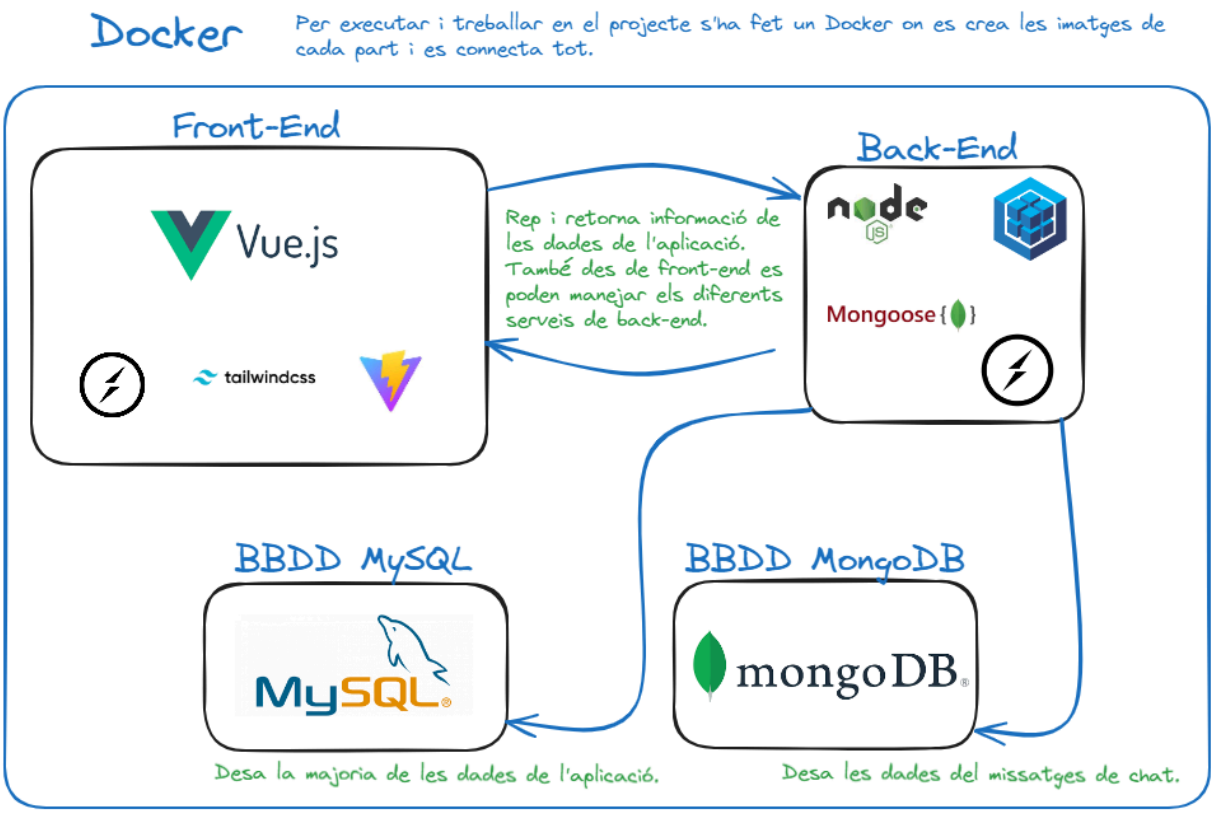
Documentació Tècnica

Índex

1. Arquitectura de l’aplicació



Aquest és l’esquema visual de les eines utilitzades per la part de web, a més es troba un breu resum de com i per què es connecten.

2. Rutes de l’aplicació

Assistence

```
/**
 * Obté totes les assistències
 * @route GET /assistence/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de totes les assistències
 * @returns {number} Object[].id - Identificador de l'assistència
 * @returns {number} Object[].user_id - Identificador de l'usuari
 * @returns {number} Object[].course_id - Identificador del curs
 * @returns {string} Object[].hour - Hora de l'assistència
 * @returns {string} Object[].day - Dia de l'assistència
 * @returns {string} Object[].assisted - Estat de l'assistència ("yes", "no", "not selected")
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    const assistance = await Assistence.findAll();
    res.json(assistance);
});

/**
 * Obté una assistència per ID
 * @route GET /assistence/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'assistència
 * @returns {Object} - Detalls de l'assistència
 * @returns {number} Object.id - Identificador de l'assistència
 * @returns {number} Object.user_id - Identificador de l'usuari
 * @returns {number} Object.course_id - Identificador del curs
 * @returns {string} Object.hour - Hora de l'assistència
 * @returns {string} Object.day - Dia de l'assistència
 * @returns {string} Object.assisted - Estat de l'assistència ("yes", "no", "not selected")
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const assistance = await Assistence.findByPk(req.params.id);
        res.json(assistance);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea o actualitza una assistència
 * @route POST /assistence/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {Object} req.body - Dades de l'assistència
 * @param {number} req.body.user_id - Identificador de l'usuari
```

```
 * @param {number} req.body.course_id - Identificador del curs
 * @param {string} req.body.hour - Hora de l'assistència
 * @param {string} req.body.day - Dia de l'assistència
 * @param {string} req.body.assisted - Estat de l'assistència ("yes", "no", "not selected")
 * @returns {Object} - Assistència creada o actualitzada
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        let { user_id, course_id, hour, day, assisted } = req.body;
        if (!user_id || !course_id || !hour || !assisted) {
            return res.status(400).json({ message: "user_id, course_id, hour i assisted són obligatoris" });
        }
        course_id = parseInt(course_id);
        user_id = parseInt(user_id);
        const user = await User.findOne({ where: { id: user_id } });
        const course = await Course.findOne({ where: { id: course_id } });
        if (!user) {
            return res.status(404).json({ message: "user_id no trobat" });
        }
        if (!course) {
            return res.status(404).json({ message: "course_id no trobat" });
        }
        const userCourse = await UserCourse.findOne({ where: { user_id, course_id } });
        if (!userCourse) {
            return res.status(404).json({ message: "user_id no està inscrit al course_id" });
        }
        const hourExists = await Assistence.findOne({ where: { hour, user_id, day } });
        let assistance;
        if (hourExists) {
            assistance = Assistence.update({ assisted }, { where: { hour, user_id, day } });
        }
        else {
            assistance = await Assistence.create({ user_id, course_id, hour, assisted, day });
        }
        res.json(assistance);
    }
    catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza l'estat d'una assistència per ID
 * @route PUT /assistence/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'assistència
 * @param {Object} req.body - Dades a actualitzar
```

```
  * @param {string} req.body.assisted - Nou estat de l'assistència
("yes", "no", "not selected")
 * @returns {Object} - Resultat de l'actualització
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { assisted } = req.body;
           const assistance = await Assistence.update({ assisted }, {
where: { id: req.params.id } });
        res.json(assistance);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina una assistència per ID
 * @route DELETE /assistence/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'assistència
 * @returns {Object} message - Missatge de confirmació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    const assistance = await Assistence.findByPk(req.params.id);
    if (!assistance) {
         return res.status(404).json({ message: "Assistència no trobada"
});
    } else {
        try {
            await Assistence.destroy();
               res.json({ message: "Assistència eliminada correctament"
});
        } catch (error) {
            res.status(500).json({ message: error.message });
        }
    }
});

/**
 * Obté totes les assistències d'un curs
 * @route GET /assistence/course/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador del curs
 * @returns {Object[]} - Llista d'assistències del curs
 * @returns {number} Object[].id - Identificador de l'assistència
 * @returns {number} Object[].user_id - Identificador de l'usuari
 * @returns {number} Object[].course_id - Identificador del curs
 * @returns {string} Object[].hour - Hora de l'assistència
 * @returns {string} Object[].day - Dia de l'assistència
 * @returns {string} Object[].assisted - Estat de l'assistència ("yes",
"no", "not selected")
 */
router.get("/course/:id", verifyTokenMiddleware, async (req, res) => {
    try {
              const assistance = await Assistence.findAll({ where: {
course_id: req.params.id } });
```

```
        res.json(assistance);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté totes les assistències d'un curs en un dia específic
 * @route GET /assistence/course/:courseId/day/:day
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} courseId - Identificador del curs
 * @param {string} day - Dia de l'assistència (format: "YYYY-MM-DD")
 * @returns {Object[]} - Llista d'assistències del curs en un dia específic
 * @returns {number} Object[].id - Identificador de l'assistència
 * @returns {number} Object[].user_id - Identificador de l'usuari
 * @returns {number} Object[].course_id - Identificador del curs
 * @returns {string} Object[].hour - Hora de l'assistència
 * @returns {string} Object[].day - Dia de l'assistència
 * @returns {string} Object[].assisted - Estat de l'assistència ("yes", "no", "not selected")
 */
router.get("/course/:courseId/day/:day", verifyTokenMiddleware, async (req, res) => {
    try {
        const { courseId, day } = req.params;
            const assistance = await Assistence.findAll({ where: { course_id: courseId, day } });
        res.json(assistance);
    }
    catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté totes les assistències d'un usuari en un curs específic (exclou les marcades com "not selected")
 * @route GET /assistence/user/:userId/course/:courseId
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} userId - Identificador de l'usuari
 * @param {number} courseId - Identificador del curs
 * @returns {Object[]} - Llista d'assistències de l'usuari en el curs específic (només "yes" o "no")
 * @returns {number} Object[].id - Identificador de l'assistència
 * @returns {number} Object[].user_id - Identificador de l'usuari
 * @returns {number} Object[].course_id - Identificador del curs
 * @returns {string} Object[].hour - Hora de l'assistència
 * @returns {string} Object[].day - Dia de l'assistència
 * @returns {string} Object[].assisted - Estat de l'assistència ("yes" o "no")
 */
router.get("/user/:userId/course/:courseId", verifyTokenMiddleware, async (req, res) => {
    try {
        const { userId, courseId } = req.params;
```

```
        const assistance = await Assistence.findAll({ where: { user_id:
userId, course_id: courseId } });
        let auxAssistance = [];
        assistance.forEach((assistance) => {
            if (assistance.assisted != "not selected") {
                auxAssistance.push(assistance);
            }
        });
        res.json(auxAssistance);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Canteen Item

```
/**
 * Obté tots els elements de cantina
 * @route GET /canteenItem/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista de tots els elements de cantina
 * @returns {number} Object[].id - Identificador de l'element
 * @returns {string} Object[].product_name - Nom del producte
 * @returns {number} Object[].product_price - Preu del producte
 * @returns {boolean} Object[].product_enabled - Indica si el producte
està disponible
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const canteenItems = await CanteenItem.findAll();
        res.json(canteenItems);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté un element de cantina per ID
 * @route GET /canteenItem/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'element de cantina
 * @returns {Object} - Detalls de l'element de cantina
 * @returns {number} Object.id - Identificador de l'element
 * @returns {string} Object.product_name - Nom del producte
 * @returns {number} Object.product_price - Preu del producte
 * @returns {boolean} Object.product_enabled - Indica si el producte
està disponible
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const canteenItem = await CanteenItem.findByPk(req.params.id);
        if (!canteenItem) {
```

```
            return res.status(404).json({ message: "Element de cantina
no trobat" });
        }
        res.json(canteenItem);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea un nou element de cantina
 * @route POST /canteenItem/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {Object} req.body - Dades del nou element de cantina
 * @param {string} req.body.product_name - Nom del producte
 * @param {number} req.body.product_price - Preu del producte
 * @returns {Object} - Element de cantina creat
 * @returns {number} Object.id - Identificador de l'element creat
 * @returns {string} Object.product_name - Nom del producte
 * @returns {number} Object.product_price - Preu del producte
 * @returns {boolean} Object.product_enabled - Indica si el producte
està disponible (true per defecte)
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { product_name, product_price } = req.body;
          const canteenItem = await CanteenItem.create({ product_name,
product_price, product_enabled: true });
        res.status(201).json(canteenItem);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza un element de cantina per ID
 * @route PUT /canteenItem/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'element de cantina
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} [req.body.product_name] - Nom del producte
 * @param {number} [req.body.product_price] - Preu del producte
 * @param {boolean} [req.body.product_enabled] - Indica si el producte
està disponible
 * @returns {Object} - Resultat de l'actualització
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
            const { product_name, product_price, product_enabled } =
req.body;
        const canteenItem = await CanteenItem.update(
            { product_name, product_price, product_enabled },
            { where: { id: req.params.id } }
        );
        if (!canteenItem) {
```

```
            return res.status(404).json({ message: "Element de cantina
no trobat" });
        }
        res.json(canteenItem);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina un element de cantina per ID
 * @route DELETE /canteenItem/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'element de cantina
 * @returns {Object} message - Missatge de confirmació d'eliminació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const canteenItem = await CanteenItem.destroy({ where: { id:
req.params.id } });
        if (!canteenItem) {
            return res.status(404).json({ message: "Element de cantina
no trobat" });
        }
        res.json({ message: "Element de cantina eliminat correctament"
});
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté tots els elements de cantina habilitats
 * @route GET /canteenItem/allItems/enabled
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista d'elements de cantina habilitats
 * @returns {number} Object[].id - Identificador de l'element
 * @returns {string} Object[].product_name - Nom del producte
 * @returns {number} Object[].product_price - Preu del producte
 * @returns {boolean} Object[].product_enabled - Sempre és true en
aquesta ruta
 */
router.get("/allItems/enabled", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const canteenItems = await CanteenItem.findAll({ where: {
product_enabled: true } });
        res.json(canteenItems);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté tots els elements de cantina deshabilitats
 * @route GET /canteenItem/allItems/disabled
```

