full px-10 bg-main-dark h-full text-white flex flex-col items-center text-center

      md:w-1/2'>

        <p className='text-center text-3xl font-bold my-10

        md:text-4xl'>

          Inicia Sesión

        </p>

        <div className='justify-center w-full

        md:w-2/3'>

          <Input placeholder='Email' value={email} onChange={setEmail} className='my-2'/>

          <Input placeholder='Contraseña' type='password' value={password} onChange={setPassword} className='my-3'/>

          <Button label='Iniciar Sesión' className='my-2'

          onClick={loginHandler}/>

          <p className='text-center font-black underline my-5 hover:cursor-pointer'

          onClick={createAccount}>

            Aún no tengo cuenta

          </p>

        </div>

      </div>

    </div>

  </PageLayout>

  )

}

export default Login;

my-groups.tsx

import {NextPage} from 'next'

import Router from 'next/router'

import React, { useState } from 'react'

import Button from '../components/General/Button'

import 'animate.css';

import GroupCard from '../components/General/GroupCard'

import { useProfessorInfo } from '../contexts/ProfessorInfoContext'

import AuthLayout from '../components/Layout/AuthLayout'

import ConfirmModal from '../components/Modals/ConfirmModal';

import useAPI from '../hooks/useAPI';

import { toast } from 'react-toastify';

import { Group } from '../types';

const MyGroups: NextPage = () => {

  const {deleteGroup} = useAPI();

  const [open, setOpen] = useState(false);

  const {name, groups} = useProfessorInfo();

  const [groupToDelete, setGroupToDelete] = useState<Group>();

  const openDeleteModal = (group: Group) => {

    setOpen(true);

    setGroupToDelete(group);

  };

  const handleDeleteGroup = async ()=>{

    console.log(groupToDelete);

    if (groupToDelete) {

      const auxname = groupToDelete.groupName;

      deleteGroup(groupToDelete.groupCode).then((res)=>{

        toast.success(`Grupo ${auxname} eliminado con éxito!`);

        setOpen(false);

      });

    }

  }

  const createGroup = () =>{

    Router.push('create-group');

  };

  return (

  <AuthLayout>

    <ConfirmModal

      isOpen={open}

      onConfirm={handleDeleteGroup}

      closeModal={()=>setOpen(false)}

    >

        <span className='font-bold text-xl mb-3 text-center

        md:text-2xl'>

          {`¿Seguro que quieres borrar el grupo "${groupToDelete?.groupName}"?`}

        </span>

        <p className='text-center mx-2 my-3

        md:mx-10'>

          Borrar el grupo expulsará a todos los estudiantes de el.

          Sin embargo, seguirán conservando el puntaje obtenido en los juegos.

        </p>

    </ConfirmModal>

    <div className='flex flex-col bg-matiliztli bg-center bg-cover text-white h-screen'>

      <div className='mx-6

        lg:mx-56'>

        <div>

          <p className='font-medium text-xl mb-5

          md:text-3xl md:mb-8

          animate\_\_animated animate\_\_fadeInUp'>

            Hola, {name}

          </p>

          <p className='font-bold text-2xl mb-5

          md:text-4xl md:mb-7

          animate\_\_animated animate\_\_fadeInUp animate\_\_delay-1s'>

            Mis grupos

          </p>

          <Button label='+ Crear nuevo grupo' className='w-52 mb-8'

          onClick={createGroup}/>

        </div>

        <div className='flex flex-col items-center overflow-auto h-[30rem] space-y-5

        px-10 py-10

        md:flex-row md:space-y-0 md:flex-wrap md:items-start'>

          {(groups && groups.length>0)?

            groups.map((group, i)=>(

              <GroupCard

                key={group.groupID}

                id={i}

                group={group}

                openModal={()=>openDeleteModal(group)}

              />

            )):

            <span>Sin grupos</span>

          }

        </div>

      </div>

    </div>

  </AuthLayout>

  )

}

export default MyGroups;

useAPI.ts

mport axios from "axios";

import { useProfessorInfo } from "../contexts/ProfessorInfoContext";

const API\_URL = process.env.NEXT\_PUBLIC\_API\_URL;

const axiosInstance = axios.create({

  withCredentials: true

})

export default function useAPI() {

  const { setProfessorData, setGroups, signOut, \_id } = useProfessorInfo();

  const login = async (email: string, password: string) => {

    return axiosInstance.post(`${API\_URL}/professor/login`, {

      email,

      password,

    }).then((res) => {

      const data = res.data;

      setProfessorData(data);

      localStorage.setItem('matiliztli-id', data.\_id);

      return data;

    }).catch((error) => {

      throw error.message;

    });

  };

  const logout = async () => {

    return axiosInstance.post(`${API\_URL}/logout`, {}).then((res) => {

      const data = res.data;

      localStorage.removeItem('matiliztli-id');

      signOut();

      return data;

    }).catch((error) => {

      throw error.message;

    });

  };

  const register = async (

    name: string,

    email: string,

    password: string,

  ) => {

    return axiosInstance.post(`${API\_URL}/professor`, {

      name,

      email,

      password,

    }).then((res) => {

      const data = res.data;

      login(email, password)

      setProfessorData(data);

      return res.data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const getProfessorInfo = async (id?: string) => {

    return axiosInstance.get(`${API\_URL}/professor/${id || \_id}`).then((res) => {

      const data = res.data;

      setProfessorData(data);

      return res.data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const getProfessorGroups = async () => {

    return axiosInstance.get(`${API\_URL}/professor/groups`).then((res) => {

      const data = res.data;

      setGroups(data);

      return res.data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const getGroup = async (id: string) => {

    return axiosInstance.get(`${API\_URL}/group/${id}`).then((res) => {

      const data = res.data;

      return res.data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const addGroup = async (

    groupName: string,

    grade: string,

  ) => {

    return axiosInstance.post(`${API\_URL}/group`, {

      groupName,

      grade,

    }).then((res) => {

      const data = res.data;

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  const deleteGroup = async (

    groupCode: number,

  ) => {

    return axiosInstance.post(`${API\_URL}/group/delete`, {

      groupCode,

    }).then((res) => {

      const data = res.data;

      setGroups(data);

      return data;

    }).catch((error) => {

      throw error.response.data;

    });

  };

  return {

    login,

    logout,

    register,

    getProfessorInfo,

    getProfessorGroups,

    getGroup,

    addGroup,

    deleteGroup,

  };

}

professorInfoContext.tsx

import axios from "axios";

import { createContext, ReactNode, useContext, useEffect, useState } from "react";

import { Group, Professor, ProfessorInfoContextInterface } from "../types";

const ProfessorInfoContext = createContext< ProfessorInfoContextInterface | undefined>(undefined);

const API\_URL = process.env.NEXT\_PUBLIC\_API\_URL;

const ProfessorInfoContextProvider = ({children}: {children: ReactNode}) => {

  const [loggedIn, setLoggedIn] = useState(false);

  const [loggingIn, setLoggingIn] = useState(true);

  const [\_id, setId] = useState<string | undefined>();

  const [name, setName] = useState<string | undefined>();

  const [email, setEmail] = useState<string | undefined>();

  const [groups, setGroups] = useState<Group[] | undefined>();

  const getProfessorInfo = async (id?: string)=>{

    return axios.get(`${API\_URL}/professor/${id || \_id}`).then( (res) => {

      const data = res.data;

      setProfessorData(data);

      return res.data;

    }).catch( (error) => {

      throw error.response.data;

    });

  };

  const setProfessorData = (professor: Professor) => {

    setId(professor.\_id);

    setName(professor.name);

    setEmail(professor.email);

    setGroups(professor.groups);

    setLoggedIn(true);

  };

  const signOut = () => {

    setId(undefined);

    setName(undefined);

    setEmail(undefined);

    setGroups(undefined);

    setLoggedIn(false);

  };

  const refreshInfo = getProfessorInfo;

  useEffect(() => {

    // set info at start

    const id = localStorage.getItem('matiliztli-id');

    if (id) {

      getProfessorInfo(id);

      setLoggedIn(true);

    } else {

      setLoggedIn(false);

    }

    setLoggingIn(false);

  }, []);

  return (

    <ProfessorInfoContext.Provider

      value={{

        refreshInfo,

        loggedIn,

        \_id,

        setId,

        name,

        setName,

        email,

        setEmail,

        groups,

        setGroups,

        setProfessorData,

        signOut

      }}>

      {children}

    </ProfessorInfoContext.Provider>

  );

}

const useProfessorInfo = ()=>{

  const context = useContext(ProfessorInfoContext);

  if (context === undefined) {

    throw new Error('useProfessorInfo must be used within an ProfessorProvider')

  }

  return context;

}

export {useProfessorInfo, ProfessorInfoContext, ProfessorInfoContextProvider};

global.css

@tailwind base;

@tailwind components;

@tailwind utilities;

@import url('https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;700;900&display=swap');

TT App

AccountAPI.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AccountAPI : MonoBehaviour

{

    public MenuUIManager menuUIManager;

    public AvatarsUIManager avatarsUIManager;

    public StudentData student;

    public void OnAddScoreResponse(string res)

    {

        StudentScores studentScores = (StudentScores)JsonUtility.FromJson(res, typeof(StudentScores));

        Debug.Log(studentScores.game2.grade5[0].date);

    }

    public void OnAddScoreError(string res)

    {

        Debug.Log(res);

        if (res[0] == '"') res = res.Substring(1, res.Length - 2);

    }

    public void AddScore(int score, int gameNumber, int gradeNumber)

    {

        WWWForm form = new WWWForm();

        form.AddField("score", score);

        form.AddField("gameNumber", gameNumber);

        form.AddField("gradeNumber", gradeNumber);

        StartCoroutine(API.SendHTTP("score", form, OnAddScoreResponse, OnAddScoreError));

    }

    public void onJoinGroupResponse(string res)

    {

        Debug.Log(res);

        menuUIManager.hideCardjoinGroup();

    }

    public void onJoinGroupError(string res)

    {

        if (res[0] == '"') res = res.Substring(1, res.Length - 2);

        menuUIManager.ShowJoinGroupError(res);

    }

    public void JoinGroup(int groupCode)

    {

        WWWForm form = new WWWForm();

        form.AddField("groupCode", groupCode);

        StartCoroutine(API.SendHTTP("group/join", form, onJoinGroupResponse, onJoinGroupError));

    }

    public void ChangeAvatarID(int newID)

    {

        WWWForm form = new WWWForm();

        form.AddField("avatarID", newID);

        StartCoroutine(API.SendHTTP("student/avatar", form, OnChangeAvatarResponse, OnChangeAvatarError));

    }

    void OnChangeAvatarResponse(string res)

    {

    JsonUtility.FromJsonOverwrite(res, student);

    PersistentDataManager.SaveStudentData(student);

    //student.Init(studentSerializable);

        avatarsUIManager.UpdateAvatar(student.avatarID);

    }

    void OnChangeAvatarError(string res)

    {

        if (res[0] == '"') res = res.Substring(1, res.Length - 2);

        avatarsUIManager.ShowError(res);

    }

}

API.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Networking;

public static class API

{

    // const string baseURL = "http://localhost:5000/";

    const string baseURL = "https://matiliztli-9dxxs.ondigitalocean.app/";

    public static IEnumerator SendHTTP(string route, WWWForm form,

        Action<string> responseCallback,

        Action<string> errorCallback){

        UnityWebRequest request = UnityWebRequest.Post(baseURL+route, form);

        request.uploadHandler.contentType = "application/x-www-form-urlencoded; charset=UTF-8";

        // request.SetRequestHeader("Content-Type", "application/json");

        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.ConnectionError ){

            Debug.Log("Error While Sending: " + request.error);

        }else{

            string data = request.downloadHandler.text;

            Debug.Log("Received: " + data);

            // check if is JSON

            if (data[0]=='{') responseCallback?.Invoke(data);

            else errorCallback?.Invoke(data);

        }

    }

    public static IEnumerator SendHTTPGET(string route,

        Action<string> responseCallback,

        Action<string> errorCallback){

        UnityWebRequest request = UnityWebRequest.Get(baseURL+route);

        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.ConnectionError ){

            Debug.Log("Error While Sending: " + request.error);

        }else{

            string data = request.downloadHandler.text;

            Debug.Log("Received: " + data);

            // check if is JSON

            if (data[0]=='{') responseCallback?.Invoke(data);

            else errorCallback?.Invoke(data);

        }

    }

}

LoginAPI.cs

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class LoginAPI : MonoBehaviour

{

    public LoginUIManager loginUIManager;

    public StudentData student;

    public void OnLogin (string res) {

        JsonUtility.FromJsonOverwrite(res, student);

        PersistentDataManager.SaveStudentData(student);

        SceneManager.LoadScene("MenuScene");

    }

    public void OnLoginError (string res) {

        if (res[0]=='"') res=res.Substring(1, res.Length-2);

        loginUIManager.ShowLoginError(res);

    }

    public void LoginStudent(string username, string password){

        WWWForm form = new WWWForm();

        form.AddField("username", username);

        form.AddField("password", password);

        StartCoroutine(API.SendHTTP("student/login", form, OnLogin, OnLoginError));

    }

    public void GetStudentData(string \_id, Action<StudentData> responseCallback){

        WWWForm form = new WWWForm();

        StartCoroutine(API.SendHTTPGET("student/"+\_id, (res)=>{

            JsonUtility.FromJsonOverwrite(res, student);

            PersistentDataManager.SaveStudentData(student);

            responseCallback(student);

        }, OnLoginError));

    }

}

RegiterAPI.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RegisterAPI : MonoBehaviour

{

    public LoginUIManager loginUIManager;

    public void OnRegister (string res) {

        loginUIManager.GoToLogin();

        loginUIManager.ShowSuccessMessage("Te has registrado! Ahora inicia sesión");

    }

    public void OnRegisterError (string res) {

        if (res[0]=='"') res=res.Substring(1, res.Length-2);

        loginUIManager.ShowRegisterError(res);

    }

    public void RegisterUser(string name, string password){

        WWWForm form = new WWWForm();

        form.AddField("username", name);

        form.AddField("password", password);

        StartCoroutine(API.SendHTTP("student", form, OnRegister, OnRegisterError));

    }

}

AnswerController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class AnswerController : MonoBehaviour

{

    public float speed = 2;

    public int index;

    public Answer answer;

    public bool answerChecked = false;

    public QuestionData questionData;

    public CarreritasGameData gameData;

    Vector3 startPosition;