```
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista d'elements de cantina deshabilitats
 * @returns {number} Object[].id - Identificador de l'element
 * @returns {string} Object[].product_name - Nom del producte
 * @returns {number} Object[].product_price - Preu del producte
 * @returns {boolean} Object[].product_enabled - Sempre és false en aquesta ruta
 */
router.get("/allItems/disabled", verifyTokenMiddleware, async (req, res) => {
    try {
            const canteenItems = await CanteenItem.findAll({ where: { product_enabled: false } });
        res.json(canteenItems);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Course

```
/**
 * Obté tots els cursos
 * @route GET /course/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de tots els cursos
 * @returns {number} Object[].id - Identificador del curs
 * @returns {string} Object[].course_name - Nom del curs
 * @returns {number} Object[].course_hours_available - Hores disponibles del curs
 * @returns {string} Object[].course_description - Descripció del curs
 * @returns {number} Object[].teacher_id - Identificador del professor assignat al curs
 * @returns {Date} Object[].createdAt - Data de creació del curs
 * @returns {Date} Object[].updatedAt - Data d'última actualització del curs
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const courses = await Course.findAll();
    res.json(courses);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté un curs per ID
 * @route GET /course/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador del curs
```

```
 * @returns {Object} - Detalls del curs
 * @returns {number} Object.id - Identificador del curs
 * @returns {string} Object.course_name - Nom del curs
 * @returns {number} Object.course_hours_available - Hores disponibles
del curs
 * @returns {string} Object.course_description - Descripció del curs
 * @returns {number} Object.teacher_id - Identificador del professor
assignat al curs
 * @returns {Date} Object.createdAt - Data de creació del curs
 * @returns {Date} Object.updatedAt - Data d'última actualització del
curs
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const course = await Course.findByPk(req.params.id);
    res.json(course);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Crea un nou curs
 * @route POST /course/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {Object} req.body - Dades del nou curs
 * @param {string} req.body.course_name - Nom del curs
 * @param {number} req.body.course_hours_available - Hores disponibles
del curs
 * @param {string} req.body.course_description - Descripció del curs
 * @param {number} [req.body.teacher_id] - Identificador del professor
assignat al curs (opcional)
 * @returns {Object} - Curs creat
 * @returns {number} Object.id - Identificador del curs creat
 * @returns {string} Object.course_name - Nom del curs
 * @returns {number} Object.course_hours_available - Hores disponibles
del curs
 * @returns {string} Object.course_description - Descripció del curs
 * @returns {number} Object.teacher_id - Identificador del professor
assignat al curs
 * @returns {Date} Object.createdAt - Data de creació del curs
 * @returns {Date} Object.updatedAt - Data d'última actualització del
curs
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const {
      course_name,
      course_hours_available,
      course_description,
      teacher_id
    } = req.body;

    if (!course_name || !course_hours_available || !course_description)
{
      return res.status(400).json({ message: "Falten camps obligatoris"
});
```

```
    }

     const courseExists = await Course.findOne({ where: { course_name } });
    if (courseExists) {
       return res.status(400).json({ message: "Ja existeix un curs amb aquest nom" });
    }

    if (teacher_id) {
      const isTeacher = await checkIfUserIsTeacher(teacher_id);
      if (!isTeacher) {
            return res.status(400).json({ message: "L'usuari no és un professor" });
      }
    }

    const course = await Course.create({
      course_name,
      course_hours_available,
      course_description,
      teacher_id
    });
    res.status(201).json(course);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Actualitza un curs per ID
 * @route PUT /course/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador del curs
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} [req.body.course_name] - Nom del curs
   * @param {number} [req.body.course_hours_available] - Hores disponibles del curs
 * @param {string} [req.body.course_description] - Descripció del curs
 * @param {number} [req.body.teacher_id] - Identificador del professor assignat al curs
 * @returns {Object} - Curs actualitzat
 * @returns {number} Object.id - Identificador del curs
 * @returns {string} Object.course_name - Nom actualitzat del curs
 * @returns {number} Object.course_hours_available - Hores disponibles actualitzades del curs
   * @returns {string} Object.course_description - Descripció actualitzada del curs
 * @returns {number} Object.teacher_id - Identificador actualitzat del professor assignat al curs
 * @returns {Date} Object.createdAt - Data de creació del curs
 * @returns {Date} Object.updatedAt - Data d'última actualització del curs
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const {
```

```
      course_name,
      course_hours_available,
      course_description,
      teacher_id
    } = req.body;

    const course = await Course.findByPk(req.params.id);
    if (!course) {
      return res.status(404).json({ message: "Curs no trobat" });
    }

    if (teacher_id && teacher_id !== course.teacher_id) {
      const isTeacher = await checkIfUserIsTeacher(teacher_id);
      if (!isTeacher) {
            return res.status(400).json({ message: "L'usuari no és un
professor" });
      }
    }

    await course.update({
      course_name,
      course_hours_available,
      course_description,
      teacher_id
    });
    res.json(course);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Elimina un curs per ID
 * @route DELETE /course/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador del curs
 * @returns {Object} message - Missatge de confirmació d'eliminació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const course = await Course.findByPk(req.params.id);
    if (!course) {
      return res.status(404).json({ message: "Curs no trobat" });
    }
    await course.destroy();
    res.json({ message: "Curs eliminat correctament" });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté estadístiques dels cursos
 * @route GET /course/stats/count
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object} - Estadístiques dels cursos
```

```
 * @returns {number} Object.total - Nombre total de cursos
 * @returns {number} Object.createdThisMonth - Nombre de cursos creats
aquest mes
 */
router.get("/stats/count", verifyTokenMiddleware, async (req, res) => {
  try {
    const totalCourses = await Course.count();
    const today = new Date();
              const   startOfMonth   =   new   Date(today.getFullYear(),
today.getMonth(), 1);
    const endOfMonth = new Date(
      today.getFullYear(),
      today.getMonth() + 1,
      0,
      23,
      59,
      59,
      999
    );

    const thisMonthCourses = await Course.count({
      where: {
        createdAt: {
          [Op.between]: [startOfMonth, endOfMonth],
        },
      },
    });

    res.json({
      total: totalCourses,
      createdThisMonth: thisMonthCourses,
    });
  } catch (error) {
    console.error("Error al obtenir estadístiques de cursos:", error);
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté tots els cursos d'un professor específic
 * @route GET /course/teacher/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador del professor
 * @returns {Object[]} - Llista de cursos assignats al professor
 * @returns {number} Object[].id - Identificador del curs
 * @returns {string} Object[].course_name - Nom del curs
   *   @returns   {number}   Object[].course_hours_available   -   Hores
disponibles del curs
 * @returns {string} Object[].course_description - Descripció del curs
 * @returns {number} Object[].teacher_id - Identificador del professor
(igual al paràmetre id)
 * @returns {Date} Object[].createdAt - Data de creació del curs
 * @returns {Date} Object[].updatedAt - Data d'última actualització del
curs
 */
router.get("/teacher/:id", verifyTokenMiddleware, async (req, res) => {
  try {
```

```
    const courses = await Course.findAll({
      where: { course_teacher_id: req.params.id },
    });
    if (!courses) {
        return res.status(404).json({ message: "No s'han trobat cursos"
});
    }
    res.json(courses);
  }
  catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté el curs més recent
 * @function getLatestCourse
 * @returns {Object|null} - El curs més recent o null si no n'hi ha cap
 * @returns {number} Object.id - Identificador del curs
 * @returns {string} Object.course_name - Nom del curs
 * @returns {number} Object.course_hours_available - Hores disponibles
del curs
 * @returns {string} Object.course_description - Descripció del curs
 * @returns {number} Object.teacher_id - Identificador del professor
assignat al curs
 * @returns {Date} Object.createdAt - Data de creació del curs
 * @returns {Date} Object.updatedAt - Data d'última actualització del
curs
 */
export async function getLatestCourse() {
  try {
    const latestCourse = await Course.findOne({
      order: [['createdAt', 'DESC']]
    });

    return latestCourse;
  } catch (error) {
    console.error("Error al obtenir curs recent:", error);
    throw error;
  }
}

/**
 * Comprova si un usuari és professor
 * @function checkIfUserIsTeacher
 * @param {number} teacher_id - Identificador de l'usuari a comprovar
 * @returns {boolean} - True si l'usuari és professor (user_type ===
1), False si no
 */
async function checkIfUserIsTeacher(teacher_id) {
  const user = await User.findByPk(teacher_id);
  if (!user) {
    return false;
  }
  return user.user_type == 1;
}
```

Department

```
/**
 * Obté tots els departaments
 * @route GET /department/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de tots els departaments
 * @returns {number} Object[].id - Identificador del departament
 * @returns {string} Object[].name - Nom del departament
 * @returns {Date} Object[].createdAt - Data de creació del departament
 * @returns {Date} Object[].updatedAt - Data d'última actualització del departament
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const departments = await Department.findAll();
        res.json(departments);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté un departament per ID
 * @route GET /department/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador del departament
 * @returns {Object} - Detalls del departament
 * @returns {number} Object.id - Identificador del departament
 * @returns {string} Object.name - Nom del departament
 * @returns {Date} Object.createdAt - Data de creació del departament
 * @returns {Date} Object.updatedAt - Data d'última actualització del departament
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const department = await Department.findByPk(req.params.id);
        res.json(department);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea un nou departament
 * @route POST /department/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {Object} req.body - Dades del nou departament
 * @param {string} req.body.name - Nom del departament
 * @returns {Object} - Departament creat
 * @returns {number} Object.id - Identificador del departament creat
 * @returns {string} Object.name - Nom del departament
 * @returns {Date} Object.createdAt - Data de creació del departament
```

```
 * @returns {Date} Object.updatedAt - Data d'última actualització del
departament
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { name } = req.body;
        const department = await Department.create({ name });
        res.json(department);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza un departament per ID
 * @route PUT /department/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador del departament
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} req.body.name - Nou nom del departament
 * @returns {Object} - Resultat de l'actualització
 * @returns {number} Object[0] - Nombre de files afectades (1 si s'ha
actualitzat correctament, 0 si no)
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { name } = req.body;
         const department = await Department.update({ name }, { where: {
id: req.params.id } });
        res.json(department);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina un departament per ID
 * @route DELETE /department/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador del departament
 * @returns {Object} message - Missatge de confirmació d'eliminació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const department = await Department.findByPk(req.params.id);
        if (!department) {
                return res.status(404).json({ message: "Departament no
trobat" });
        } else {
            await Department.destroy({ where: { id: req.params.id } });
            res.json({ message: "Departament eliminat correctament" });
        }
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Grade

```
/**
 * Obté totes les qualificacions
 * @route GET /grade/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de totes les qualificacions
 * @returns {number} Object[].id - Identificador de la qualificació
 * @returns {number} Object[].grade - Nota (valor entre 0 i 10)
 * @returns {number} Object[].user_id - Identificador de l'usuari
 * @returns {number} Object[].task_id - Identificador de la tasca
 * @returns {Date} Object[].createdAt - Data de creació de la qualificació
 * @returns {Date} Object[].updatedAt - Data d'última actualització de la qualificació
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const grades = await Grade.findAll();
    res.json(grades);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté una qualificació per ID
 * @route GET /grade/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de la qualificació
 * @returns {Object} - Detalls de la qualificació
 * @returns {number} Object.id - Identificador de la qualificació
 * @returns {number} Object.grade - Nota (valor entre 0 i 10)
 * @returns {number} Object.user_id - Identificador de l'usuari
 * @returns {number} Object.task_id - Identificador de la tasca
 * @returns {Date} Object.createdAt - Data de creació de la qualificació
 * @returns {Date} Object.updatedAt - Data d'última actualització de la qualificació
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const grade = await Grade.findByPk(req.params.id);
    res.json(grade);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les qualificacions d'una tasca específica
 * @route GET /grade/getAllGradesFromTask/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de la tasca
 * @returns {Object[]} - Llista de qualificacions de la tasca
```

```
 * @returns {number} Object[].id - Identificador de la qualificació
 * @returns {number} Object[].grade - Nota (valor entre 0 i 10)
 * @returns {number} Object[].user_id - Identificador de l'usuari
  * @returns {number} Object[].task_id - Identificador de la tasca (igual al paràmetre id)
  * @returns {Date} Object[].createdAt - Data de creació de la qualificació
 * @returns {Date} Object[].updatedAt - Data d'última actualització de la qualificació
 */
router.get("/getAllGradesFromTask/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    let grades = [];
    grades = await Grade.findAll({
      where: { task_id: req.params.id },
    });
    return res.json(grades);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les qualificacions d'un usuari específic
 * @route GET /grade/getAllGradesFromUser/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'usuari
 * @returns {Object[]} - Llista de qualificacions simplificades de l'usuari
 * @returns {number} Object[].id - Identificador de la qualificació
 * @returns {number} Object[].grade - Nota (valor entre 0 i 10)
 */
router.get("/getAllGradesFromUser/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const user = await Grade.findByPk(req.params.id, {
      include: [
        {
          model: Grade,
          as: "grades",
          attributes: ["id", "grade"],
        },
      ],
    });
    if (!user) {
      return res.status(404).json({ message: "Usuari no trobat" });
    }
    const grades = user.grades.map((grade) => {
      return {
        id: grade.id,
        grade: grade.grade,
      };
    });
    res.json(grades);
  } catch (error) {
    res.status(500).json({ message: error.message });
```

```
  }
});

/**
 * Crea o actualitza una qualificació
 * @route POST /grade/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {Object} req.body - Dades de la qualificació
 * @param {number} req.body.grade - Nota (ha d'estar entre 0 i 10)
 * @param {number} req.body.user_id - Identificador de l'usuari
 * @param {number} req.body.task_id - Identificador de la tasca
 * @returns {Object} - Qualificació creada o resultat de l'actualització
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
  try {
    if (req.body.grade < 0 || req.body.grade > 10) {
      return res
        .status(400)
        .json({ message: "La nota ha d'estar entre 0 i 10" });
    } else {
      let grade = {};
      if (await checkIfGradeAlreadyExists(req.body.user_id, req.body.task_id)) {
        grade = await Grade.update(req.body, {
          where: { user_id: req.body.user_id, task_id: req.body.task_id },
        });
      } else {
        grade = await Grade.create(req.body);
      }
      res.json(grade);
    }
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Actualitza una qualificació per ID
 * @route PUT /grade/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de la qualificació
 * @param {Object} req.body - Dades a actualitzar
 * @param {number} req.body.grade - Nova nota (ha d'estar entre 0 i 10)
 * @param {number} [req.body.user_id] - Identificador de l'usuari (opcional)
 * @param {number} [req.body.task_id] - Identificador de la tasca (opcional)
 * @returns {Object} - Qualificació actualitzada
 * @returns {number} Object.id - Identificador de la qualificació
 * @returns {number} Object.grade - Nova nota
 * @returns {number} Object.user_id - Identificador de l'usuari
 * @returns {number} Object.task_id - Identificador de la tasca
 * @returns {Date} Object.createdAt - Data de creació de la qualificació
```

```
 * @returns {Date} Object.updatedAt - Data d'última actualització de la
qualificació
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    if (req.body.grade < 0 || req.body.grade > 10) {
      return res
        .status(400)
        .json({ message: "La nota ha d'estar entre 0 i 10" });
    }
    const grade = await Grade.update(req.body, {
      where: { id: req.params.id },
    });
    if (!grade) {
      return res.status(404).json({ message: "Nota no trobada" });
    }
    const updatedGrade = await Grade.findByPk(req.params.id);
    if (!updatedGrade) {
      return res.status(404).json({ message: "Nota no trobada" });
    }
    res.json(updatedGrade);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les qualificacions d'un usuari en un curs específic
 * @route GET /grade/getAllGradesFromUserAndCourse/:userId/:courseId
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} userId - Identificador de l'usuari
 * @param {number} courseId - Identificador del curs
 * @returns {Object[]} - Llista de qualificacions de l'usuari en el
curs especificat
 * @returns {number} Object[].id - Identificador de la qualificació
 * @returns {number|string} Object[].grade - Nota o "Sense qualificar"
si no té nota
 * @returns {number} Object[].task_id - Identificador de la tasca
 * @returns {string} Object[].task_name - Nom de la tasca
 */
router.get(
                    "/getAllGradesFromUserAndCourse/:userId/:courseId",
verifyTokenMiddleware,
  async (req, res) => {
    try {
      const { userId, courseId } = req.params;
      let arrangedGrades = [];
      const grades = await Grade.findAll({
        where: {
          user_id: userId,
        },
      });

      await Promise.all(
        grades.map(async (grade) => {
          let auxTask = await Task.findOne({
            where: { id: grade.task_id, course_id: courseId },
```

```
          });
          if (auxTask) {
            arrangedGrades.push({
              id: grade.id,
              grade: grade.grade ?? "Sense qualificar",
              task_id: grade.task_id,
              task_name: auxTask.task_name,
            });
          }
        })
      );

      res.json(arrangedGrades);
    } catch (error) {
      res
        .status(500)
        .json({ message: error.message });
    }
  }
);

/**
 * Elimina una qualificació per ID
 * @route DELETE /grade/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de la qualificació
 * @returns {Object} message - Missatge de confirmació d'eliminació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const grade = await Grade.findByPk(req.params.id);
    if (!grade) {
      return res.status(404).json({ message: "Nota no trobada" });
    } else {
      await Grade.destroy({ where: { id: req.params.id } });
      res.json({ message: "Nota eliminada correctament" });
    }
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Comprova si ja existeix una qualificació per a un usuari i tasca específics
 * @function checkIfGradeAlreadyExists
 * @param {number} user_id - Identificador de l'usuari
 * @param {number} task_id - Identificador de la tasca
 * @returns {boolean} - True si la qualificació existeix, False si no
 */
async function checkIfGradeAlreadyExists(user_id, task_id) {
  let auxGrade = await Grade.findOne({
    where: { user_id: user_id, task_id: task_id },
  });
  if (auxGrade) {
    return true;
  } else {
```

```
    return false;
  }
}
```

Lost Objects

```
/**
 * Obté tots els objectes perduts ordenats per data de creació (més recents primer)
 * @route GET /lostObject/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de tots els objectes perduts
 * @returns {number} Object[].id - Identificador de l'objecte perdut
 * @returns {string} Object[].title - Títol de l'objecte perdut
 * @returns {string} Object[].description - Descripció de l'objecte perdut
 * @returns {string|null} Object[].image - Ruta a la imatge de l'objecte perdut (si existeix)
 * @returns {number} Object[].user_id - Identificador de l'usuari que ha creat l'anunci
 * @returns {Date} Object[].createdAt - Data de creació de l'anunci
 * @returns {Date} Object[].updatedAt - Data d'última actualització de l'anunci
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    const lostObjectsList = await LostObjects.findAll({ order: [["createdAt", "DESC"]] });
  res.json(lostObjectsList);
});

/**
 * Obté un objecte perdut per ID
 * @route GET /lostObject/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @returns {Object} - Detalls de l'objecte perdut
 * @returns {number} Object.id - Identificador de l'objecte perdut
 * @returns {string} Object.title - Títol de l'objecte perdut
 * @returns {string} Object.description - Descripció de l'objecte perdut
 * @returns {string|null} Object.image - Ruta a la imatge de l'objecte perdut (si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha creat l'anunci
 * @returns {Date} Object.createdAt - Data de creació de l'anunci
 * @returns {Date} Object.updatedAt - Data d'última actualització de l'anunci
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
    const lostObject = await LostObjects.findOne({ where: { id } });
    if (!lostObject) {
```

```
      return res.status(404).json({ message: "Objecte perdut no trobat"
});
    }
    res.json(lostObject);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Crea un nou objecte perdut amb possible imatge adjunta
 * @route POST /lostObject/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @middleware upload.single('image') - Processa la càrrega de la
imatge
 * @param {Object} req.body - Dades del nou objecte perdut
 * @param {string} req.body.title - Títol de l'objecte perdut
 * @param {string} req.body.description - Descripció de l'objecte
perdut
 * @param {number} req.body.user_id - Identificador de l'usuari que
crea l'anunci
 * @param {File} [req.file] - Fitxer d'imatge adjunt (opcional)
 * @returns {Object} - Objecte perdut creat
 * @returns {number} Object.id - Identificador de l'objecte perdut
creat
 * @returns {string} Object.title - Títol de l'objecte perdut
 * @returns {string} Object.description - Descripció de l'objecte
perdut
 * @returns {string|null} Object.image - Ruta a la imatge carregada (si
existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat l'anunci
 * @returns {Date} Object.createdAt - Data de creació de l'anunci
 * @returns {Date} Object.updatedAt - Data d'última actualització de
l'anunci
 */
router.post("/", verifyTokenMiddleware, upload.single('image'), async
(req, res) => {
  try {
    const { title, description, user_id } = req.body;
    const image = req.file ? req.file.path : null;

    if (!title || !description || !user_id) {
      return res.status(400).json({ message: "Falten camps obligatoris"
});
    }

    const newLostObject = await LostObjects.create({
      title,
      description,
      image,
      user_id: Number(user_id) || 1,
    });

    res.status(201).json(newLostObject);
  } catch (error) {
    console.error("Error en crear objecte perdut:", error);
```

```
    res.status(500).json({ message: error.message });
  }
});

/**
 * Actualitza un objecte perdut per ID
 * @route PUT /lostObject/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} [req.body.title] - Nou títol de l'objecte perdut
 * @param {string} [req.body.description] - Nova descripció de l'objecte perdut
 * @param {string} [req.body.image] - Nova ruta a la imatge de l'objecte perdut
 * @param {number} [req.body.user_id] - Nou usuari propietari de l'anunci
 * @returns {Object} - Objecte perdut actualitzat
 * @returns {number} Object.id - Identificador de l'objecte perdut
 * @returns {string} Object.title - Títol actualitzat de l'objecte perdut
 * @returns {string} Object.description - Descripció actualitzada de l'objecte perdut
 * @returns {string|null} Object.image - Ruta actualitzada a la imatge de l'objecte perdut
 * @returns {number} Object.user_id - Identificador de l'usuari propietari de l'anunci
 * @returns {Date} Object.createdAt - Data de creació de l'anunci
 * @returns {Date} Object.updatedAt - Data d'última actualització de l'anunci
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
    const lostObject = await LostObjects.findOne({ where: { id } });
    if (!lostObject) {
      return res.status(404).json({ message: "Objecte perdut no trobat" });
    }
    await lostObject.update(req.body);
    res.json(lostObject);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Elimina un objecte perdut per ID
 * @route DELETE /lostObject/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @returns {null} - Resposta buida amb codi d'estat 204 (No Content)
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
```

```
    const lostObject = await LostObjects.findOne({ where: { id } });
    if (!lostObject) {
       return res.status(404).json({ message: "Objecte perdut no trobat"
});
    }
    await lostObject.destroy();
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les respostes d'un objecte perdut específic
 * @route GET /lostObject/:id/responses
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @returns {Object[]} - Llista de respostes a l'objecte perdut
 * @returns {number} Object[].id - Identificador de la resposta
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha fet la resposta
  * @returns {number} Object[].lostAndFound_id - Identificador de
l'objecte perdut (igual al paràmetre id)
 * @returns {string} Object[].comment - Contingut de la resposta
 * @returns {Date} Object[].createdAt - Data de creació de la resposta
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
la resposta
 */
router.get("/:id/responses", verifyTokenMiddleware, async (req, res) =>
{
    const { id } = req.params;
    try {
        const responses = await Response.findAll({
            where: { lostAndFound_id: id },
            order: [["createdAt", "DESC"]]
        });
        res.json(responses);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea una nova resposta per a un objecte perdut
 * @route POST /lostObject/:id/responses
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @param {Object} req.body - Dades de la nova resposta
 * @param {number} req.body.user_id - Identificador de l'usuari que fa
la resposta
 * @param {string} req.body.comment - Contingut de la resposta
 * @returns {Object} - Resposta creada
 * @returns {number} Object.id - Identificador de la resposta creada
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
fet la resposta
```

```
   *   @returns   {number}   Object.lostAndFound_id   -   Identificador   de
l'objecte perdut
 * @returns {string} Object.comment - Contingut de la resposta
 * @returns {Date} Object.createdAt - Data de creació de la resposta
 * @returns {Date} Object.updatedAt - Data d'última actualització de la
resposta
 */
router.post("/:id/responses",  verifyTokenMiddleware,  async  (req,  res)
=> {
    const { id } = req.params;
    try {
        const { user_id, comment } = req.body;

        if (!user_id || !comment) {
              return res.status(400).json({ message: "user_id i comment
són obligatoris" });
        }

        const newResponse = await Response.create({
            user_id,
            lostAndFound_id: id,
            comment
        });
        res.status(201).json(newResponse);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina una resposta específica d'un objecte perdut
 * @route DELETE /lostObject/:id/responses/:responseId
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'objecte perdut
 * @param {number} responseId - Identificador de la resposta a eliminar
 * @returns {null} - Resposta buida amb codi d'estat 204 (No Content)
 */
router.delete("/:id/responses/:responseId",       verifyTokenMiddleware,
async (req, res) => {
    const { id, responseId } = req.params;
    try {
        const response = await Response.findOne({
            where: {
                id: responseId,
                lostAndFound_id: id
            }
        });

        if (!response) {
                  return  res.status(404).json({  message:  "Resposta  no
trobada" });
        }

        await response.destroy();
        res.status(204).send();
    } catch (error) {
        res.status(500).json({ message: error.message });
```

```
    }
});

/**
 * Obté estadístiques dels objectes perduts
 * @route GET /lostObject/stats/count
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object} - Estadístiques d'objectes perduts
 * @returns {number} Object.total - Nombre total d'objectes perduts
 * @returns {number} Object.reportedToday - Nombre d'objectes perduts reportats avui
 */
router.get("/stats/count", verifyTokenMiddleware, async (req, res) => {
  try {
    const totalLostObjects = await LostObjects.count();

    const today = new Date();
    const startOfDay = new Date(today.setHours(0, 0, 0, 0));
    const endOfDay = new Date(today.setHours(23, 59, 59, 999));

    const todayLostObjects = await LostObjects.count({
      where: {
        createdAt: {
          [Op.between]: [startOfDay, endOfDay]
        }
      }
    });

    res.json({
      total: totalLostObjects,
      reportedToday: todayLostObjects
    });
  } catch (error) {
    console.error("Error al obtenir estadístiques d'objectes perduts:", error);
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté l'objecte perdut més recent
 * @function getLatestLostObject
 * @returns {Object|null} - L'objecte perdut més recent o null si no n'hi ha cap
 * @returns {number} Object.id - Identificador de l'objecte perdut
 * @returns {string} Object.title - Títol de l'objecte perdut
 * @returns {string} Object.description - Descripció de l'objecte perdut
 * @returns {string|null} Object.image - Ruta a la imatge de l'objecte perdut (si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha creat l'anunci
 * @returns {Date} Object.createdAt - Data de creació de l'anunci
 * @returns {Date} Object.updatedAt - Data d'última actualització de l'anunci
 */
export async function getLatestLostObject() {
```

```
    try {
        const latestLostObject = await LostObjects.findOne({
            order: [['createdAt', 'DESC']]
        });

        return latestLostObject;
    } catch (error) {
            console.error("Error al obtenir objecte perdut recent:",
error);
        throw error;
    }
}
```

Reports

```
/**
  * Obté tots els informes amb dades d'usuari, usuari assignat i
habitació
 * @route GET /report/
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista de tots els informes amb relacions
 * @returns {number} Object[].id - Identificador de l'informe
 * @returns {string} Object[].report - Descripció de l'informe
  * @returns {string|null} Object[].image - Ruta a la imatge de
l'informe (si existeix)
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha creat l'informe
 * @returns {number} Object[].room_id - Identificador de l'habitació de
l'informe
  * @returns {number|null} Object[].user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
 * @returns {string} Object[].status - Estat de l'informe ('pending',
'revising', 'revised')
  * @returns {boolean} Object[].finished - Indica si l'informe està
finalitzat
 * @returns {string|null} Object[].note - Nota associada a l'informe
(si existeix)
 * @returns {Object} Object[].User - Dades de l'usuari creador
 * @returns {number} Object[].User.id - Identificador de l'usuari
 * @returns {string} Object[].User.name - Nom de l'usuari
 * @returns {string} Object[].User.email - Email de l'usuari
  * @returns {Object|null} Object[].AssignedUser - Dades de l'usuari
assignat (si existeix)
  * @returns {number} Object[].AssignedUser.id - Identificador de
l'usuari assignat
  * @returns {string} Object[].AssignedUser.name - Nom de l'usuari
assignat
 * @returns {string} Object[].AssignedUser.email - Email de l'usuari
assignat
 * @returns {Object} Object[].Room - Dades de l'habitació
 * @returns {number} Object[].Room.id - Identificador de l'habitació
 * @returns {string} Object[].Room.room_name - Nom de l'habitació
 * @returns {Date} Object[].createdAt - Data de creació de l'informe
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
l'informe
```

```
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const reports = await Reports.findAll({
      include: [
        {
          model: User,
          as: "User",
          attributes: ["id", "name", "email"],
        },
        {
          model: User,
          as: "AssignedUser",
          attributes: ["id", "name", "email"],
        },
        {
          model: Room,
          attributes: ["id", "room_name"],
        },
      ],
    });
    res.json(reports);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté tots els informes d'un usuari específic
 * @route GET /report/user/:user_id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} user_id - Identificador de l'usuari
 * @returns {Object[]} - Llista d'informes de l'usuari
 * @returns {number} Object[].id - Identificador de l'informe
 * @returns {string} Object[].report - Descripció de l'informe
  * @returns {string|null} Object[].image - Ruta a la imatge de
l'informe (si existeix)
  * @returns {number} Object[].user_id - Identificador de l'usuari
(igual al paràmetre user_id)
 * @returns {number} Object[].room_id - Identificador de l'habitació de
l'informe
  * @returns {number|null} Object[].user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
  * @returns {string} Object[].status - Estat de l'informe ('pending',
'revising', 'revised')
  * @returns {boolean} Object[].finished - Indica si l'informe està
finalitzat
  * @returns {string|null} Object[].note - Nota associada a l'informe
(si existeix)
 * @returns {Date} Object[].createdAt - Data de creació de l'informe
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
l'informe
 */
router.get("/user/:user_id", verifyTokenMiddleware, async (req, res) =>
{
  const { user_id } = req.params;
  try {
```

```
    const reports = await Reports.findAll({ where: { user_id } });
    if (!reports) {
      return res
        .status(404)
        .json({ message: "No s'han trobat informes per a aquest usuari"
});
    }
    res.json(reports);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté tots els informes d'una habitació específica
 * @route GET /report/room/:room_id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} room_id - Identificador de l'habitació
 * @returns {Object[]} - Llista d'informes de l'habitació
 * @returns {number} Object[].id - Identificador de l'informe
 * @returns {string} Object[].report - Descripció de l'informe
 * @returns {string|null} Object[].image - Ruta a la imatge de
l'informe (si existeix)
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha creat l'informe
 * @returns {number} Object[].room_id - Identificador de l'habitació
(igual al paràmetre room_id)
 * @returns {number|null} Object[].user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
 * @returns {string} Object[].status - Estat de l'informe ('pending',
'revising', 'revised')
 * @returns {boolean} Object[].finished - Indica si l'informe està
finalitzat
 * @returns {string|null} Object[].note - Nota associada a l'informe
(si existeix)
 * @returns {Date} Object[].createdAt - Data de creació de l'informe
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
l'informe
 */
router.get("/room/:room_id", verifyTokenMiddleware, async (req, res) =>
{
  const { room_id } = req.params;
  try {
    const reports = await Reports.findAll({ where: { room_id } });
    if (!reports) {
      return res
        .status(404)
            .json({ message: "No s'han trobat informes per a aquesta
habitació" });
    }
    res.json(reports);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
```

```
 * Obté tots els informes finalitzats
 * @route GET /report/finished
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista d'informes finalitzats
 * @returns {number} Object[].id - Identificador de l'informe
 * @returns {string} Object[].report - Descripció de l'informe
  * @returns {string|null} Object[].image - Ruta a la imatge de
l'informe (si existeix)
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha creat l'informe
 * @returns {number} Object[].room_id - Identificador de l'habitació de
l'informe
  * @returns {number|null} Object[].user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
 * @returns {string} Object[].status - Estat de l'informe ('revised')
 * @returns {boolean} Object[].finished - Sempre és true en aquesta
ruta
 * @returns {string|null} Object[].note - Nota associada a l'informe
(si existeix)
 * @returns {Date} Object[].createdAt - Data de creació de l'informe
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
l'informe
 */
router.get("/finished", verifyTokenMiddleware, async (req, res) => {
  try {
     const reports = await Reports.findAll({ where: { finished: true }
});
    if (!reports) {
      return res
        .status(404)
        .json({ message: "No s'han trobat informes acabats" });
    }
    res.json(reports);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté tots els informes no finalitzats
 * @route GET /report/not-finished
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista d'informes no finalitzats
 * @returns {number} Object[].id - Identificador de l'informe
 * @returns {string} Object[].report - Descripció de l'informe
  * @returns {string|null} Object[].image - Ruta a la imatge de
l'informe (si existeix)
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha creat l'informe
 * @returns {number} Object[].room_id - Identificador de l'habitació de
l'informe
  * @returns {number|null} Object[].user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
 * @returns {string} Object[].status - Estat de l'informe ('pending' o
'revising')
```

```
 * @returns {boolean} Object[].finished - Sempre és false en aquesta
ruta
 * @returns {string|null} Object[].note - Nota associada a l'informe
(si existeix)
 * @returns {Date} Object[].createdAt - Data de creació de l'informe
 * @returns {Date} Object[].updatedAt - Data d'última actualització de
l'informe
 */
router.get("/not-finished", verifyTokenMiddleware, async (req, res) =>
{
  try {
     const reports = await Reports.findAll({ where: { finished: false }
});
    if (!reports) {
      return res
        .status(404)
        .json({ message: "No s'han trobat informes no acabats" });
    }
    res.json(reports);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté un informe per ID
 * @route GET /report/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'informe
 * @returns {Object} - Detalls de l'informe
 * @returns {number} Object.id - Identificador de l'informe
 * @returns {string} Object.report - Descripció de l'informe
 * @returns {string|null} Object.image - Ruta a la imatge de l'informe
(si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat l'informe
 * @returns {number} Object.room_id - Identificador de l'habitació de
l'informe
 * @returns {number|null} Object.user_assigned - Identificador de
l'usuari assignat a l'informe (si existeix)
 * @returns {string} Object.status - Estat de l'informe ('pending',
'revising', 'revised')
 * @returns {boolean} Object.finished - Indica si l'informe està
finalitzat
 * @returns {string|null} Object.note - Nota associada a l'informe (si
existeix)
 * @returns {Date} Object.createdAt - Data de creació de l'informe
 * @returns {Date} Object.updatedAt - Data d'última actualització de
l'informe
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
    const report = await Reports.findOne({ where: { id } });
    if (!report) {
      return res.status(404).json({ message: "Informe no trobat" });
    }
```