    [SerializeField] TMP\_Text answerText;

    // Start is called before the first frame update

    void Start()

    {

        CarreritasEventManager.OnQuestionGenerated+=RestartPosition;

    }

    private void OnDestroy() {

        CarreritasEventManager.OnQuestionGenerated-=RestartPosition;

    }

    public void Initialize(int index){

        startPosition=this.transform.position;

        this.index=index;

        this.answer=questionData.answers[index+1];

        answerText.text= this.answer.text;

    }

    public void RestartPosition(){

        this.transform.position = startPosition;

        answer=questionData.answers[index+1];

        answerText.text= this.answer.text;

        answerChecked=false;

        gameObject.SetActive(true);

    }

    // Update is called once per frame

    void Update()

    {

        // Move the object forward along its z axis 1 unit/second.

        if (!gameData.paused && !gameData.finished)

        {

            transform.Translate(Vector3.back \* Time.deltaTime \* speed);

        }

    }

}

AnswerManager.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AnswersManager : MonoBehaviour

{

    public SettingsCarreritas settings;

    public QuestionGenerator questionGenerator;

    public CarreritasGameData gameData;

    public QuestionData questionData;

    List<AnswerController> instancedAnswers = new List<AnswerController>();

    void Start()

    {

        GenerateAnswers();

    }

    void Update() {

        foreach (AnswerController answer in instancedAnswers.ToArray()){

            CheckAnswer(answer);

        }

    }

    // Update is called once per frame

    void CheckAnswer(AnswerController answer){

        if (!answer) return;

        if (answer.transform.position.z<= settings.collisionThreeshold && !answer.answerChecked){

            answer.answerChecked=true;

            if (gameData.playerPositionIndex==answer.index){

                answer.gameObject.SetActive(false);

                bool isCorrect = questionData.answers[answer.index+1].isCorrect;

                CarreritasEventManager.\_OnAnswerQuestion(isCorrect);

            }

        }

    }

    void GenerateAnswers()

    {

        for (int i = 0; i < settings.QUANTITY\_OF\_ANSWERS; i++)

        {

            GameObject prefab = settings.fruitPrefabs[Random.Range(0, settings.fruitPrefabs.Length)];

            float xCoord = (i\*settings.xDistanceBetweenFruits)-settings.xDistanceBetweenFruits;

            GameObject answer = Instantiate(prefab, new Vector3(xCoord, settings.yCordOffset, settings.zCordOffset), Quaternion.identity);

            AnswerController answerController = answer.GetComponent("AnswerController") as AnswerController;

            answerController.Initialize(i-1);

            instancedAnswers.Add(answerController);

        }

    }

}

CarreritasEventManager.cs

using System;

using UnityEngine;

public class CarreritasEventManager : MonoBehaviour

{

    public static event Action OnPause;

    public static event Action OnQuestionGenerated;

    public static event Action<bool> OnAnswerQuestion;

    public static event Action OnFinishGame;

    public static void \_OnQuestionGenerated()

    {

        if (OnQuestionGenerated!=null) OnQuestionGenerated();

    }

    public static void \_OnPause()

    {

        if (OnPause!=null) OnPause();

    }

    public static void \_OnAnswerQuestion(bool correct)

    {

        if (OnAnswerQuestion!=null) OnAnswerQuestion(correct);

    }

    public static void \_OnFinishGame()

    {

        if (OnFinishGame!=null) OnFinishGame();

    }

}

CarreritasGameManager.cs

using UnityEngine;

public class CarreritasGameManager : MonoBehaviour

{

    public GameObject player;

    public CarreritasGameData gameData;

    public SettingsCarreritas settings;

    bool canMove = true;

    private void Start() {

        gameData.playerPositionIndex=0;

        player.transform.position = new Vector3(0, 0, 0);

        gameData.score=0;

        CarreritasEventManager.OnPause+=PauseGame;

        CarreritasEventManager.OnFinishGame+=FinishGame;

        CarreritasEventManager.OnAnswerQuestion+=UpdateScore;

        CarreritasEventManager.OnQuestionGenerated+=EnableMovement;

    }

    //IMPORTANT: unsuscribe from all events on destroy

    private void OnDestroy() {

        CarreritasEventManager.OnPause-=PauseGame;

        CarreritasEventManager.OnAnswerQuestion-=UpdateScore;

        gameData.paused=false;

        gameData.finished=false;

        gameData.score=0;

    }

    void EnableMovement()

    {

        canMove=true;

    }

    void UpdateScore(bool isCorrect)

    {

        canMove=false;

        if (isCorrect)

        {

            gameData.score++;

        }

    }

    void FinishGame()

    {

        gameData.finished=true;

    }

    void PauseGame()

    {

        gameData.paused=!gameData.paused;

    }

    // Update is called once per frame

    void Update()

    {

        if (!gameData.paused && !gameData.finished && canMove)

        {

            CheckKeyPress();

            CheckTouch();

        }

    }

    void CheckKeyPress()

    {

        if (Input.GetKeyDown("left")) MoveLeft();

        if (Input.GetKeyDown("right")) MoveRight();

    }

    void CheckTouch()

    {

        if (Input.touchCount > 0)

        {

            var touch = Input.GetTouch(0);

            if (touch.position.y<800 && touch.phase == TouchPhase.Began)

            {

                if (touch.position.x < Screen.width/2) MoveLeft();

                else if (touch.position.x > Screen.width/2) MoveRight();

            }

        }

    }

    void MoveLeft()

    {

        if (gameData.playerPositionIndex>-1){

           gameData.playerPositionIndex--;

           player.transform.position = new Vector3(gameData.playerPositionIndex\*settings.xDistanceBetweenFruits, 0, 0);

        }

    }

    void MoveRight()

    {

        if (gameData.playerPositionIndex<1){

           gameData.playerPositionIndex++;

           player.transform.position = new Vector3(gameData.playerPositionIndex\*settings.xDistanceBetweenFruits, 0, 0);

        }

    }

}

CarreritasUIManager.cs

using UnityEngine.SceneManagement;

using UnityEngine;

using TMPro;

public class CarreritasUIManager : MonoBehaviour

{

    public TextMeshProUGUI questionText;

    public TextMeshProUGUI scoreText;

    public CarreritasGameData gameData;

    public QuestionData questionData;

    public SelectedGameData selectedGameData;

    public GameObject finishGamePanel;

    public TextMeshProUGUI finalScoreText;

    public AccountAPI accountAPI;

    public GameObject pausePanel;

    void Start()

    {

        // Debug.Log(questionData.questionText);

        questionText.text = "";

        CarreritasEventManager.OnQuestionGenerated+=UpdateQuestionText;

        CarreritasEventManager.OnAnswerQuestion+=UpdateScore;

        CarreritasEventManager.OnFinishGame+=ShowFinishGamePanel;

    }

    private void OnDestroy() {

        CarreritasEventManager.OnQuestionGenerated-=UpdateQuestionText;

        CarreritasEventManager.OnAnswerQuestion-=UpdateScore;

        CarreritasEventManager.OnFinishGame-=ShowFinishGamePanel;

    }

    void UpdateQuestionText()

    {

        questionText.text = questionData.questionText;

    }

    public void TriggerPause()

    {

        pausePanel.SetActive(!pausePanel.activeSelf);

        CarreritasEventManager.\_OnPause();