```
    res.json(report);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Crea un nou informe amb possible imatge adjunta
 * @route POST /report/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @middleware upload.single("image") - Processa la càrrega de la imatge
 * @param {Object} req.body.data - Dades del nou informe (en format JSON)
 * @param {number} req.body.data.user_id - Identificador de l'usuari que crea l'informe
 * @param {string} req.body.data.report - Descripció de l'informe
 * @param {number} req.body.data.room_id - Identificador de l'habitació
 * @param {File} [req.file] - Fitxer d'imatge adjunt (opcional)
 * @returns {Object} - Informe creat
 * @returns {number} Object.id - Identificador de l'informe creat
 * @returns {string} Object.report - Descripció de l'informe
 * @returns {string|null} Object.image - Ruta a la imatge carregada (si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha creat l'informe
 * @returns {number} Object.room_id - Identificador de l'habitació de l'informe
 * @returns {string} Object.status - Estat inicial de l'informe ('pending' per defecte)
 * @returns {boolean} Object.finished - Indica si l'informe està finalitzat (false per defecte)
 * @returns {Date} Object.createdAt - Data de creació de l'informe
 * @returns {Date} Object.updatedAt - Data d'última actualització de l'informe
 */
router.post("/", verifyTokenMiddleware, upload.single("image"), async (req, res) => {
  try {
    const reportData = JSON.parse(req.body.data);
    const { user_id, report, room_id } = reportData;
    const image = req.file ? req.file.path : null;

    if (!user_id || !report || !room_id) {
      return res
        .status(400)
        .json({ message: "user_id, report i room_id són obligatoris" });
    }

    const newReport = await Reports.create({
      user_id,
      report,
      image,
      room_id,
    });
```

```
    res.status(201).json(newReport);
  } catch (error) {
    console.error("Error en crear informe:", error);
    res.status(500).json({ message: error.message });
  }
});

/**
 * Actualitza un informe per ID i envia correu si l'estat canvia a
'revised'
 * @route PUT /report/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'informe
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} [req.body.report] - Nova descripció de l'informe
 * @param {string} [req.body.status] - Nou estat de l'informe
('pending', 'revising', 'revised')
 * @param {boolean} [req.body.finished] - Nou estat de finalització de
l'informe
 * @param {string} [req.body.note] - Nova nota associada a l'informe
 * @param {number} [req.body.user_assigned] - Nou usuari assignat a
l'informe
 * @returns {Object} - Informe actualitzat
 * @returns {number} Object.id - Identificador de l'informe
 * @returns {string} Object.report - Descripció actualitzada de
l'informe
 * @returns {string|null} Object.image - Ruta a la imatge de l'informe
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat l'informe
 * @returns {number} Object.room_id - Identificador de l'habitació de
l'informe
 * @returns {number|null} Object.user_assigned - Identificador
actualitzat de l'usuari assignat
 * @returns {string} Object.status - Estat actualitzat de l'informe
 * @returns {boolean} Object.finished - Estat actualitzat de
finalització de l'informe
 * @returns {string|null} Object.note - Nota associada actualitzada
 * @returns {Date} Object.createdAt - Data de creació de l'informe
 * @returns {Date} Object.updatedAt - Data d'última actualització de
l'informe
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
    const report = await Reports.findOne({ where: { id } });
    if (!report) {
      return res.status(404).json({ message: "Informe no trobat" });
    }
    await report.update(req.body);
    if (report.status === "revised") {
        const user = await User.findOne({ where: { id: report.user_id }
});
      let note = null;
      if (report.note) {
                  note = `Benvolgut/da ${user.name},\n\nEns complau
informar-lo/la que l'informe amb ID ${report.id} amb descripció:
${report.report}\n\n     ha estat revisat i s'ha completat amb
```

```
èxit.\n\nAtentament,\nL'equip de gestió d'informes. \n\nNota: ${report.note}`;

      } else {
                  note = `Benvolgut/da ${user.name},\n\nEns complau informar-lo/la que l'informe amb ID ${report.id}amb descripció: ${report.report}\n\n ha estat revisat i s'ha completat amb èxit.\n\nAtentament,\nL'equip de gestió d'informes.`;

      }
      const mailOptions = {
        from: process.env.EMAIL_USER,
        to: user.email,
        subject: "Informe Revisat",
        text: note,
      };
      transporter.sendMail(mailOptions, (error, info) => {
        if (error) {
          console.error("Error en enviar correu:", error);
        } else {
          console.log("Correu enviat:", info.response);
        }
      });
    }
    res.json(report);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Elimina un informe per ID i la seva imatge associada si existeix
 * @route DELETE /report/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'informe
 * @returns {null} - Resposta buida amb codi d'estat 204 (No Content)
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
    const report = await Reports.findOne({ where: { id } });

    if (!report) {
      return res.status(404).json({ message: "Informe no trobat" });
    }

    if (report.image) {
      const fs = await import('fs');
      const path = await import('path');

      const imagePath = path.default.join(process.cwd(), report.image);

      if (fs.default.existsSync(imagePath)) {
        fs.default.unlinkSync(imagePath);
      } else {
              console.log(`Imatge no trobada al sistema de fitxers: ${imagePath}`);
```

```
      }
    }

    await report.destroy();

    res.status(204).send();
  } catch (error) {
    console.error("Error en eliminar informe:", error);
    res.status(500).json({ message: error.message });
  }
});

/**
 * Assigna un usuari a un informe i retorna l'informe amb les relacions
carregades
 * @route PUT /report/:id/assign
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'informe
 * @param {Object} req.body - Dades d'assignació
 * @param {number} req.body.user_assigned - Identificador de l'usuari a
assignar
 * @returns {Object} - Informe actualitzat amb relacions carregades
 * @returns {number} Object.id - Identificador de l'informe
 * @returns {string} Object.report - Descripció de l'informe
 * @returns {string|null} Object.image - Ruta a la imatge de l'informe
(si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat l'informe
 * @returns {number} Object.room_id - Identificador de l'habitació de
l'informe
 * @returns {number} Object.user_assigned - Identificador de l'usuari
assignat
 * @returns {string} Object.status - Estat de l'informe
 * @returns {boolean} Object.finished - Indica si l'informe està
finalitzat
 * @returns {string|null} Object.note - Nota associada a l'informe (si
existeix)
 * @returns {Object} Object.User - Dades de l'usuari creador
 * @returns {Object} Object.AssignedUser - Dades de l'usuari assignat
 * @returns {Object} Object.Room - Dades de l'habitació
 * @returns {Date} Object.createdAt - Data de creació de l'informe
 * @returns {Date} Object.updatedAt - Data d'última actualització de
l'informe
 */
router.put("/:id/assign", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  const { user_assigned } = req.body;
  try {
    const report = await Reports.findOne({ where: { id } });
    if (!report) {
      return res.status(404).json({ message: "Informe no trobat" });
    }
    await report.update({ user_assigned });

    const updatedReport = await Reports.findOne({
      where: { id },
      include: [
```

```
        {
          model: User,
          as: "User",
          attributes: ["id", "name", "email"],
        },
        {
          model: User,
          as: "AssignedUser",
          attributes: ["id", "name", "email"],
        },
        {
          model: Room,
          attributes: ["id", "room_name"],
        },
      ],
    });
    if (!updatedReport) {
      return res.status(404).json({ message: "Informe no trobat" });
    }
    res.json(updatedReport);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté estadístiques dels informes
 * @route GET /report/stats/count
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object} - Estadístiques dels informes
 * @returns {number} Object.total - Nombre total d'informes
 * @returns {number} Object.pending - Nombre d'informes pendents (estat 'pending' o 'revising')
 * @returns {number} Object.resolved - Nombre d'informes resolts (estat 'revised')
 */
router.get("/stats/count", verifyTokenMiddleware, async (req, res) => {
  try {
    const totalReports = await Reports.count();

    const pendingReports = await Reports.count({
      where: {
        status: {
          [Op.in]: ['pending', 'revising']
        }
      }
    });

    const resolvedReports = await Reports.count({
      where: { status: 'revised' }
    });

    res.json({
      total: totalReports,
      pending: pendingReports,
      resolved: resolvedReports
    });
```

```
  } catch (error) {
      console.error("Error en obtenir estadístiques d'incidències:",
error);
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté l'informe més recent amb informació d'usuari i habitació
 * @function getLatestReport
 * @returns {Object|null} - L'informe més recent o null si no n'hi ha
cap
 * @returns {number} Object.id - Identificador de l'informe
 * @returns {string} Object.report - Descripció de l'informe
 * @returns {string|null} Object.image - Ruta a la imatge de l'informe
(si existeix)
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat l'informe
 * @returns {number} Object.room_id - Identificador de l'habitació de
l'informe
 * @returns {number|null} Object.user_assigned - Identificador de
l'usuari assignat (si existeix)
 * @returns {string} Object.status - Estat de l'informe
 * @returns {boolean} Object.finished - Indica si l'informe està
finalitzat
 * @returns {string|null} Object.note - Nota associada a l'informe (si
existeix)
 * @returns {Object} Object.User - Dades de l'usuari creador
 * @returns {string} Object.User.name - Nom de l'usuari creador
 * @returns {Object} Object.Room - Dades de l'habitació
 * @returns {string} Object.Room.room_name - Nom de l'habitació
 * @returns {Date} Object.createdAt - Data de creació de l'informe
 * @returns {Date} Object.updatedAt - Data d'última actualització de
l'informe
 */
export async function getLatestReport() {
  try {
    const latestReport = await Reports.findOne({
      order: [['createdAt', 'DESC']],
      include: [
        {
          model: User,
          as: 'User',
          attributes: ['name']
        },
        {
          model: Room,
          attributes: ['room_name']
        }
      ]
    });

    return latestReport;
  } catch (error) {
    console.error("Error en obtenir incidència recent:", error);
    throw error;
  }
}
```

Response

```
/**
 * Obté totes les respostes
 * @route GET /response/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de totes les respostes
 * @returns {number} Object[].id - Identificador de la resposta
 * @returns {number} Object[].user_id - Identificador de l'usuari que ha creat la resposta
 * @returns {number} Object[].lostAndFound_id - Identificador de l'objecte perdut associat
 * @returns {string} Object[].comment - Contingut de la resposta
 * @returns {Date} Object[].createdAt - Data de creació de la resposta
 * @returns {Date} Object[].updatedAt - Data d'última actualització de la resposta
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    const responses = await Response.findAll();
    res.json(responses);
});

/**
 * Obté una resposta per ID
 * @route GET /response/:id
 * @param {number} id - Identificador de la resposta
 * @returns {Object} - Detalls de la resposta
 * @returns {number} Object.id - Identificador de la resposta
 * @returns {number} Object.user_id - Identificador de l'usuari que ha creat la resposta
 * @returns {number} Object.lostAndFound_id - Identificador de l'objecte perdut associat
 * @returns {string} Object.comment - Contingut de la resposta
 * @returns {Date} Object.createdAt - Data de creació de la resposta
 * @returns {Date} Object.updatedAt - Data d'última actualització de la resposta
 */
router.get("/:id", async (req, res) => {
    const { id } = req.params;
    try {
        const response = await Response.findOne({ where: { id } });
        if (!response) {
                    return res.status(404).json({ message: "Resposta no trobada" });
        }
        res.json(response);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea una nova resposta
 * @route POST /response/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
```

```
 * @param {Object} req.body - Dades de la nova resposta
  * @param {number} req.body.user_id - Identificador de l'usuari que
crea la resposta
   * @param {number} req.body.lostAndFound_id - Identificador de
l'objecte perdut associat
 * @param {string} req.body.comment - Contingut de la resposta
 * @returns {Object} - Resposta creada
 * @returns {number} Object.id - Identificador de la resposta creada
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
creat la resposta
   * @returns {number} Object.lostAndFound_id - Identificador de
l'objecte perdut associat
 * @returns {string} Object.comment - Contingut de la resposta
 * @returns {Date} Object.createdAt - Data de creació de la resposta
 * @returns {Date} Object.updatedAt - Data d'última actualització de la
resposta
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { user_id, lostAndFound_id, comment } = req.body;

        if (!user_id || !lostAndFound_id || !comment) {
                    return res.status(400).json({ message: "user_id,
lostAndFound_id, comment són obligatoris" });
        }
                const newResponse = await Response.create({ user_id,
lostAndFound_id, comment });
        res.status(201).json(newResponse);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza una resposta per ID
 * @route PUT /response/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de la resposta
 * @param {Object} req.body - Dades a actualitzar
 * @param {number} [req.body.user_id] - Nou identificador de l'usuari
  * @param {number} [req.body.lostAndFound_id] - Nou identificador de
l'objecte perdut associat
 * @param {string} [req.body.comment] - Nou contingut de la resposta
 * @returns {Object} - Resposta actualitzada
 * @returns {number} Object.id - Identificador de la resposta
  * @returns {number} Object.user_id - Identificador actualitzat de
l'usuari
 * @returns {number} Object.lostAndFound_id - Identificador actualitzat
de l'objecte perdut associat
  * @returns {string} Object.comment - Contingut actualitzat de la
resposta
 * @returns {Date} Object.createdAt - Data de creació de la resposta
 * @returns {Date} Object.updatedAt - Data d'última actualització de la
resposta
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    const { id } = req.params;
```

```
    try {
        const response = await Response.findOne({ where: { id } });
        if (!response) {
                    return res.status(404).json({ message: "Resposta no
trobada" });
        }
        await response.update(req.body);
        res.json(response);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina una resposta per ID
 * @route DELETE /response/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de la resposta
 * @returns {null} - Resposta buida amb codi d'estat 204 (No Content)
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    const { id } = req.params;
    try {
        const response = await Response.findOne({ where: { id } });
        if (!response) {
                    return res.status(404).json({ message: "Resposta no
trobada" });
        }
        await response.destroy();
        res.status(204).send();
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Room

```
/**
 * Obté totes les aules i processa les seves hores disponibles per dia
 * @route GET /room/
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
   * @returns {Object[]} - Llista de totes les aules amb hores
processades
 * @returns {number} Object[].id - Identificador de l'aula
 * @returns {string} Object[].room_name - Nom de l'aula
 * @returns {Object} Object[].room_hours_available - Objecte amb hores
disponibles per dia
  * @returns {string[]} Object[].room_hours_available_monday - Franges
horàries disponibles el dilluns
 * @returns {string[]} Object[].room_hours_available_tuesday - Franges
horàries disponibles el dimarts
   * @returns {string[]} Object[].room_hours_available_wednesday -
Franges horàries disponibles el dimecres
```

```
 * @returns {string[]} Object[].room_hours_available_thursday - Franges horàries disponibles el dijous
 * @returns {string[]} Object[].room_hours_available_friday - Franges horàries disponibles el divendres
 * @returns {string} Object[].room_description - Descripció de l'aula
 * @returns {boolean} Object[].available - Indica si l'aula està disponible
 * @returns {Date} Object[].createdAt - Data de creació del registre d'aula
 * @returns {Date} Object[].updatedAt - Data d'última actualització del registre d'aula
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const rooms = await Room.findAll();
        const processedRooms = rooms.map((room) => {
            const hoursAvailable = room.room_hours_available || {};
             const days = ["monday", "tuesday", "wednesday", "thursday", "friday"];
            const processedRoom = { ...room.toJSON() };

            days.forEach((day) => {
                if (hoursAvailable[day]) {
                        processedRoom[`room_hours_available_${day}`] = hoursAvailable[day].flatMap((range) => {
                        const [start, end] = range.split("-");
                                                const startTime = new Date(`1970-01-01T${start}:00Z`);
                                                const endTime = new Date(`1970-01-01T${end}:00Z`);
                        const times = [];
                        while (startTime < endTime) {
                                                    const nextTime = new Date(startTime.getTime() + 60 * 60 * 1000);
                            times.push(
                                    `${startTime.toISOString().substr(11, 5)}-${nextTime.toISOString().substr(11, 5)}`
                            );
                            startTime.setTime(nextTime.getTime());
                        }
                        return times;
                    });
                } else {
                        processedRoom[`room_hours_available_${day}`] = null;
                }
            });

            return processedRoom;
        });

        res.json(processedRooms);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
```

```
 * Obté una aula per ID
 * @route GET /room/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'aula
 * @returns {Object} - Detalls de l'aula
 * @returns {number} Object.id - Identificador de l'aula
 * @returns {string} Object.room_name - Nom de l'aula
 * @returns {Object} Object.room_hours_available - Objecte amb hores disponibles per dia
 * @returns {string[]} [Object.room_hours_available.monday] - Franges horàries disponibles el dilluns
 * @returns {string[]} [Object.room_hours_available.tuesday] - Franges horàries disponibles el dimarts
 * @returns {string[]} [Object.room_hours_available.wednesday] - Franges horàries disponibles el dimecres
 * @returns {string[]} [Object.room_hours_available.thursday] - Franges horàries disponibles el dijous
 * @returns {string[]} [Object.room_hours_available.friday] - Franges horàries disponibles el divendres
 * @returns {string} Object.room_description - Descripció de l'aula
 * @returns {boolean} Object.available - Indica si l'aula està disponible
 * @returns {Date} Object.createdAt - Data de creació del registre d'aula
 * @returns {Date} Object.updatedAt - Data d'última actualització del registre d'aula
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const room = await Room.findByPk(req.params.id);
        if (!room) {
            return res.status(404).json({ message: "Aula no trobada" });
        }
        res.json(room);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea una nova aula
 * @route POST /room/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {Object} req.body - Dades de la nova aula
 * @param {string} req.body.room_name - Nom de l'aula
 * @param {Object} req.body.room_hours_available - Objecte amb hores disponibles per dia
 * @param {string[]} [req.body.room_hours_available.monday] - Franges horàries disponibles el dilluns
 * @param {string[]} [req.body.room_hours_available.tuesday] - Franges horàries disponibles el dimarts
 * @param {string[]} [req.body.room_hours_available.wednesday] - Franges horàries disponibles el dimecres
 * @param {string[]} [req.body.room_hours_available.thursday] - Franges horàries disponibles el dijous
```

```
 * @param {string[]} [req.body.room_hours_available.friday] - Franges horàries disponibles el divendres
 * @param {string} req.body.room_description - Descripció de l'aula
 * @param {boolean} [req.body.available] - Indica si l'aula està disponible (true per defecte)
 * @returns {Object} - Aula creada
 * @returns {number} Object.id - Identificador de l'aula creada
 * @returns {string} Object.room_name - Nom de l'aula
 * @returns {Object} Object.room_hours_available - Objecte amb hores disponibles per dia
 * @returns {string} Object.room_description - Descripció de l'aula
 * @returns {boolean} Object.available - Indica si l'aula està disponible
 * @returns {Date} Object.createdAt - Data de creació del registre d'aula
 * @returns {Date} Object.updatedAt - Data d'última actualització del registre d'aula
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { room_name, room_hours_available, room_description, available } = req.body;
        if (!room_name || !room_hours_available || !room_description) {
            return res.status(400).json({ message: "room_name, room_hours_available i room_description són obligatoris" });
        }
        const roomExists = await Room.findOne({ where: { room_name } });
        if (roomExists) {
            return res.status(400).json({ message: "Ja hi ha una aula amb aquest nom" });
        }
        const room = await Room.create({ room_name, room_hours_available, room_description, available: available !== undefined ? available : true });
        res.status(201).json(room);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza una aula per ID
 * @route PUT /room/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} id - Identificador de l'aula
 * @param {Object} req.body - Dades a actualitzar
 * @param {string} [req.body.room_name] - Nou nom de l'aula
 * @param {Object} [req.body.room_hours_available] - Noves hores disponibles per dia
 * @param {string} [req.body.room_description] - Nova descripció de l'aula
 * @param {boolean} [req.body.available] - Nou estat de disponibilitat de l'aula
 * @returns {Object} - Resultat de l'actualització
 * @returns {number} Object[0] - Nombre de files afectades (1 si s'ha actualitzat correctament, 0 si no)
```

```
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
            const { room_name, room_hours_available, room_description,
available } = req.body;

        const room = await Room.update(
                { room_name, room_hours_available, room_description,
available },
            { where: { id: req.params.id } }
        );
        res.json(room);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Elimina una aula per ID
 * @route DELETE /room/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'aula
 * @returns {Object} message - Missatge de confirmació d'eliminació
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    const room = await Room.findByPk(req.params.id);
    if (!room) {
        return res.status(404).json({ message: "Aula no trobada" });
    }
    try {
        await room.destroy();
        res.json({ message: "Aula eliminada correctament" });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté estadístiques de les aules
 * @route GET /room/stats/count
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object} - Estadístiques de les aules
 * @returns {number} Object.total - Nombre total d'aules
 * @returns {number} Object.available - Nombre d'aules disponibles
 * @returns {number} Object.maintenance - Nombre d'aules en manteniment
 */
router.get("/stats/count", verifyTokenMiddleware, async (req, res) => {
    try {
        const totalRooms = await Room.count();

        const availableRooms = await Room.count({
            where: { available: true }
        });

        const maintenanceRooms = await Room.count({
            where: { available: false }
```

```
        });

        res.json({
            total: totalRooms,
            available: availableRooms,
            maintenance: maintenanceRooms
        });
    } catch (error) {
            console.error("Error en obtenir estadístiques d'aules:",
error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * Actualitza l'estat de disponibilitat d'una aula
 * @route PUT /room/:id/availability
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de l'aula
 * @param {Object} req.body - Dades a actualitzar
 * @param {boolean} req.body.available - Nou estat de disponibilitat de
l'aula
 * @returns {Object} - Aula actualitzada
 * @returns {number} Object.id - Identificador de l'aula
 * @returns {string} Object.room_name - Nom de l'aula
 * @returns {Object} Object.room_hours_available - Objecte amb hores
disponibles per dia
 * @returns {string} Object.room_description - Descripció de l'aula
 * @returns {boolean} Object.available - Nou estat de disponibilitat de
l'aula
 * @returns {Date} Object.createdAt - Data de creació del registre
d'aula
 * @returns {Date} Object.updatedAt - Data d'última actualització del
registre d'aula
 */
router.put("/:id/availability", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { id } = req.params;
        const { available } = req.body;

        const room = await Room.findByPk(id);
        if (!room) {
            return res.status(404).json({ message: "Aula no trobada"
});
        }

        await room.update({ available });
        res.json(room);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Room Reservation

```
/**
 * Obté totes les reserves d'aules
 * @route GET /roomReservation/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista de totes les reserves d'aules
 * @returns {number} Object[].id - Identificador de la reserva
 * @returns {number} Object[].user_id - Identificador de l'usuari que ha fet la reserva
 * @returns {number} Object[].room_id - Identificador de l'aula reservada
 * @returns {Date} Object[].start_time - Data i hora d'inici de la reserva
 * @returns {Date} Object[].end_time - Data i hora de finalització de la reserva
 * @returns {Date} Object[].createdAt - Data de creació del registre de reserva
 * @returns {Date} Object[].updatedAt - Data d'última actualització del registre de reserva
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const reservations = await RoomReservation.findAll();
    res.json(reservations);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté les reserves futures d'un usuari específic
 * @route GET /roomReservation/user/:user_id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {number} user_id - Identificador de l'usuari
 * @returns {Object[]} - Llista de reserves futures de l'usuari
 * @returns {number} Object[].id - Identificador de la reserva
 * @returns {number} Object[].user_id - Identificador de l'usuari (igual al paràmetre user_id)
 * @returns {number} Object[].room_id - Identificador de l'aula reservada
 * @returns {Date} Object[].start_time - Data i hora d'inici de la reserva (posterior a la data actual)
 * @returns {Date} Object[].end_time - Data i hora de finalització de la reserva
 * @returns {Date} Object[].createdAt - Data de creació del registre de reserva
 * @returns {Date} Object[].updatedAt - Data d'última actualització del registre de reserva
 */
router.get("/user/:user_id", verifyTokenMiddleware, async (req, res) => {
  const { user_id } = req.params;
  try {
    const reservations = await RoomReservation.findAll({
```

```
      where: {
        user_id,
        start_time: { [Op.gte]: new Date() },
      },
    });
    if (!reservations) {
      return res
        .status(404)
        .json({
              message: "No s'han trobat reserves d'aula per a aquest
usuari",
        });
    }
    res.json(reservations);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les reserves futures d'aules
 * @route GET /roomReservation/reserved
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object[]} - Llista de reserves futures d'aules
 * @returns {number} Object[].id - Identificador de la reserva
  * @returns {number} Object[].user_id - Identificador de l'usuari que
ha fet la reserva
  *  @returns {number} Object[].room_id - Identificador de l'aula
reservada
  * @returns {Date} Object[].start_time - Data i hora d'inici de la
reserva (posterior a la data actual)
 * @returns {Date} Object[].end_time - Data i hora de finalització de
la reserva
 * @returns {Date} Object[].createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object[].updatedAt - Data d'última actualització del
registre de reserva
 */
router.get("/reserved", verifyTokenMiddleware, async (req, res) => {
  try {
    const reservations = await RoomReservation.findAll({
      where: {
        start_time: { [Op.gte]: new Date() },
      },
    });
    if (!reservations) {
       return res.status(404).json({ message: "No hi ha reserves d'aula"
});
    }
    res.json(reservations);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté una reserva d'aula per ID
```

```
 * @route GET /roomReservation/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de la reserva
 * @returns {Object} - Detalls de la reserva
 * @returns {number} Object.id - Identificador de la reserva
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
fet la reserva
 * @returns {number} Object.room_id - Identificador de l'aula reservada
 * @returns {Date} Object.start_time - Data i hora d'inici de la
reserva
 * @returns {Date} Object.end_time - Data i hora de finalització de la
reserva
 * @returns {Date} Object.createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object.updatedAt - Data d'última actualització del
registre de reserva
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
     const reservation = await RoomReservation.findOne({ where: { id }
});
    if (!reservation) {
      return res
        .status(404)
        .json({ message: "Reserva d'aula no trobada" });
    }
    res.json(reservation);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les reserves d'un usuari específic (tant passades com
futures)
 * @route GET /roomReservation/user/:user_id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} user_id - Identificador de l'usuari
 * @returns {Object[]} - Llista de totes les reserves de l'usuari
 * @returns {number} Object[].id - Identificador de la reserva
 * @returns {number} Object[].user_id - Identificador de l'usuari
(igual al paràmetre user_id)
 * @returns {number} Object[].room_id - Identificador de l'aula
reservada
 * @returns {Date} Object[].start_time - Data i hora d'inici de la
reserva
 * @returns {Date} Object[].end_time - Data i hora de finalització de
la reserva
 * @returns {Date} Object[].createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object[].updatedAt - Data d'última actualització del
registre de reserva
 */
router.get("/user/:user_id", verifyTokenMiddleware, async (req, res) =>
{
```

```
  const { user_id } = req.params;
  try {
    const reservations = await RoomReservation.findAll({
      where: { user_id },
    });
    if (!reservations) {
       return res.status(404).json({ message: "No s'han trobat reserves
d'aula per a aquest usuari" });
    }
    res.json(reservations);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Crea una nova reserva d'aula
 * @route POST /roomReservation/
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {Object} req.body - Dades de la nova reserva
 * @param {number} req.body.user_id - Identificador de l'usuari que fa
la reserva
 * @param {number} req.body.room_id - Identificador de l'aula a
reservar
 * @param {Date} req.body.start_time - Data i hora d'inici de la
reserva (ha de ser posterior a la data actual)
 * @param {Date} req.body.end_time - Data i hora de finalització de la
reserva (ha de ser posterior a start_time)
 * @returns {Object} - Reserva creada
 * @returns {number} Object.id - Identificador de la reserva creada
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
fet la reserva
 * @returns {number} Object.room_id - Identificador de l'aula reservada
 * @returns {Date} Object.start_time - Data i hora d'inici de la
reserva
 * @returns {Date} Object.end_time - Data i hora de finalització de la
reserva
 * @returns {Date} Object.createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object.updatedAt - Data d'última actualització del
registre de reserva
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
  try {
    const { user_id, room_id, start_time, end_time } = req.body;

    if (!user_id || !room_id || !start_time || !end_time) {
      return res
        .status(400)
        .json({
                message: "user_id, room_id, start_time i end_time són
obligatoris",
        });
    }
    if (new Date(start_time) >= new Date(end_time)) {
       return res.status(400).json({ message: "La data d'inici ha de ser
anterior a la data de finalització" });
```

```
    }
    if (new Date(start_time) < new Date()) {
        return res.status(400).json({ message: "La data d'inici no pot
ser anterior a la data actual" });
    }
    const existingReservation = await RoomReservation.findOne({
      where: {
        room_id,
        [Op.or]: [
          {
            start_time: {
              [Op.between]: [start_time, end_time],
            },
          },
          {
            end_time: {
              [Op.between]: [start_time, end_time],
            },
          },
        ],
      },
    });
    if (existingReservation) {
      return res
        .status(400)
          .json({ message: "L'aula ja està reservada en aquest horari"
});
    }
    const newReservation = await RoomReservation.create({
      user_id,
      room_id,
      start_time,
      end_time,
    });
    res.status(201).json(newReservation);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Actualitza una reserva d'aula per ID
 * @route PUT /roomReservation/:id
  * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de la reserva
 * @param {Object} req.body - Dades a actualitzar
 * @param {number} [req.body.user_id] - Nou identificador de l'usuari
 * @param {number} [req.body.room_id] - Nou identificador de l'aula
 * @param {Date} [req.body.start_time] - Nova data i hora d'inici de la
reserva
 * @param {Date} [req.body.end_time] - Nova data i hora de finalització
de la reserva
 * @returns {Object} - Reserva actualitzada
 * @returns {number} Object.id - Identificador de la reserva
  * @returns {number} Object.user_id - Identificador actualitzat de
l'usuari
```

```
  * @returns {number} Object.room_id - Identificador actualitzat de
l'aula
   * @returns {Date} Object.start_time - Data i hora d'inici
actualitzades
  * @returns {Date} Object.end_time - Data i hora de finalització
actualitzades
 * @returns {Date} Object.createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object.updatedAt - Data d'última actualització del
registre de reserva
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
     const reservation = await RoomReservation.findOne({ where: { id }
});
    if (!reservation) {
      return res
        .status(404)
        .json({ message: "Reserva d'aula no trobada" });
    }
    await reservation.update(req.body);
    res.json(reservation);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Elimina una reserva d'aula per ID
 * @route DELETE /roomReservation/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {number} id - Identificador de la reserva
 * @returns {null} - Resposta buida amb codi d'estat 204 (No Content)
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
  const { id } = req.params;
  try {
     const reservation = await RoomReservation.findOne({ where: { id }
});
    if (!reservation) {
      return res
        .status(404)
        .json({ message: "Reserva d'aula no trobada" });
    }
    await reservation.destroy();
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté totes les reserves d'una aula específica
 * @route GET /roomReservation/room/:id
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
```

```
 * @param {number} id - Identificador de l'aula
 * @returns {Object[]} - Llista de reserves de l'aula
 * @returns {number} Object[].id - Identificador de la reserva
 * @returns {number} Object[].user_id - Identificador de l'usuari que
ha fet la reserva
 * @returns {number} Object[].room_id - Identificador de l'aula (igual
al paràmetre id)
 * @returns {Date} Object[].start_time - Data i hora d'inici de la
reserva
 * @returns {Date} Object[].end_time - Data i hora de finalització de
la reserva
 * @returns {Date} Object[].createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object[].updatedAt - Data d'última actualització del
registre de reserva
 */
router.get("/room/:id", verifyTokenMiddleware, async (req, res) => {
  try {
    const { id } = req.params;
    const reservations = await RoomReservation.findAll({
      where: {
        room_id: id,
      },
    });
    if (!reservations) {
       return res.status(404).json({ message: "No s'han trobat reserves
d'aula per a aquesta aula" });
    }
    res.json(reservations);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

/**
 * Obté la reserva d'aula més recent amb informació d'aula i usuari
 * @function getLatestRoomReservation
 * @returns {Object|null} - La reserva d'aula més recent o null si no
n'hi ha cap
 * @returns {number} Object.id - Identificador de la reserva
 * @returns {number} Object.user_id - Identificador de l'usuari que ha
fet la reserva
 * @returns {number} Object.room_id - Identificador de l'aula reservada
 * @returns {Date} Object.start_time - Data i hora d'inici de la
reserva
 * @returns {Date} Object.end_time - Data i hora de finalització de la
reserva
 * @returns {Date} Object.createdAt - Data de creació del registre de
reserva
 * @returns {Date} Object.updatedAt - Data d'última actualització del
registre de reserva
 * @returns {Object} Object.Room - Dades de l'aula reservada
 * @returns {string} Object.Room.room_name - Nom de l'aula
 * @returns {Object} Object.User - Dades de l'usuari
 * @returns {string} Object.User.name - Nom de l'usuari
 */
export async function getLatestRoomReservation() {
  try {
```

```
    const latestReservation = await RoomReservation.findOne({
      order: [['createdAt', 'DESC']],
      include: [
        {
          model: Room,
          attributes: ['room_name']
        },
        {
          model: User,
          attributes: ['name']
        }
      ]
    });

    return latestReservation;
  } catch (error) {
    console.error("Error en obtenir reserva recent:", error);
    throw error;
  }
}
```

Task