    }

    public void UpdateScore(bool isCorrect){

        if (isCorrect)

        {

            scoreText.text= "Score: "+gameData.score;

        }

    }

    public void ShowFinishGamePanel()

    {

        int game = 1+(int)selectedGameData.selectedGame;

        int grade = 4+(int)selectedGameData.selectedGrade;

        accountAPI.AddScore(gameData.score, game, grade);

        finishGamePanel.SetActive(true);

        finalScoreText.text= "Score: "+gameData.score;

    }

    public void RestartGame()

    {

        SceneManager.LoadScene("Carreritas");

    }

    public void ReturnToMenu()

    {

        SceneManager.LoadScene("MenuScene");

    }

}

EnvironmentManager.cs

using UnityEngine;

public class EnvironmentManager : MonoBehaviour

{

    public CarreritasGameData gameData;

    public GameObject terrain1;

    public GameObject terrain2;

    public float speed = 2;

    public float threeshold=-45;

    GameObject currentTerrain;

    Vector3 terrain1StartPosition;

    Vector3 terrain2StartPosition;

    void Start() {

        currentTerrain=terrain2;

        terrain1StartPosition= terrain1.transform.position;

        terrain2StartPosition= terrain2.transform.position;

    }

    void Update()

    {

        if (!gameData.paused && !gameData.finished)

        {

            transform.Translate(Vector3.back \* Time.deltaTime \* speed);

            //Debug.Log(currentTerrain.transform.position);

            if (currentTerrain.transform.position.z<=terrain1StartPosition.z){

                if (currentTerrain==terrain1){

                    terrain2.transform.position=terrain2StartPosition;

                    currentTerrain=terrain2;

                } else {

                    terrain1.transform.position=terrain2StartPosition;

                    currentTerrain=terrain1;

                }

            }

        }

    }

}

QuestionGenerator.cs

using System.Collections;

using System.Linq;

using UnityEngine;

using System.Collections.Generic;

public class QuestionGenerator : MonoBehaviour

{

    public QuestionData questionData;

    public SettingsQuestions settingsQuestions;

    public SettingsCarreritas settingsCarreritas;

    public SelectedGameData selectedGameData;

    private void Start()

    {

        StartCoroutine(GenerateQuestions());

    }

    IEnumerator GenerateQuestions()

    {

        yield return new WaitForSeconds(1);

        for (int j = 0; j < settingsCarreritas.NUM\_OF\_QUESTIONS; j++)

        {

            GenerateNewQuestion();

            yield return new WaitForSeconds(settingsCarreritas.timeBetweenGeneration);

        }

        CarreritasEventManager.\_OnFinishGame();

    }

    public void GenerateNewQuestion()

    {

        SubjectThemeCarreritas randomSubjectTheme = (SubjectThemeCarreritas)0;

        if (selectedGameData.selectedGrade == Grade.sixthGrade)

            randomSubjectTheme = (SubjectThemeCarreritas)Random.Range(0, System.Enum.GetValues(typeof(SubjectThemeCarreritas)).Length);

        else randomSubjectTheme = (SubjectThemeCarreritas)Random.Range(0, System.Enum.GetValues(typeof(SubjectThemeCarreritas)).Length - 1);

        questionData.subject = randomSubjectTheme;

        int figures1 = 0, figures2 = 0, firstNumber = 0, secondNumber = 0, result = 0;

        int[] indexes = { 0, 1, 2 };

        switch (randomSubjectTheme)

        {

            case SubjectThemeCarreritas.Addition:

                if (selectedGameData.selectedGrade == Grade.fourthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.additionSettings.fourthGrade.minFigures,

                        settingsQuestions.additionSettings.fourthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.additionSettings.fourthGrade.minFigures,

                        settingsQuestions.additionSettings.fourthGrade.maxFigures + 1

                    ) - 1;

                    firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                    secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    result = firstNumber + secondNumber;

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.questionText = $"{firstNumber} + {secondNumber} = ?";

                }

                else if (selectedGameData.selectedGrade == Grade.fifthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.additionSettings.fifthGrade.minFigures,

                        settingsQuestions.additionSettings.fifthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.additionSettings.fifthGrade.minFigures,

                        settingsQuestions.additionSettings.fifthGrade.maxFigures + 1

                    ) - 1;

                    firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                    secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    result = firstNumber + secondNumber;

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.questionText = $"{firstNumber} + {secondNumber} = ?";

                }

                else if (selectedGameData.selectedGrade == Grade.sixthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.additionSettings.sixthGrade.minFigures,

                        settingsQuestions.additionSettings.sixthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.additionSettings.sixthGrade.minFigures,

                        settingsQuestions.additionSettings.sixthGrade.maxFigures + 1

                    ) - 1;

                    float firstNumberD = 0.0f, secondNumberD = 0.0f, resultD = 0.0f;

                    firstNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures1);

                    secondNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    resultD = firstNumberD + secondNumberD;

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0.0f, 100.0f).ToString("F2"));

                    questionData.answers[indexes[1]] = new Answer(resultD.ToString("F2"), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0.0f, 100.0f).ToString("F2"));

                    questionData.questionText = $"{firstNumberD.ToString("F2")} + {secondNumberD.ToString("F2")} = ?";

                }

                break;

            case SubjectThemeCarreritas.Substraction:

                if (selectedGameData.selectedGrade == Grade.fourthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.substractionSettings.fourthGrade.minFigures,

                        settingsQuestions.substractionSettings.fourthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.substractionSettings.fourthGrade.minFigures,

                        settingsQuestions.substractionSettings.fourthGrade.maxFigures + 1

                    ) - 1;

                    firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                    secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                    if (firstNumber < 0) firstNumber = firstNumber \* (-1);

                    if (secondNumber < 0) secondNumber = secondNumber \* (-1);

                    if (firstNumber < secondNumber)

                    {

                        int aux = firstNumber;

                        firstNumber = secondNumber;

                        secondNumber = aux;

                    }

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    result = firstNumber - secondNumber;

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.questionText = $"{firstNumber} - {secondNumber} = ?";

                }

                else if (selectedGameData.selectedGrade == Grade.fifthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.substractionSettings.fifthGrade.minFigures,

                        settingsQuestions.substractionSettings.fifthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.substractionSettings.fifthGrade.minFigures,

                        settingsQuestions.substractionSettings.fifthGrade.maxFigures + 1

                    ) - 1;

                    firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                    secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                    if (firstNumber < 0) firstNumber = firstNumber \* (-1);

                    if (secondNumber < 0) secondNumber = secondNumber \* (-1);

                    if (firstNumber < secondNumber)

                    {

                        int aux = firstNumber;

                        firstNumber = secondNumber;

                        secondNumber = aux;

                    }

                    result = firstNumber - secondNumber;

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                    questionData.questionText = $"{firstNumber} - {secondNumber} = ?";

                }

                else if (selectedGameData.selectedGrade == Grade.sixthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.substractionSettings.sixthGrade.minFigures,

                        settingsQuestions.substractionSettings.sixthGrade.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.substractionSettings.sixthGrade.minFigures,

                        settingsQuestions.substractionSettings.sixthGrade.maxFigures + 1

                    ) - 1;

                    float firstNumberD = 0.0f, secondNumberD = 0.0f, resultD = 0.0f;

                    firstNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures1);

                    secondNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures2);

                    if (firstNumberD < 0.0f) firstNumberD = firstNumberD \* (-1);

                    if (secondNumberD < 0.0f) secondNumberD = secondNumberD \* (-1);

                    if (firstNumberD < secondNumberD)

                    {

                        float aux = firstNumberD;

                        firstNumberD = secondNumberD;

                        secondNumberD = aux;

                    }

                    resultD = firstNumberD - secondNumberD;

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    questionData.answers[indexes[0]] = new Answer(Random.Range(0.0f, 100.0f).ToString("F2"));

                    questionData.answers[indexes[1]] = new Answer(resultD.ToString("F2"), true);

                    questionData.answers[indexes[2]] = new Answer(Random.Range(0.0f, 100.0f).ToString("F2"));

                    questionData.questionText = $"{firstNumberD.ToString("F2")} - {secondNumberD.ToString("F2")} = ?";

                }

                break;

            case SubjectThemeCarreritas.Multiplication:

                if (selectedGameData.selectedGrade == Grade.fourthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.multiplicationSettings.fourthGrade.firstNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.fourthGrade.firstNumberSettings.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.multiplicationSettings.fourthGrade.SecondNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.fourthGrade.SecondNumberSettings.maxFigures + 1

                    ) - 1;

                }

                else if (selectedGameData.selectedGrade == Grade.fifthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.multiplicationSettings.fifthGrade.firstNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.fifthGrade.firstNumberSettings.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.multiplicationSettings.fifthGrade.SecondNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.fifthGrade.SecondNumberSettings.maxFigures + 1

                    ) - 1;

                }

                else if (selectedGameData.selectedGrade == Grade.sixthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.multiplicationSettings.sixthGrade.firstNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.sixthGrade.firstNumberSettings.maxFigures + 1

                    ) - 1;

                    figures2 = Random.Range(

                        settingsQuestions.multiplicationSettings.sixthGrade.SecondNumberSettings.minFigures,

                        settingsQuestions.multiplicationSettings.sixthGrade.SecondNumberSettings.maxFigures + 1

                    ) - 1;

                }

                firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                result = firstNumber \* secondNumber;

                indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                questionData.questionText = $"{firstNumber} x {secondNumber} = ?";

                break;

            case SubjectThemeCarreritas.Division:

                if (selectedGameData.selectedGrade == Grade.fifthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.divisionSettings.fifthGrade.firstNumberSettings.minFigures,

                        settingsQuestions.divisionSettings.fifthGrade.firstNumberSettings.maxFigures + 1

                    ) - 1; ;

                    figures2 = Random.Range(

                        settingsQuestions.divisionSettings.fifthGrade.SecondNumberSettings.minFigures,

                        settingsQuestions.divisionSettings.fifthGrade.SecondNumberSettings.maxFigures + 1

                    ) - 1;

                }

                else if (selectedGameData.selectedGrade == Grade.sixthGrade)

                {

                    figures1 = Random.Range(

                      settingsQuestions.divisionSettings.sixthGrade.firstNumberSettings.minFigures,

                      settingsQuestions.divisionSettings.sixthGrade.firstNumberSettings.maxFigures + 1

                  ) - 1; ;

                    figures2 = Random.Range(

                        settingsQuestions.divisionSettings.sixthGrade.SecondNumberSettings.minFigures,

                        settingsQuestions.divisionSettings.sixthGrade.SecondNumberSettings.maxFigures + 1

                    ) - 1;

                }

                firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                int multiplication = firstNumber \* secondNumber;

                if (multiplication > 1000)

                {

                    multiplication = multiplication / 10;

                    secondNumber = secondNumber / 10;

                }

                result = secondNumber;

                indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                questionData.answers[indexes[0]] = new Answer(Random.Range(0, 100).ToString());

                questionData.answers[indexes[1]] = new Answer(result.ToString(), true);

                questionData.answers[indexes[2]] = new Answer(Random.Range(0, 100).ToString());

                questionData.questionText = $"{multiplication} / {firstNumber} = ?";

                break;

            case SubjectThemeCarreritas.NumberConversion:

                int numeralSys = 0; //Random.Range(0, 1);

                int[] numbersToConvert = { Random.Range(1, 100), Random.Range(1, 100), Random.Range(1, 100) };

                if (numeralSys == 0)

                { //Roman Sys

                    List<string> romanNumbers = new List<string>();

                    // string hundreds = numberToConvert.ToString().Substring(0, 1);

                    string tens;

                    string units;

                    foreach (int numberToConvert in numbersToConvert)

                    {

                        if (numberToConvert < 10)

                        {

                            tens = "0";

                            units = numberToConvert.ToString().Substring(0, 1);

                        }

                        else

                        {

                            tens = numberToConvert.ToString().Substring(0, 1);

                            units = numberToConvert.ToString().Substring(1, 1);

                        }

                        string romanNumber = "";

                        switch (tens)

                        {

                            case "1":

                                romanNumber += "X";

                                break;

                            case "2":

                                romanNumber += "XX";

                                break;

                            case "3":

                                romanNumber += "XXX";

                                break;

                            case "4":

                                romanNumber += "XL";

                                break;

                            case "5":

                                romanNumber += "L";

                                break;

                            case "6":

                                romanNumber += "LX";

                                break;

                            case "7":

                                romanNumber += "LXX";

                                break;

                            case "8":

                                romanNumber += "LXXX";

                                break;

                            case "9":

                                romanNumber += "XC";

                                break;

                            default:

                                Debug.Log(tens);

                                break;

                        }

                        switch (units)

                        {

                            case "1":

                                romanNumber += "I";

                                break;

                            case "2":

                                romanNumber += "II";

                                break;

                            case "3":

                                romanNumber += "III";

                                break;

                            case "4":

                                romanNumber += "IV";

                                break;

                            case "5":

                                romanNumber += "V";

                                break;

                            case "6":

                                romanNumber += "VI";

                                break;

                            case "7":

                                romanNumber += "VII";

                                break;

                            case "8":

                                romanNumber += "VIII";

                                break;

                            case "9":

                                romanNumber += "IX";

                                break;

                            default:

                                Debug.Log(units);

                                break;

                        }

                        romanNumbers.Add(romanNumber);

                    }

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    questionData.answers[indexes[0]] = new Answer(romanNumbers.ElementAt(0), true);

                    questionData.answers[indexes[1]] = new Answer(romanNumbers.ElementAt(1));

                    questionData.answers[indexes[2]] = new Answer(romanNumbers.ElementAt(2));

                    questionData.questionText = $"{numbersToConvert[0]} = ?";

                }

                else

                { //Mayan Sys

                }

                break;

            case SubjectThemeCarreritas.Order:

                if (selectedGameData.selectedGrade == Grade.fourthGrade)

                {

                    figures1 = Random.Range(

                       settingsQuestions.orderSettings.fourthGrade.minFigures,

                       settingsQuestions.orderSettings.fourthGrade.maxFigures + 1

                   ) - 1; ;

                    figures2 = Random.Range(

                       settingsQuestions.orderSettings.fourthGrade.minFigures,

                       settingsQuestions.orderSettings.fourthGrade.maxFigures + 1

                    ) - 1;

                    firstNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures1);

                    secondNumber = Random.Range(1, 9) \* (int)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    if (firstNumber > secondNumber)