```
/**
 * @route GET /
 * @description Obtiene todas las tareas del sistema
 * @access Private (Requiere token de autenticación)
 * @returns {Array} Lista de todas las tareas con sus datos
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const Tasks = await Task.findAll();
        res.json(Tasks);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /:id
 * @description Obtiene una tarea específica por su ID
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID de la tarea
 * @returns {Object} Datos de la tarea solicitada
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const Task = await Task.findByPk(req.params.id);
        res.json(Task);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /
```

```
 * @description Crea una nueva tarea en el sistema
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {string} req.body.task_name - Nombre de la tarea
 * @param {number} req.body.course_id - ID del curso al que pertenece
la tarea
 * @param {string} req.body.task_description - Descripción de la tarea
 * @returns {Object} Tarea creada con sus datos
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { task_name, course_id, task_description } = req.body;
        const task_ended = false; // Valor per defecte
            const newTask = await Task.create({ task_name, course_id,
task_description, task_ended });
        res.json(newTask);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /:id
 * @description Actualiza una tarea existente
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID de la tarea a actualizar
 * @param {Object} req.body
 * @param {string} req.body.task_name - Nuevo nombre de la tarea
 * @param {number} req.body.course_id - Nuevo ID del curso
 * @param {string} req.body.task_description - Nueva descripción de la
tarea
 * @param {boolean} req.body.task_ended - Estado de finalización de la
tarea
 * @returns {Object} Tarea actualizada
 */
router.put("/:id",verifyTokenMiddleware, async (req, res) => {
    try {
         const { task_name, course_id, task_description, task_ended } =
req.body;
        const auxTask = await Task.findByPk(req.params.id);
        if (!auxTask) {
            return res.status(404).json({ message: "Task not found" });
        }
        auxTask.task_name = task_name;
        auxTask.course_id = course_id;
        auxTask.task_description = task_description;
        auxTask.task_ended = task_ended;
        await auxTask.save();
        res.json(auxTask);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route DELETE /:id
 * @description Elimina una tarea del sistema
 * @access Private (Requiere token de autenticación)
```

```
 * @param {number} req.params.id - ID de la tarea a eliminar
 * @returns {Object} Mensaje de confirmación de eliminación
 */
router.delete("/:id",verifyTokenMiddleware, async (req, res) => {
    try {
        const Task = await Task.findByPk(req.params.id);
        if (!Task) {
            return res.status(404).json({ message: "Task not found" });
        } else {
            await Task.destroy({ where: { id: req.params.id } });
            res.json({ message: "Task deleted successfully" });
        }
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /getAllGradesFromTask/:id
 * @description Obtiene todas las calificaciones asociadas a una tarea
específica
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID de la tarea
 * @returns {Array} Lista de calificaciones de la tarea
 */
router.get("/getAllGradesFromTask/:id",   verifyTokenMiddleware,   async
(req, res) => {
    try {
        const task = await Task.findByPk(req.params.id, {
            include: [
                {
                    model: Grade,
                    as: "grades",
                },
            ],
        });
        if (!task) {
            return res.status(404).json({ message: "Task not found" });
        }
        res.json(task.grades);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
}
);

/**
 * @route GET /getAllTasksFromCourse/:id
 * @description Obtiene todas las tareas de un curso específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del curso
 * @returns {Array} Lista de tareas del curso
 */
router.get("/getAllTasksFromCourse/:id",   verifyTokenMiddleware,   async
(req, res) => {
    try {
            const  task  =  await  Task.findAll({  where:  {  course_id:
req.params.id } });
```

```
        if (!task) {
            return res.status(404).json({ message: "Task not found" });
        }
        res.json(task);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
}
);

/**
 * @route GET /getAllTasksFromStudent/:id
  * @description Obtiene todas las tareas asignadas a un estudiante
específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del estudiante
 * @returns {Array} Lista de tareas del estudiante
 */
router.get("/getAllTasksFromStudent/:id", verifyTokenMiddleware, async
(req, res) => {
    try {
            const task = await Task.findAll({ where: { user_id:
req.params.id } });
        if (!task) {
            return res.status(404).json({ message: "Task not found" });
        }
        res.json(task);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
}
);

/**
 * @route GET /getAllTasksFromStudentFromCourse/:courseId/:studentId
 * @description Obtiene todas las tareas de un estudiante específico en
un curso específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.courseId - ID del curso
 * @param {number} req.params.studentId - ID del estudiante
  * @returns {Array} Lista de tareas del estudiante en el curso
especificado
 */
router.get("/getAllTasksFromStudentFromCourse/:courseId/:studentId",
verifyTokenMiddleware, async (req, res) => {
    try {
            const task = await Task.findAll({ where: { course_id:
req.params.courseId, user_id: req.params.studentId } });
        if (!task) {
            return res.status(404).json({ message: "Task not found" });
        }
        res.json(task);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
}
);
```

```
export async function getLatestTask() {
    try {
        const latestTask = await Task.findOne({
            order: [['createdAt', 'DESC']],
            include: [
                {
                    model: Course,
                    attributes: ['course_name']
                }
            ]
        });

        return latestTask;
    } catch (error) {
        console.error("Error al obtener tarea reciente:", error);
        throw error;
    }
}
```

Type Users

```
/**
 * @route GET /
 * @description Obtiene todos los tipos de usuario disponibles en el sistema
 * @access Public
 * @returns {Array} Lista de todos los tipos de usuario con sus datos
 */
router.get("/", async (req, res) => {
    try {
        const typeUsers = await TypeUser.findAll();
        res.json(typeUsers);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /:id
 * @description Obtiene un tipo de usuario específico por su ID
 * @access Public
 * @param {number} req.params.id - ID del tipo de usuario
 * @returns {Object} Datos del tipo de usuario solicitado
 */
router.get("/:id", async (req, res) => {
    try {
        const typeUser = await TypeUser.findByPk(req.params.id);
        if (!typeUser) {
            return res.status(404).json({ message: "Tipo de usuario no encontrado" });
        }
        res.json(typeUser);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

```
/**
 * @route POST /
 * @description Crea un nuevo tipo de usuario en el sistema
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {string} req.body.name - Nombre del tipo de usuario
 * @returns {Object} Tipo de usuario creado con sus datos
 */
router.post("/",verifyTokenMiddleware,async (req, res) => {
    try {
        const { name } = req.body;
        // Check if type user already exists
         const existingTypeUser = await TypeUser.findOne({ where: { name
} });
        if (existingTypeUser) {
             return res.status(400).json({ message: "El tipo de usuario
ya existe" });
        }
        // Create new type user
        const typeUser = await TypeUser.create({ name });
        res.json(typeUser);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /:id
 * @description Actualiza un tipo de usuario existente
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del tipo de usuario a actualizar
 * @param {Object} req.body
 * @param {string} req.body.name - Nuevo nombre para el tipo de usuario
 * @returns {Object} Tipo de usuario actualizado
 */
router.put("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { name } = req.body;
        const [updated] = await TypeUser.update({ name }, {
            where: { id: req.params.id },
            returning: true
        });

        if (!updated) {
             return res.status(404).json({ message: "Tipo de usuario no
encontrado" });
        }

        const typeUser = await TypeUser.findByPk(req.params.id);
        res.json(typeUser);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route DELETE /:id
```

```
 * @description Elimina un tipo de usuario del sistema
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del tipo de usuario a eliminar
 * @returns {Object} Mensaje de confirmación de eliminación
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const typeUser = await TypeUser.findByPk(id);

        if (!typeUser) {
             return res.status(404).json({ message: "Tipo de usuario no
encontrado" });
        }

        await typeUser.destroy();
          res.json({ message: "Tipo de usuario eliminado correctamente"
});
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

User

```
/**
 * @route GET /
 * @description Obtiene todos los usuarios registrados en el sistema
 * @access Private (Requiere token de autenticación)
 * @returns {Array} Lista de todos los usuarios con sus datos
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const users = await User.findAll();
        res.json(users);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /
 * @description Crea un nuevo usuario en el sistema
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - ID del tipo de usuario
 * @param {string} req.body.name - Nombre del usuario
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @param {File} req.file - Archivo de imagen de perfil (opcional)
 * @returns {Object} Usuario creado con sus datos
 */
router.post("/", upload.single('profile'), verifyTokenMiddleware, async
(req, res) => {
    console.log("Req body:", req.body);
    console.log("Req file:", req.file);
```

```
    try {
        const { typeUsers_id, name, email, password } = req.body;

        // Verificar si ya existe un usuario con ese correo
        const existingUser = await User.findOne({ where: { email } });

        if (existingUser) {
            return res.status(400).json({
                        message: "Ya existe un usuario con este correo
electrónico"
            });
        }

        // Capturar la ruta del archivo si existe
           const profile = req.file ? `uploads/${req.file.filename}` :
null;

        const user = await User.create({
            typeUsers_id,
            name,
            email,
            password,
            profile  // Ahora guardamos la ruta de la imagen
        });

        res.json(user);
    } catch (error) {
        console.error("Error creating user:", error);
        res.status(500).json({ message: error.message });
    }
});

// Función auxiliar para descargar la imagen desde una URL
async function downloadImage(url, filename) {
    const response = await fetch(url);
    if (!response.ok) {
                throw new Error(`Error al descargar la imagen:
${response.statusText}`);
    }

    // Crear directorio 'uploads' si no existe
    if (!fs.existsSync('uploads')) {
        fs.mkdirSync('uploads');
    }

    const filePath = `uploads/${filename}`;
    const fileStream = fs.createWriteStream(filePath);

    return new Promise((resolve, reject) => {
        response.body.pipe(fileStream);
        response.body.on("error", reject);
        fileStream.on("finish", () => resolve(filePath));
    });
}

/**
 * @route POST /register
```

```
 * @description Registra un nuevo usuario en el sistema
 * @access Public
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - ID del tipo de usuario
 * @param {string} req.body.name - Nombre del usuario
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @param {string} req.body.profile - URL o ruta de la imagen de perfil
(opcional)
 * @param {File} req.file - Archivo de imagen de perfil (opcional)
 * @returns {Object} Usuario registrado con sus datos
 */
router.post("/register", upload.single('profile'), async (req, res) =>
{
    console.log("Req body:", req.body);
    console.log("Req file:", req.file);

    try {
            const { typeUsers_id, name, email, password, profile } =
req.body;

        // Verificar si ya existe un usuario con ese correo
        const existingUser = await User.findOne({ where: { email } });

        if (existingUser) {
            return res.status(400).json({
                        message: "Ya existe un usuario con este correo
electrónico"
            });
        }

        let profilePath = null;

        // Caso 1: Si hay un archivo subido mediante multer
        if (req.file) {
            profilePath = `uploads/${req.file.filename}`;
        }
        // Caso 2: Si hay una URL de imagen en el campo profile
              else if (profile && (profile.startsWith('http://') ||
profile.startsWith('https://'))) {
            try {
                // Generar un nombre único y seguro para el archivo
                  // Evitar usar la extensión de la URL original, ya que
puede ser problemática
                                                    const uniqueFilename =
`${Date.now()}-${Math.round(Math.random() * 1E9)}.jpg`;

                    // Descargar la imagen y obtener la ruta donde se
guardó
                            profilePath = await downloadImage(profile,
uniqueFilename);
                        console.log(`Imagen descargada y guardada en:
${profilePath}`);
            } catch (downloadError) {
                       console.error("Error al descargar la imagen:",
downloadError);
                // Continuar sin imagen si hay error en la descarga
            }
```

```
        }
        // Caso 3: Si ya es una ruta local (empieza con 'uploads/')
        else if (profile && profile.startsWith('uploads/')) {
            profilePath = profile;
        }

         // Crear el usuario con la ruta de la imagen (o null si no hay
imagen)
        const user = await User.create({
            typeUsers_id,
            name,
            email,
            password,
            profile: profilePath
        });

        res.json(user);
    } catch (error) {
        console.error("Error registering user:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /:id
 * @description Obtiene un usuario específico por su ID
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @returns {Object} Datos del usuario solicitado
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /email
 * @description Obtiene un usuario por su correo electrónico
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico del usuario
 * @returns {Object} Usuario con sus datos y tipo de usuario
 */
router.post("/email", verifyTokenMiddleware, async (req, res) => {
    try {
        const { email } = req.body;
        console.log(email);
```

```
        // Obtener el usuario con su tipo de usuario
        const user = await User.findOne({
            where: { email },
            include: [
                {
                    model: TypeUser,
                    as: 'typeusers',
                    attributes: ['id', 'name']
                }
            ]
        });

        console.log(user);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /personalData/:id
 * @description Actualiza los datos personales de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {string} req.body.name - Nuevo nombre del usuario
 * @param {string} req.body.password - Nueva contraseña
 * @param {string} req.body.profile - Nueva imagen de perfil
 * @param {number} req.body.department_id - ID del departamento
 * @param {string} req.body.description - Descripción del usuario
 * @returns {Object} Usuario actualizado
 */
router.put("/personalData/:id", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { id } = req.params;
         const { name, password, profile, department_id, description } =
req.body;

        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

           await user.update({ name, password, profile, department_id,
description });
        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
```

```
    }
});

/**
 * @route PUT /updateRole/:id
 * @description Actualiza el rol de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - Nuevo ID del tipo de usuario
 * @returns {Object} Usuario con rol actualizado
 */
router.put("/updateRole/:id", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { id } = req.params;
        const { typeUsers_id } = req.body;

        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        await user.update({ typeUsers_id });
        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route DELETE /:id
 * @description Elimina un usuario del sistema
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario a eliminar
 * @returns {Object} Mensaje de confirmación de eliminación
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        // Eliminar la imagen si existe
        if (user.profile) {
            try {
                    // La ruta almacenada en la base de datos ya debe
incluir 'uploads/'
                const imagePath = path.resolve(user.profile);

                // Verificar que el archivo existe antes de intentar
eliminarlo
```

```
                if (fs.existsSync(imagePath)) {
                    fs.unlinkSync(imagePath);
                    console.log(`Imagen eliminada: ${imagePath}`);
                }
            } catch (imageError) {
                    console.error("Error al eliminar la imagen:",
imageError);
                // Continuamos con la eliminación del usuario incluso
si hay error con la imagen
            }
        }

        // Eliminar el usuario
        await user.destroy();
        res.json({ message: "Usuario eliminado correctamente" });
    } catch (error) {
        console.error("Error al eliminar usuario:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /login
 * @description Autentica a un usuario en el sistema
 * @access Public
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @returns {Object} Token de acceso y datos del usuario
 */
router.post('/login', async (req, res) => {
    const { email, password } = req.body;

    try {
        const existingUser = await User.findOne({ where: { email } });

        console.log('Existing user:', existingUser);

        if (!existingUser) {
            return res.status(404).json({ error: 'User not found' });
        }

        // Verify password
                    const   match   =   await   bcrypt.compare(password,
existingUser.password);

        if (!match) {
            return res.status(404).json({ error: 'Invalid password' });
        }

        const tokens = generateToken(existingUser);
        return res.status(200).json({
            message: 'Login successful',
            accessToken: tokens,
            userLogin: existingUser
        });
    } catch (error) {
        console.error('Login error:', error);
```

```
        res.status(500).json({ error: 'Server error during login' });
    }
});

/**
 * @route GET /teachers/:department_id
  *  @description  Obtiene  todos  los  profesores  de  un  departamento
específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.department_id - ID del departamento
 * @returns {Array} Lista de profesores del departamento
 */
router.get("/teachers/:department_id",   verifyTokenMiddleware,   async
(req, res) => {
    try {
        const { department_id } = req.params;
        const users = await User.findAll({
            where: {
                typeUsers_id: 1,
                department_id: department_id
            }
        });
        res.json(users);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /stats/count
  * @description Obtiene estadísticas de usuarios (total y registrados
hoy)
 * @access Public
 * @returns {Object} Estadísticas de usuarios
 * @returns {number} total - Total de usuarios registrados
 * @returns {number} registeredToday - Usuarios registrados hoy
 */
router.get("/stats/count", async (req, res) => {
    try {
        // Get total count of users
        const totalUsers = await User.count();

        // Get today's date boundaries (start and end of day)
        const today = new Date();
        const startOfDay = new Date(today.setHours(0, 0, 0, 0));
        const endOfDay = new Date(today.setHours(23, 59, 59, 999));