                    {

                        questionData.answers[indexes[0]] = new Answer(">", true);

                        questionData.answers[indexes[1]] = new Answer("=");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    else if (secondNumber < firstNumber)

                    {

                        questionData.answers[indexes[0]] = new Answer("<", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("=");

                    }

                    else

                    {

                        questionData.answers[indexes[0]] = new Answer("=", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    questionData.questionText = $"{firstNumber} ? {secondNumber}";

                }

                else if (selectedGameData.selectedGrade == Grade.fifthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.orderSettings.fifthGrade.minFigures,

                        settingsQuestions.orderSettings.fifthGrade.maxFigures + 1

                    ) - 1; ;

                    figures2 = Random.Range(

                       settingsQuestions.orderSettings.fifthGrade.minFigures,

                       settingsQuestions.orderSettings.fifthGrade.maxFigures + 1

                    ) - 1;

                    float firstNumberD = 0.0f, secondNumberD = 0.0f;

                    firstNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures1);

                    secondNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    if (firstNumberD > secondNumberD)

                    {

                        questionData.answers[indexes[0]] = new Answer(">", true);

                        questionData.answers[indexes[1]] = new Answer("=");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    else if (secondNumberD < firstNumberD)

                    {

                        questionData.answers[indexes[0]] = new Answer("<", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("=");

                    }

                    else

                    {

                        questionData.answers[indexes[0]] = new Answer("=", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    questionData.questionText = $"{firstNumberD.ToString("F2")} ? {secondNumberD.ToString("F2")}";

                }

                else if (selectedGameData.selectedGrade == Grade.sixthGrade)

                {

                    figures1 = Random.Range(

                        settingsQuestions.orderSettings.sixthGrade.minFigures,

                        settingsQuestions.orderSettings.sixthGrade.maxFigures + 1

                    ) - 1; ;

                    figures2 = Random.Range(

                       settingsQuestions.orderSettings.sixthGrade.minFigures,

                       settingsQuestions.orderSettings.sixthGrade.maxFigures + 1

                    ) - 1;

                    float firstNumberD = 0.0f, secondNumberD = 0.0f;

                    firstNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures1);

                    secondNumberD = Random.Range(1.0f, 9.0f) \* (float)Mathf.Pow(10, figures2);

                    indexes = indexes.OrderBy(x => Random.Range(0, 10)).ToArray();

                    if (firstNumberD > secondNumberD)

                    {

                        questionData.answers[indexes[0]] = new Answer(">", true);

                        questionData.answers[indexes[1]] = new Answer("=");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    else if (secondNumberD < firstNumberD)

                    {

                        questionData.answers[indexes[0]] = new Answer("<", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("=");

                    }

                    else

                    {

                        questionData.answers[indexes[0]] = new Answer("=", true);

                        questionData.answers[indexes[1]] = new Answer(">");

                        questionData.answers[indexes[2]] = new Answer("<");

                    }

                    questionData.questionText = $"{firstNumberD.ToString("F2")} ? {secondNumberD.ToString("F2")}";

                }

                break;

            default:

                randomSubjectTheme = (SubjectThemeCarreritas)Random.Range(0, System.Enum.GetValues(typeof(SubjectThemeCarreritas)).Length);

                break;

        }

        CarreritasEventManager.\_OnQuestionGenerated();

    }

}

ChocolatosoGameData.cs

using System;

using UnityEngine;

[CreateAssetMenu(fileName = "ChocolatosoGameData", menuName = "Chocolatoso/GameData", order = 0)]

public class ChocolatosoGameData : ScriptableObject

{

    public int NUMBER\_OF\_QUESTIONS;

    public int questionNumber = 1;

    public int score;

    public int xSlider;

    public int ySlider;

    public int commonDenominator;

    public Fraction fraction1;

    public Fraction fraction2;

    public Fraction fractionResult;

    public Fraction fraction1Equivalent;

    public Fraction fraction2Equivalent;

}

[Serializable]

public struct Fraction

{

    public int numerator;

    public int denominator;

    public Fraction(int numerator, int denominator)

    {

        this.numerator = numerator;

        this.denominator = denominator;

    }

}

ChocolatosoUIManager.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

using UnityEngine.UI;

public class ChocolatosoUIManager : MonoBehaviour

{

    public SelectedGameData selectedGameData;

    public ChocolatosoSettingsData settingsData;

    public ChocolatosoGameData gameData;

    public Slider xSlider;

    public Slider ySlider;

    public TextMeshProUGUI xSliderText;

    public TextMeshProUGUI ySliderText;

    public TextMeshProUGUI mcmText;

    public FractionText fractionText1;

    public FractionText fractionText2;

    public FractionText equivalentFractionText1;

    public FractionText equivalentFractionText2;

    public GameObject chocolatPrefab;

    public GameObject chocolatsParent;

    public float xOffset;

    public float yOffset;

    GameObject[,] chocolats;

    public GameObject panel1;

    public GameObject panel2;

    public GameObject panel3;

    public GameObject panel4;

    //panel 3

    public TMP\_InputField EquivalentN1Input;

    public TMP\_InputField EquivalentD1Input;

    public TMP\_InputField EquivalentN2Input;

    public TMP\_InputField EquivalentD2Input;

    // chocolats panel

    public FractionText fractionText1Panel;

    public FractionText fractionText2Panel;

    public FractionText fractionTextChocolat;

    public GameObject chocolatsPanel;

    public GameObject scalableUI;

    public GameObject chocolatsPanelParent;

    public GameObject fractionsUIHelp;

    public TMP\_InputField CommonDenominatorInput;

    public RectTransform pinkBox;

    public RectTransform blueBox;

    // panel 4

    public FractionText fractionText1Orig;

    public FractionText fractionText1Equiv;

    public FractionText fractionText2Orig;

    public FractionText fractionText2Equiv;

    public FractionText fractionTextLeft;

    public FractionText fractionTextRight;

    private void Start() {

        GenerateQuestion();

        gameData.xSlider=1;

        gameData.ySlider=1;

        GenerateGrid();

        UpdateGrid();

        xSlider.value=gameData.xSlider;

        ySlider.value=gameData.ySlider;

    }

    public void GoToPanel(int id)

    {

        switch (id)

        {

            case 1:

                GenerateQuestion();

                DestroyPanelChocolats();

                panel4.SetActive(false);

                panel1.SetActive(true);

                chocolatsPanel.SetActive(false);

                break;

            case 2:

                panel1.SetActive(false);

                panel2.SetActive(true);

                chocolatsPanel.SetActive(true);

                GenerateChocolatsPanelGrid();

                break;

            case 3:

                panel2.SetActive(false);

                panel3.SetActive(true);

                gameData.commonDenominator = int.Parse(CommonDenominatorInput.text);

                mcmText.text= "mcm: "+gameData.commonDenominator;

                break;

            case 4:

                panel3.SetActive(false);

                panel4.SetActive(true);

                gameData.fraction1Equivalent = new Fraction(

                    int.Parse(EquivalentN1Input.text),

                    int.Parse(EquivalentD1Input.text)

                );

                gameData.fraction2Equivalent = new Fraction(

                    int.Parse(EquivalentN2Input.text),

                    int.Parse(EquivalentD2Input.text)