        // Count users registered today
        const todayUsers = await User.count({
            where: {
                createdAt: {
                    [Op.between]: [startOfDay, endOfDay]
                }
            }
        });

        res.json({
            total: totalUsers,
```

```
            registeredToday: todayUsers
        });
    } catch (error) {
        console.error("Error getting user stats:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /check-email
 * @description Verifica si un correo electrónico ya está registrado
 * @access Public
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico a verificar
 * @returns {Object} Estado de existencia del correo
 * @returns {boolean} exists - Indica si el correo existe
 */
router.post("/check-email", async (req, res) => {
    try {
        const { email } = req.body;

        if (!email) {
                return res.status(400).json({ message: "El correo es
requerido" });
        }

        // Verificar si el correo ya existe
        const existingUser = await User.findOne({ where: { email } });

        return res.json({
            exists: !!existingUser
        });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /updateDepartment/:id
 * @description Actualiza el departamento de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {number} req.body.department_id - Nuevo ID del departamento
 * @returns {Object} Usuario con departamento actualizado
 */
router.put("/updateDepartment/:id", verifyTokenMiddleware, async (req,
res) => {
    try {
        const { id } = req.params;
        const { department_id } = req.body;

        const user = await User.findByPk(id);

        if (!user) {
                return res.status(404).json({ message: "Usuario no
encontrado" });
        }
```

```
        await user.update({ department_id });
        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

// Añadir esta función al final del archivo, antes de export default
router
export async function getLatestUser() {
    try {
        // No necesitamos verificar token aquí
        const latestUser = await User.findOne({
            order: [['createdAt', 'DESC']],
                  attributes: ['id', 'name', 'email', 'typeUsers_id',
'createdAt'],
            include: [
                {
                    model: TypeUser,
                    as: 'typeusers',
                    attributes: ['name']
                }
            ]
        });

        return latestUser;
    } catch (error) {
        console.error("Error al obtener usuario reciente:", error);
        throw error;
    }
}
/**
 * @route GET /
 * @description Obtiene todos los usuarios registrados en el sistema
 * @access Private (Requiere token de autenticación)
 * @returns {Array} Lista de todos los usuarios con sus datos
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const users = await User.findAll();
        res.json(users);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /
 * @description Crea un nuevo usuario en el sistema
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - ID del tipo de usuario
 * @param {string} req.body.name - Nombre del usuario
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @param {File} req.file - Archivo de imagen de perfil (opcional)
 * @returns {Object} Usuario creado con sus datos
```

```
 */
router.post("/", upload.single('profile'), verifyTokenMiddleware, async
(req, res) => {
    console.log("Req body:", req.body);
    console.log("Req file:", req.file);

    try {
        const { typeUsers_id, name, email, password } = req.body;

        // Verificar si ya existe un usuario con ese correo
        const existingUser = await User.findOne({ where: { email } });

        if (existingUser) {
            return res.status(400).json({
                        message: "Ya existe un usuario con este correo
electrónico"
            });
        }

        // Capturar la ruta del archivo si existe
           const profile = req.file ? `uploads/${req.file.filename}` :
null;

        const user = await User.create({
            typeUsers_id,
            name,
            email,
            password,
            profile  // Ahora guardamos la ruta de la imagen
        });

        res.json(user);
    } catch (error) {
        console.error("Error creating user:", error);
        res.status(500).json({ message: error.message });
    }
});

// Función auxiliar para descargar la imagen desde una URL
async function downloadImage(url, filename) {
    const response = await fetch(url);
    if (!response.ok) {
                    throw new Error(`Error al descargar la imagen:
${response.statusText}`);
    }

    // Crear directorio 'uploads' si no existe
    if (!fs.existsSync('uploads')) {
        fs.mkdirSync('uploads');
    }

    const filePath = `uploads/${filename}`;
    const fileStream = fs.createWriteStream(filePath);

    return new Promise((resolve, reject) => {
        response.body.pipe(fileStream);
        response.body.on("error", reject);
        fileStream.on("finish", () => resolve(filePath));
```

```
    });
}

/**
 * @route POST /register
 * @description Registra un nuevo usuario en el sistema
 * @access Public
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - ID del tipo de usuario
 * @param {string} req.body.name - Nombre del usuario
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @param {string} req.body.profile - URL o ruta de la imagen de perfil
(opcional)
 * @param {File} req.file - Archivo de imagen de perfil (opcional)
 * @returns {Object} Usuario registrado con sus datos
 */
router.post("/register", upload.single('profile'), async (req, res) =>
{
    console.log("Req body:", req.body);
    console.log("Req file:", req.file);

    try {
            const { typeUsers_id, name, email, password, profile } =
req.body;

        // Verificar si ya existe un usuario con ese correo
        const existingUser = await User.findOne({ where: { email } });

        if (existingUser) {
            return res.status(400).json({
                        message: "Ya existe un usuario con este correo
electrónico"
            });
        }

        let profilePath = null;

        // Caso 1: Si hay un archivo subido mediante multer
        if (req.file) {
            profilePath = `uploads/${req.file.filename}`;
        }
        // Caso 2: Si hay una URL de imagen en el campo profile
               else if (profile && (profile.startsWith('http://') ||
profile.startsWith('https://'))) {
            try {
                // Generar un nombre único y seguro para el archivo
                  // Evitar usar la extensión de la URL original, ya que
puede ser problemática
                                                const uniqueFilename =
`${Date.now()}-${Math.round(Math.random() * 1E9)}.jpg`;

                    // Descargar la imagen y obtener la ruta donde se
guardó
                            profilePath = await downloadImage(profile,
uniqueFilename);
                        console.log(`Imagen descargada y guardada en:
${profilePath}`);
```

```
            } catch (downloadError) {
                        console.error("Error al descargar la imagen:",
downloadError);
                // Continuar sin imagen si hay error en la descarga
            }
        }
        // Caso 3: Si ya es una ruta local (empieza con 'uploads/')
        else if (profile && profile.startsWith('uploads/')) {
            profilePath = profile;
        }

         // Crear el usuario con la ruta de la imagen (o null si no hay
imagen)
        const user = await User.create({
            typeUsers_id,
            name,
            email,
            password,
            profile: profilePath
        });

        res.json(user);
    } catch (error) {
        console.error("Error registering user:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /:id
 * @description Obtiene un usuario específico por su ID
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @returns {Object} Datos del usuario solicitado
 */
router.get("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /email
 * @description Obtiene un usuario por su correo electrónico
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico del usuario
 * @returns {Object} Usuario con sus datos y tipo de usuario
```

```
 */
router.post("/email", verifyTokenMiddleware, async (req, res) => {
    try {
        const { email } = req.body;
        console.log(email);

        // Obtener el usuario con su tipo de usuario
        const user = await User.findOne({
            where: { email },
            include: [
                {
                    model: TypeUser,
                    as: 'typeusers',
                    attributes: ['id', 'name']
                }
            ]
        });

        console.log(user);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /personalData/:id
 * @description Actualiza los datos personales de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {string} req.body.name - Nuevo nombre del usuario
 * @param {string} req.body.password - Nueva contraseña
 * @param {string} req.body.profile - Nueva imagen de perfil
 * @param {number} req.body.department_id - ID del departamento
 * @param {string} req.body.description - Descripción del usuario
 * @returns {Object} Usuario actualizado
 */
router.put("/personalData/:id", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { id } = req.params;
         const { name, password, profile, department_id, description } =
req.body;

        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }
```

```
            await user.update({ name, password, profile, department_id,
description });
            res.json(user);
        } catch (error) {
            res.status(500).json({ message: error.message });
        }
});

/**
 * @route PUT /updateRole/:id
 * @description Actualiza el rol de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {number} req.body.typeUsers_id - Nuevo ID del tipo de usuario
 * @returns {Object} Usuario con rol actualizado
 */
router.put("/updateRole/:id", verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { id } = req.params;
        const { typeUsers_id } = req.body;

        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        await user.update({ typeUsers_id });
        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route DELETE /:id
 * @description Elimina un usuario del sistema
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario a eliminar
 * @returns {Object} Mensaje de confirmación de eliminación
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        // Eliminar la imagen si existe
        if (user.profile) {
            try {
```

```
                    // La ruta almacenada en la base de datos ya debe
incluir 'uploads/'
                const imagePath = path.resolve(user.profile);

                   // Verificar que el archivo existe antes de intentar
eliminarlo
                if (fs.existsSync(imagePath)) {
                    fs.unlinkSync(imagePath);
                    console.log(`Imagen eliminada: ${imagePath}`);
                }
            } catch (imageError) {
                        console.error("Error al eliminar la imagen:",
imageError);
                  // Continuamos con la eliminación del usuario incluso
si hay error con la imagen
            }
        }

        // Eliminar el usuario
        await user.destroy();
        res.json({ message: "Usuario eliminado correctamente" });
    } catch (error) {
        console.error("Error al eliminar usuario:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /login
 * @description Autentica a un usuario en el sistema
 * @access Public
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico del usuario
 * @param {string} req.body.password - Contraseña del usuario
 * @returns {Object} Token de acceso y datos del usuario
 */
router.post('/login', async (req, res) => {
    const { email, password } = req.body;

    try {
        const existingUser = await User.findOne({ where: { email } });

        console.log('Existing user:', existingUser);

        if (!existingUser) {
            return res.status(404).json({ error: 'User not found' });
        }

        // Verify password
                    const   match   =   await   bcrypt.compare(password,
existingUser.password);

        if (!match) {
            return res.status(404).json({ error: 'Invalid password' });
        }

        const tokens = generateToken(existingUser);
        return res.status(200).json({
```

```
            message: 'Login successful',
            accessToken: tokens,
            userLogin: existingUser
        });
    } catch (error) {
        console.error('Login error:', error);
        res.status(500).json({ error: 'Server error during login' });
    }
});

/**
 * @route GET /teachers/:department_id
  *  @description  Obtiene  todos  los  profesores  de  un  departamento
específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.department_id - ID del departamento
 * @returns {Array} Lista de profesores del departamento
 */
router.get("/teachers/:department_id",   verifyTokenMiddleware,   async
(req, res) => {
    try {
        const { department_id } = req.params;
        const users = await User.findAll({
            where: {
                typeUsers_id: 1,
                department_id: department_id
            }
        });
        res.json(users);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /stats/count
 * @description Obtiene estadísticas de usuarios (total y registrados
hoy)
 * @access Public
 * @returns {Object} Estadísticas de usuarios
 * @returns {number} total - Total de usuarios registrados
 * @returns {number} registeredToday - Usuarios registrados hoy
 */
router.get("/stats/count", async (req, res) => {
    try {
        // Get total count of users
        const totalUsers = await User.count();

        // Get today's date boundaries (start and end of day)
        const today = new Date();
        const startOfDay = new Date(today.setHours(0, 0, 0, 0));
        const endOfDay = new Date(today.setHours(23, 59, 59, 999));

        // Count users registered today
        const todayUsers = await User.count({
            where: {
                createdAt: {
                    [Op.between]: [startOfDay, endOfDay]
```

```
                }
            }
        });

        res.json({
            total: totalUsers,
            registeredToday: todayUsers
        });
    } catch (error) {
        console.error("Error getting user stats:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route POST /check-email
 * @description Verifica si un correo electrónico ya está registrado
 * @access Public
 * @param {Object} req.body
 * @param {string} req.body.email - Correo electrónico a verificar
 * @returns {Object} Estado de existencia del correo
 * @returns {boolean} exists - Indica si el correo existe
 */
router.post("/check-email", async (req, res) => {
    try {
        const { email } = req.body;

        if (!email) {
                return res.status(400).json({ message: "El correo es
requerido" });
        }

        // Verificar si el correo ya existe
        const existingUser = await User.findOne({ where: { email } });

        return res.json({
            exists: !!existingUser
        });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route PUT /updateDepartment/:id
 * @description Actualiza el departamento de un usuario
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID del usuario
 * @param {Object} req.body
 * @param {number} req.body.department_id - Nuevo ID del departamento
 * @returns {Object} Usuario con departamento actualizado
 */
router.put("/updateDepartment/:id", verifyTokenMiddleware, async (req,
res) => {
    try {
        const { id } = req.params;
        const { department_id } = req.body;
```

```
        const user = await User.findByPk(id);

        if (!user) {
                    return res.status(404).json({ message: "Usuario no
encontrado" });
        }

        await user.update({ department_id });
        res.json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

// Añadir esta función al final del archivo, antes de export default
router
export async function getLatestUser() {
    try {
        // No necesitamos verificar token aquí
        const latestUser = await User.findOne({
            order: [['createdAt', 'DESC']],
                   attributes: ['id', 'name', 'email', 'typeUsers_id',
'createdAt'],
            include: [
                {
                    model: TypeUser,
                    as: 'typeusers',
                    attributes: ['name']
                }
            ]
        });

        return latestUser;
    } catch (error) {
        console.error("Error al obtener usuario reciente:", error);
        throw error;
    }
}
```

User Course

```
/**
 * @route GET /
 * @description Obtiene todas las relaciones usuario-curso existentes
 * @access Private (Requiere token de autenticación)
 * @returns {Array} Lista de todas las relaciones usuario-curso
 */
router.get("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const userCourses = await UserCourse.findAll();
        res.json(userCourses);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

```
/**
 * @route POST /
 * @description Crea una nueva relación entre un usuario y un curso
 * @access Private (Requiere token de autenticación)
 * @param {Object} req.body
 * @param {number} req.body.user_id - ID del usuario
 * @param {number} req.body.course_id - ID del curso
 * @returns {Object} La relación usuario-curso creada
 */
router.post("/", verifyTokenMiddleware, async (req, res) => {
    try {
        const { course_id, user_id } = req.body;
        if (!course_id || !user_id) {
             return res.status(400).json({ message: "user_id y course_id
son obligatorios" });
        }
        // Verificar si la relación ya existe
        const existingUserCourse = await UserCourse.findOne({
            where: {
                user_id: user_id,
                course_id: course_id,
            },
        });
        if (existingUserCourse) {
                    return res.status(400).json({ message: "La relación
usuario-curso ya existe" });
        }
        // Crear la nueva relación
        // const { user_id, course_id } = req.body;

        // Validate user existence
        const userExists = await User.findByPk(user_id);
        if (!userExists) {
                 return res.status(400).json({ message: "El usuario no
existe" });
        }

        // Validate course existence
        const courseExists = await Course.findByPk(course_id);
        if (!courseExists) {
             return res.status(400).json({ message: "El curso no existe"
});
        }

         const userCourse = await UserCourse.create({ user_id, course_id
});
        res.json(userCourse);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route DELETE /:id
 * @description Elimina una relación usuario-curso por su ID
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.id - ID de la relación usuario-curso
 * @returns {Object} Mensaje de confirmación de eliminación
```

```
 */
router.delete("/:id", verifyTokenMiddleware, async (req, res) => {
    try {
        const { id } = req.params;
        const userCourse = await UserCourse.findByPk(id);

        if (!userCourse) {
                    return res.status(404).json({ message: "Relación
usuario-curso no encontrada" });
        }

        await userCourse.destroy();
               res.json({ message: "Relación usuario-curso eliminada
correctamente" });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /course/:course_id
  * @description Obtiene todos los usuarios inscritos en un curso
específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.course_id - ID del curso
 * @returns {Array} Lista de usuarios con sus nombres inscritos en el
curso
 */
router.get("/course/:course_id",verifyTokenMiddleware, async (req, res)
=> {
    try {
        const { course_id } = req.params;
             const userCourses = await UserCourse.findAll({ where: {
course_id } });

        if (!userCourses) {
              return res.status(404).json({ message: "No se encontraron
relaciones para este curso" });
        }
                              const    userCoursesWithNames    =    await
Promise.all(userCourses.map(async (userCourse) => {
                const user = await User.findByPk(userCourse.user_id, {
attributes: ['name'] });
                    return { user_id: userCourse.user_id, course_id:
userCourse.course_id, name: user ? user.name : null };
        }));
        // console.log("++++++++++++++++++", userCoursesWithNames);
        res.json(userCoursesWithNames);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /user/:userId
 * @description Obtiene todos los cursos en los que está inscrito un
usuario específico
 * @access Private (Requiere token de autenticación)
```

```
 * @param {number} req.params.userId - ID del usuario
 * @returns {Array} Lista de cursos con sus detalles en los que está
inscrito el usuario
 */
router.get("/user/:userId", verifyTokenMiddleware, async (req, res) =>
{
    try {
        const { userId } = req.params;
        const userCourses = await UserCourse.findAll({
            where: { user_id: userId },
        });
        const userCoursesWithNames = await Promise.all(
            userCourses.map(async (userCourse) => {
                  const course = await Course.findOne({ where: { id:
userCourse.course_id } });
            return {
                ...userCourse.toJSON(),
                   course_name: course ? course.course_name : "Curso no
encontrado",
                course_description: course ? course.course_description
: "Sin descripción disponible"
            };
            })
        );
        res.json(userCoursesWithNames);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/**
 * @route GET /not-enrolled/:userId
 * @description Obtiene todos los cursos en los que NO está inscrito un
usuario específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.userId - ID del usuario
 * @returns {Array} Lista de cursos disponibles para inscripción
 */
router.get("/not-enrolled/:userId", verifyTokenMiddleware, async (req,
res) => {
    try {
        const { userId } = req.params;
        const userCourses = await UserCourse.findAll({
            where: { user_id: userId },
        });
             const enrolledCourseIds = userCourses.map((userCourse) =>
userCourse.course_id);
        const courses = await Course.findAll({
            where: {
                id: {
                    [Op.notIn]: enrolledCourseIds,
                },
            },
        });
        res.json(courses);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
```

```
});

/**
 * @route DELETE /:userId/:courseId
 * @description Elimina la relación entre un usuario específico y un curso específico
 * @access Private (Requiere token de autenticación)
 * @param {number} req.params.userId - ID del usuario
 * @param {number} req.params.courseId - ID del curso
 * @returns {Object} Mensaje de confirmación de eliminación
 */
router.delete("/:userId/:courseId", verifyTokenMiddleware, async (req, res) => {
    try {
        const { userId, courseId } = req.params;
        const userCourse = await UserCourse.findOne({
            where: {
                user_id: userId,
                course_id: courseId,
            },
        });
        if (!userCourse) {
                    return res.status(404).json({ message: "Relación usuario-curso no encontrada" });
        }
        await userCourse.destroy();
                res.json({ message: "Relación usuario-curso eliminada correctamente" });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});
```