                );

                fractionText1Orig.SetFraction(gameData.fraction1);

                fractionText1Equiv.SetFraction(gameData.fraction1Equivalent);

                fractionText2Orig.SetFraction(gameData.fraction2);

                fractionText2Equiv.SetFraction(gameData.fraction2Equivalent);

                fractionTextLeft.SetFraction(gameData.fraction1Equivalent);

                fractionTextRight.SetFraction(gameData.fraction2Equivalent);

                break;

            default:

                panel4.SetActive(false);

                panel1.SetActive(true);

                chocolatsPanel.SetActive(false);

                break;

        }

    }

    public void OnChangeXSlider(){

        if (xSlider.value==0f){

            xSlider.value=1f;

            return;

        }

        gameData.xSlider = (int)xSlider.value;

        xSliderText.text=gameData.xSlider.ToString();

        UpdateGrid();

    }

    public void OnChangeYSlider(){

        if (ySlider.value==0f){

            ySlider.value=1f;

            return;

        }

        gameData.ySlider = (int)ySlider.value;

        ySliderText.text=gameData.ySlider.ToString();

        UpdateGrid();

    }

    public void GenerateGrid(){

        chocolats = new GameObject[(int)xSlider.maxValue, (int)ySlider.maxValue];

        for (int i = 0; i < xSlider.maxValue; i++)

        {

            for (int j = 0; j < ySlider.maxValue; j++)

            {

                Vector3 position = new Vector3(i\*46+xOffset, j\*46+yOffset, 0);

                GameObject chocolat = Instantiate(chocolatPrefab, position, Quaternion.identity);

                chocolat.SetActive(false);

                chocolat.transform.SetParent(chocolatsParent.transform, false);

                chocolats[i,j]=chocolat;

            }

        }

    }

    public void UpdateGrid(){

        for (int i = 0; i < xSlider.maxValue; i++)

        {

            for (int j = 0; j < ySlider.maxValue; j++)

            {

                GameObject chocolat = chocolats[i,j];

                if (i<gameData.xSlider && j<gameData.ySlider)

                {

                    chocolat.SetActive(true);

                } else

                {

                    chocolat.SetActive(false);

                }

            }

        }

    }

    void GenerateQuestion()

    {

        TwoNumbersFiguresLimitsSettings settings;

        switch (selectedGameData.selectedGrade)

        {

            case Grade.fourthGrade:

                settings = settingsData.fourthGrade;

                break;

            case Grade.fifthGrade:

                settings = settingsData.fifthGrade;

                break;

            default:

                settings = settingsData.sixthGrade;

                break;

        }

        int D1 = Random.Range(settings.firstNumberSettings.minFigures, settings.firstNumberSettings.maxFigures+1);

        int N1 = Random.Range(1, D1-1);

        int D2 = Random.Range(settings.SecondNumberSettings.minFigures, settings.SecondNumberSettings.maxFigures+1);

        int N2 = Random.Range(1, D2-1);

        gameData.fraction1 = new Fraction(N1, D1);

        gameData.fraction2 = new Fraction(N2, D2);

        fractionText1.SetFraction(gameData.fraction1);

        fractionText2.SetFraction(gameData.fraction2);

        equivalentFractionText1.SetFraction(gameData.fraction1);

        equivalentFractionText2.SetFraction(gameData.fraction2);

    }

    public void GenerateChocolatsPanelGrid(){

        fractionText1Panel.SetFraction(new Fraction(1, gameData.ySlider));

        fractionText2Panel.SetFraction(new Fraction(1, gameData.xSlider));

        fractionTextChocolat.SetFraction(new Fraction(1, gameData.xSlider\*gameData.ySlider));

        float chocolatSize = chocolatPrefab.GetComponent<RectTransform>().rect.width;

        blueBox.sizeDelta = new Vector2(chocolatSize\*gameData.xSlider, chocolatSize);

        pinkBox.sizeDelta = new Vector2(chocolatSize, chocolatSize\*gameData.ySlider);

        float xOffset = (gameData.xSlider-2)\*chocolatSize/2f;

        float yOffset = (gameData.ySlider-2)\*chocolatSize/2f;

        for (int i = 0; i < gameData.xSlider; i++)

        {

            for (int j = 0; j < gameData.ySlider; j++)

            {

                Vector3 position = new Vector3(i\*chocolatSize-xOffset, j\*chocolatSize-yOffset, 0);

                GameObject chocolat = Instantiate(chocolatPrefab, position, Quaternion.identity);

                //chocolat.SetActive(false);

                chocolat.transform.SetParent(chocolatsPanelParent.transform, false);

            }

        }

        float max = Mathf.Max(gameData.xSlider, gameData.ySlider);

        float newScale = 1f-(max/11f)\*0.45f;

        scalableUI.GetComponent<RectTransform>().localScale= new Vector3(newScale, newScale, 0);

        fractionsUIHelp.transform.localPosition = new Vector3(-xOffset, -yOffset, 0);

    }

    void DestroyPanelChocolats(){

        foreach (Transform child in chocolatsPanelParent.transform) {

            GameObject.Destroy(child.gameObject);

        }

    }

}

DragDropUIController.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.EventSystems;

public class DragDropUIController : MonoBehaviour

{

    [SerializeField]

    private Canvas canvas;

    private Vector3 initialPosition;

    public EspiritusGameData gameData;

    private void Start() {

        initialPosition = transform.position;

    }

    public void OnDrag(BaseEventData data)

    {

        if (gameData.knightsCount==0) return;

        PointerEventData pointerData = (PointerEventData)data;

        Vector2 position;

        RectTransformUtility.ScreenPointToLocalPointInRectangle(

            (RectTransform)canvas.transform,

            pointerData.position,

            canvas.worldCamera,

            out position

        );

        transform.position = canvas.transform.TransformPoint(position);

    }

    public void OnDrop(BaseEventData data)

    {

        PointerEventData pointerData = (PointerEventData)data;

        float rayLength = 500f;

        RaycastHit hit;

        Ray r = Camera.main.ScreenPointToRay( pointerData.position);

        if (Physics.Raycast(r, out hit) && hit.transform.tag == "Slot")

        {

            Debug.Log("intersection");

            Transform objectHit = hit.transform;

            Debug.Log(objectHit.position);

            EspiritusEventManager.\_OnKnightSelected(objectHit.position);

        }

        Debug.DrawRay(r.origin , r.direction \* rayLength, Color.red, 2000);

        transform.position = initialPosition;

        //EspiritusEventManager.\_OnKnightSelected()

    }

}

EspiritusEventManager.cs

using System;

using UnityEngine;

public class EspiritusEventManager : MonoBehaviour

{

    public static event Action OnQuestionGenerated;

    public static event Action<Vector3> OnKnightSelected;

    public static event Action OnKnightPut;

    public static event Action OnKnightAttack;

    public static event Action OnMonsterDead;

    public static event Action OnMonsterAttacked;

    public static event Action<int> OnChangeStage;

    public static event Action OnFinishGame;

    public static void \_OnQuestionGenerated()

    {

        if (OnQuestionGenerated!=null) OnQuestionGenerated();

    }

    public static void \_OnKnightSelected(Vector3 position)

    {

        if (OnKnightSelected!=null) OnKnightSelected(position);

    }

    public static void \_OnKnightPut()

    {

        if (OnKnightPut!=null) OnKnightPut();

    }

    public static void \_OnKnightAttack()

    {

        if (OnKnightAttack!=null) OnKnightAttack();

    }

    public static void \_OnMonsterDead()

    {

        if (OnMonsterDead!=null) OnMonsterDead();

    }

    public static void \_OnMonsterAttacked()

    {

        if (OnMonsterAttacked!=null) OnMonsterAttacked();

    }

    public static void \_OnChangeStage(int stage)

    {

        if (OnChangeStage!=null) OnChangeStage(stage);

    }

    public static void \_OnFinishGame()

    {

        if (OnFinishGame!=null) OnFinishGame();

    }

}

EspiritusGameManager.cs

using System.Collections.Generic;

using UnityEngine;

public class EspiritusGameManager : MonoBehaviour

{

    public EspiritusGameData gameData;

    public GameObject knightPrefab;

    Queue<KnightController> instancedKnights = new Queue<KnightController>();

    public SelectedGameData selectedGameData;

    public SettingsEspiritusQuestions settingsEspiritusQuestions;

    // Start is called before the first frame update

    void Start()

    {

        GenerateMonsters();

        EspiritusEventManager.OnKnightSelected += PutKnight;

        EspiritusEventManager.OnKnightAttack += DecreaseMonsterLife;

        EspiritusEventManager.OnChangeStage += OnChangeStage;

    }

    private void OnDestroy()

    {

        gameData.score = 0;

        gameData.questionNumber = 0;

        gameData.knightsSelected = 0;

        gameData.knightsHaveAttack = 0;

        gameData.stage = EspiritusGameStage.Select;

        EspiritusEventManager.OnKnightSelected -= PutKnight;

        EspiritusEventManager.OnKnightAttack -= DecreaseMonsterLife;

        EspiritusEventManager.OnChangeStage -= OnChangeStage;

    }

    void OnChangeStage(int stage)

    {

        switch (stage)

        {

            case 0:

                GoToNextQuestion();

                break;

            default:

                break;

        }

    }

    public void DecreaseMonsterLife()

    {

        gameData.knightsHaveAttack++;

        int newLife = gameData.monsterLife - gameData.knightsAttack;

        if (newLife <= 0)

        {

            gameData.stage = EspiritusGameStage.Result;

            gameData.monsterLife = 0;

            EspiritusEventManager.\_OnMonsterDead();

            EspiritusEventManager.\_OnChangeStage((int)gameData.stage);

        }

        else

        {

            gameData.monsterLife -= gameData.knightsAttack;

            EspiritusEventManager.\_OnMonsterAttacked();

            if (gameData.knightsHaveAttack == gameData.knightsSelected)

            {

                gameData.stage = EspiritusGameStage.Result;

                EspiritusEventManager.\_OnChangeStage((int)gameData.stage);

            }

        }

    }

    public void GoToNextQuestion()

    {

        if (gameData.questionNumber >= gameData.NUMBER\_OF\_QUESTIONS)

        {

            EspiritusEventManager.\_OnFinishGame();

            return;

        }

        Debug.Log("gameData");

        Debug.Log(gameData.knightsCount);

        //aquí poner la lógica para evaluar cuanto score dar dependiendo de

        //gameData.knightsSelected, gameData.knightsHaveAttack, gameData.monsterLife

        gameData.score++;

        gameData.questionNumber++;

        GenerateMonsters();

    }

    // Update is called once per frame

    void PutKnight(Vector3 position)

    {

        gameData.knightsCount--;

        gameData.knightsSelected++;

        KnightController knight = instancedKnights.Dequeue();

        knight.transform.position = new Vector3(position.x, 0.45f, position.z);

        knight.SetInitialPosition();

        instancedKnights.Enqueue(knight);

        EspiritusEventManager.\_OnKnightPut();

    }

    void GenerateMonsters()

    {

        int firstNumber = 0;

        if (selectedGameData.selectedGrade == Grade.fourthGrade)

        {

            gameData.knightsAttack = Random.Range( //range [334, 999] de esta forma se garantiza una vida al mounstro de 4 digitos

                settingsEspiritusQuestions.fourthGrade.minFigures,

                settingsEspiritusQuestions.fourthGrade.maxFigures + 1

            ) - 1;

            firstNumber = Random.Range(3, 10); //numero de caballeros correcto

            gameData.knightsCount = Random.Range(firstNumber, 10); //numero de caballeros a mostrar

            gameData.monsterLife = firstNumber \* gameData.knightsAttack;

        }

        else if (selectedGameData.selectedGrade == Grade.fifthGrade || selectedGameData.selectedGrade == Grade.sixthGrade)

        {

            gameData.knightsAttack = Random.Range( //range [3334, 9999] de esta forma se garantiza una vida al mounstro de 5 digitos

                settingsEspiritusQuestions.fourthGrade.minFigures,

                settingsEspiritusQuestions.fourthGrade.maxFigures + 1

            ) - 1;

            firstNumber = Random.Range(3, 10); //numero de caballeros correcto

            gameData.knightsCount = Random.Range(firstNumber, 10); //numero de caballeros a mostrar

            gameData.monsterLife = firstNumber \* gameData.knightsAttack;

        }

        if (instancedKnights.Count == 0)

        {

            for (int i = 0; i < gameData.maxKnights; i++)

            {

                GameObject knight = Instantiate(knightPrefab, new Vector3(0, -3, 0), Quaternion.identity);

                KnightController knightController = knight.GetComponent("KnightController") as KnightController;

                instancedKnights.Enqueue(knightController);

            }

        }

        else

        {

            foreach (var knight in instancedKnights)

            {

                knight.transform.position = new Vector3(0, -3, 0);

            }

        }

        gameData.knightsCount = firstNumber;

        EspiritusEventManager.\_OnQuestionGenerated();

    }

}

KnightController.cs using UnityEngine;

public class KnightController : MonoBehaviour

{

    private Color mouseOverColor = Color.blue;

    private Color originalColor = Color.yellow;

    private bool dragging = false;

    private float distance;

    private Vector3 startDist;

    private Vector3 initialPosition;

    private bool hasAttacked = false;

    public EspiritusGameData gameData;

    public void SetInitialPosition() {

        initialPosition= transform.position;

    }

    void OnMouseDown()

    {

        if (hasAttacked) return;

        if(gameData.stage==EspiritusGameStage.Attack){

            distance = Vector3.Distance(transform.position, Camera.main.transform.position);

            dragging = true;

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            Vector3 rayPoint = ray.GetPoint(distance);

            startDist = transform.position - rayPoint;

        }

    }

    void OnMouseUp()

    {

        if (dragging)

        {

            dragging = false;

            float rayLength = 500f;

            RaycastHit hit;

            Ray r = Camera.main.ScreenPointToRay( Input.mousePosition);

            if (Physics.Raycast(r, out hit) && hit.transform.tag == "Monster")

            {

                hasAttacked=true;

                EspiritusEventManager.\_OnKnightAttack();

            } else

            {

                Debug.Log("No Action");

            }

            Debug.DrawRay(r.origin , r.direction \* rayLength, Color.red, 2000);

            transform.position = initialPosition;

        }

    }

    void Update()

    {

        if (dragging)

        {

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            Vector3 rayPoint = ray.GetPoint(distance);

            Vector3 pos = rayPoint + startDist;

            transform.position = new Vector3(pos.x, initialPosition.y, pos.z);

            //transform.position = pos;

        }

    }

}