Chat

```
/**
 * Comprova i retorna l'estat de la connexió amb MongoDB
 * @function logMongoConnectionState
 * @returns {boolean} - True si la connexió està activa (readyState === 1), False en cas contrari
 */
const logMongoConnectionState = () => {
    const states = {
        0: 'desconnectat',
        1: 'connectat',
        2: 'connectant',
        3: 'desconnectant',
        99: 'no inicialitzat'
    };
                    console.log(`Estat de connexió MongoDB: ${states[mongoose.connection.readyState]}`);
    return mongoose.connection.readyState === 1;
};

/**
 * Obté l'instància de Socket.IO de l'aplicació
 * @function getIo
```

```
 * @param {Object} req - Objecte de la petició Express
 * @returns {Object} - Instància de Socket.IO configurada a l'aplicació
 */
const getIo = (req) => {
    return req.app.get('socketio');
};

/**
 * Obté previsualitzacions per als enllaços detectats en un missatge
 * @function fetchLinkPreviews
 * @param {string[]} links - Array d'URLs a processar
 * @returns {Object[]} - Array de previsualitzacions obtingudes (màxim 3)
 * @returns {string} Object[].url - URL original
 * @returns {string} Object[].title - Títol de la pàgina
 * @returns {string} Object[].description - Descripció de la pàgina
 * @returns {string} Object[].image - URL de la imatge principal
 * @returns {string} Object[].siteName - Nom del lloc web
 */
const fetchLinkPreviews = async (links) => {
    const previews = [];
    const linksToProcess = links.slice(0, 3);

    for (const url of linksToProcess) {
        try {
            console.log(`Obtenint previsualització per: ${url}`);
            const preview = await getLinkPreview(url, {
                timeout: 3000,
                followRedirects: 'follow',
                headers: {
                    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
                    'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
                }
            });

            const formattedPreview = {
                url: url,
                title: preview.title || '',
                description: preview.description || '',
                image: preview.images && preview.images.length > 0 ? preview.images[0] : '',
                siteName: preview.siteName || ''
            };

            previews.push(formattedPreview);
             console.log(`Previsualització obtinguda per ${url}:`, formattedPreview);
        } catch (error) {
             console.error(`Error al obtenir previsualització per ${url}:`, error.message);
            previews.push({
                url: url,
                title: url,
                description: '',
```

```
                image: '',
                siteName: ''
            });
        }
    }

    return previews;
};

/**
 * Obté tots els xats
 * @route GET /api/chat/
 * @returns {Object[]} - Llista de tots els xats
 * @returns {string} Object[]._id - Identificador únic del xat (MongoDB ObjectId)
 * @returns {string} Object[].name - Nom del xat
  * @returns {number[]} Object[].teachers - Array amb els IDs dels professors que participen al xat
 * @returns {Object[]} Object[].interaction - Array d'interaccions al xat
 * @returns {string} Object[].interaction._id - Identificador únic del missatge
 * @returns {number} Object[].interaction.teacherId - ID del professor que va enviar el missatge
  * @returns {string} Object[].interaction.message - Contingut del missatge
  * @returns {boolean} Object[].interaction.hasLinks - Indica si el missatge conté enllaços
  * @returns {string[]} Object[].interaction.links - Array d'URLs detectats al missatge
    * @returns {Object[]} Object[].interaction.linkPreviews - Previsualitzacions dels enllaços
 * @returns {Date} Object[].interaction.date - Data i hora en què es va enviar el missatge
 */
router.get("/", async (req, res) => {
    console.log("GET /api/chat - Sol·licitant tots els xats");
    logMongoConnectionState();

    try {
        const messages = await Message.find();
        console.log(`S'han trobat ${messages.length} xats`);
        res.json(messages);
    } catch (error) {
        console.error("Error al buscar xats:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté tots els xats d'un professor específic
 * @route GET /api/chat/teacher/:teacherId
 * @param {number} teacherId - ID del professor
 * @returns {Object[]} - Llista de xats en què participa el professor
 * @returns {string} Object[]._id - Identificador únic del xat (MongoDB ObjectId)
 * @returns {string} Object[].name - Nom del xat
```

```
 * @returns {number[]} Object[].teachers - Array amb els IDs dels professors (inclou el teacherId)
 * @returns {Object[]} Object[].interaction - Array d'interaccions al xat
 * @returns {string} Object[].interaction._id - Identificador únic del missatge
 * @returns {number} Object[].interaction.teacherId - ID del professor que va enviar el missatge
 * @returns {string} Object[].interaction.message - Contingut del missatge
 * @returns {boolean} Object[].interaction.hasLinks - Indica si el missatge conté enllaços
 * @returns {string[]} Object[].interaction.links - Array d'URLs detectats al missatge
 * @returns {Object[]} Object[].interaction.linkPreviews - Previsualitzacions dels enllaços
 * @returns {Date} Object[].interaction.date - Data i hora en què es va enviar el missatge
 */
router.get("/teacher/:teacherId", async (req, res) => {
    console.log(`GET /api/chat/teacher/${req.params.teacherId} - Sol·licitant xats d'un professor`);
    logMongoConnectionState();

    try {
        const teacherId = parseInt(req.params.teacherId);

        if (isNaN(teacherId)) {
            return res.status(400).json({ message: "ID de professor invàlid" });
        }

        const chats = await Message.find({ teachers: teacherId });

        if (!chats || chats.length === 0) {
            console.log(`No s'han trobat xats per al professor ${teacherId}`);
            return res.status(200).json([]);
        }

        console.log(`S'han trobat ${chats.length} xats per al professor ${teacherId}`);
        res.json(chats);
    } catch (error) {
        console.error(`Error al buscar xats del professor ${req.params.teacherId}:`, error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * Obté un xat específic per ID
 * @route GET /api/chat/:id
 * @param {string} id - Identificador únic del xat (MongoDB ObjectId)
 * @returns {Object} - Detalls del xat
 * @returns {string} Object._id - Identificador únic del xat (MongoDB ObjectId)
 * @returns {string} Object.name - Nom del xat
```

```
 * @returns {number[]} Object.teachers - Array amb els IDs dels professors que participen al xat
 * @returns {Object[]} Object.interaction - Array d'interaccions al xat
 * @returns {string} Object.interaction._id - Identificador únic del missatge
 * @returns {number} Object.interaction.teacherId - ID del professor que va enviar el missatge
 * @returns {string} Object.interaction.message - Contingut del missatge
 * @returns {boolean} Object.interaction.hasLinks - Indica si el missatge conté enllaços
 * @returns {string[]} Object.interaction.links - Array d'URLs detectats al missatge
 * @returns {Object[]} Object.interaction.linkPreviews - Previsualitzacions dels enllaços
 * @returns {Date} Object.interaction.date - Data i hora en què es va enviar el missatge
 */
router.get("/:id", async (req, res) => {
    console.log(`GET /api/chat/${req.params.id} - Sol·licitant un xat específic`);
    logMongoConnectionState();

    try {
        const message = await Message.findById(req.params.id);
        if (!message) {
            console.log(`Xat '${req.params.id}' no trobat`);
            return res.status(404).json({ message: "Xat no trobat" });
        }
        console.log(`Xat '${req.params.id}' trobat`);
        res.json(message);
    } catch (error) {
        console.error(`Error al buscar xat '${req.params.id}':`, error);
        res.status(500).json({ message: error.message });
    }
});

/**
 * Crea un nou xat
 * @route POST /api/chat/
 * @param {Object} req.body - Dades del nou xat
 * @param {string} req.body.name - Nom del xat (obligatori)
 * @param {number[]} [req.body.teachers] - Array amb els IDs dels professors que participen al xat
 * @param {Object[]} [req.body.interaction] - Array inicial d'interaccions al xat (opcional)
 * @returns {Object} - Xat creat
 * @returns {string} Object._id - Identificador únic del xat creat (MongoDB ObjectId)
 * @returns {string} Object.name - Nom del xat
 * @returns {number[]} Object.teachers - Array amb els IDs dels professors que participen al xat
 * @returns {Object[]} Object.interaction - Array d'interaccions al xat (inicialment buit o amb els valors proporcionats)
 */
router.post("/", async (req, res) => {
    console.log("POST /api/chat - Creant nou xat");
```

```
    console.log("Body rebut:", JSON.stringify(req.body, null, 2));
    const isConnected = logMongoConnectionState();

    if (!isConnected) {
            console.error("Error: MongoDB no està connectat. No es pot
crear el xat.");
         return res.status(500).json({ message: "Error de connexió a la
base de dades" });
    }

    if (!req.body.name) {
        console.error("Error: Falta el nom del xat");
            return res.status(400).json({ message: "El nom del xat és
obligatori" });
    }

    const message = new Message({
        name: req.body.name,
        teachers: req.body.teachers || [],
        interaction: req.body.interaction || []
    });

      console.log("Objecte de missatge creat:", JSON.stringify(message,
null, 2));

    try {
        console.log("Intentant desar el missatge...");
        const newMessage = await message.save();
                          console.log("Missatge desat amb èxit:",
JSON.stringify(newMessage, null, 2));
        res.status(201).json(newMessage);
    } catch (error) {
        console.error("Error al desar el missatge:", error);
        res.status(400).json({ message: error.message });
    }
});

/**
 * Afegeix un nou missatge a un xat i notifica mitjançant Socket.IO
 * @route POST /api/chat/:id/message
 * @param {string} id - Identificador únic del xat (MongoDB ObjectId)
 * @param {Object} req.body - Dades del nou missatge
 * @param {number} req.body.teacherId - ID del professor que envia el
missatge
 * @param {string} req.body.message - Contingut del missatge
 * @returns {Object} - Xat actualitzat amb el nou missatge
 * @returns {string} Object._id - Identificador únic del xat
 * @returns {string} Object.name - Nom del xat
  * @returns {number[]} Object.teachers - Array amb els IDs dels
professors que participen al xat
 * @returns {Object[]} Object.interaction - Array d'interaccions al xat
(incloent el nou missatge)
  * @returns {string} Object.interaction._id - Identificador únic del
missatge
  * @returns {number} Object.interaction.teacherId - ID del professor
que va enviar el missatge
   * @returns {string} Object.interaction.message - Contingut del
missatge
```

```
  * @returns {boolean} Object.interaction.hasLinks - Indica si el
missatge conté enllaços
  * @returns {string[]} Object.interaction.links - Array d'URLs
detectats al missatge
    * @returns {Object[]} Object.interaction.linkPreviews -
Previsualitzacions dels enllaços
 * @returns {Date} Object.interaction.date - Data i hora en què es va
enviar el missatge
 * @emits {new_message} - Notificació via Socket.IO amb les dades del
nou missatge
 */
router.post("/:id/message", async (req, res) => {
       console.log(`POST /api/chat/${req.params.id}/message - Afegint
missatge`);
    logMongoConnectionState();

    try {
        const message = await Message.findById(req.params.id);
        if (!message) {
            console.log(`Xat '${req.params.id}' no trobat`);
            return res.status(404).json({ message: "Xat no trobat" });
        }

        const messageText = req.body.message;
        const urlRegex = /(https?:\/\/[^\s]+)/g;
        const links = messageText.match(urlRegex) || [];
        const hasLinks = links.length > 0;

        let linkPreviews = [];
        if (hasLinks) {
            linkPreviews = await fetchLinkPreviews(links);
        }

        const newInteraction = {
            teacherId: req.body.teacherId,
            message: messageText,
            hasLinks: hasLinks,
            links: links,
            linkPreviews: linkPreviews,
            date: new Date()
        };

        message.interaction.push(newInteraction);

        console.log("Desant interacció...");
        const updatedMessage = await message.save();
        console.log("Interacció desada amb èxit");

        const io = getIo(req);
        if (io) {
            const newMessageData = {
                chatId: req.params.id,
                userId: newInteraction.teacherId,
                id: newInteraction._id,
                message: newInteraction.message,
                hasLinks: newInteraction.hasLinks,
                links: newInteraction.links,
                linkPreviews: newInteraction.linkPreviews,
```

```
                timestamp: newInteraction.date
            };

            io.to(req.params.id).emit('new_message', newMessageData);
        }

        res.json(updatedMessage);
    } catch (error) {
        console.error("Error al afegir missatge:", error);
        res.status(400).json({ message: error.message });
    }
});

/**
 * Elimina un xat sencer i notifica mitjançant Socket.IO
 * @route DELETE /api/chat/:id
 * @param {string} id - Identificador únic del xat (MongoDB ObjectId)
 * @returns {Object} message - Missatge de confirmació d'eliminació
 * @emits {chat_deleted} - Notificació global via Socket.IO amb l'ID del xat eliminat
 * @emits {chat_room_closed} - Notificació als usuaris del xat eliminat que la sala ha estat tancada
 */
router.delete("/:id", async (req, res) => {
    console.log(`DELETE /api/chat/${req.params.id} - Eliminant xat`);
    logMongoConnectionState();

    try {
        const message = await Message.findByIdAndDelete(req.params.id);
        if (!message) {
            console.log(`Xat '${req.params.id}' no trobat`);
            return res.status(404).json({ message: "Xat no trobat" });
        }
        console.log("Xat eliminat amb èxit");

        const io = getIo(req);
        if (io) {
            io.emit('chat_deleted', {
                chatId: req.params.id,
                timestamp: new Date()
            });
            io.to(req.params.id).emit('chat_room_closed', {
                chatId: req.params.id,
                message: "Aquest xat ha estat eliminat",
                timestamp: new Date()
            });
        }

        res.json({ message: "Xat eliminat amb èxit" });
    } catch (error) {
        console.error("Error al eliminar xat:", error);
        res.status(500).json({ message: error.message });
    }
});

/**
  * Elimina un missatge específic d'un xat i notifica mitjançant Socket.IO
```

```
 * @route DELETE /api/chat/:id/message/:messageId
 * @param {string} id - Identificador únic del xat (MongoDB ObjectId)
  * @param {string} messageId - Identificador únic del missatge a
eliminar
 * @returns {Object} - Xat actualitzat sense el missatge eliminat
 * @returns {string} Object._id - Identificador únic del xat
 * @returns {string} Object.name - Nom del xat
  * @returns {number[]} Object.teachers - Array amb els IDs dels
professors que participen al xat
 * @returns {Object[]} Object.interaction - Array d'interaccions al xat
(sense el missatge eliminat)
 * @emits {message_deleted} - Notificació via Socket.IO amb les dades
del missatge eliminat
 */
router.delete("/:id/message/:messageId", async (req, res) => {
                                                    console.log(`DELETE
/api/chat/${req.params.id}/message/${req.params.messageId} - Eliminant
missatge`);
    logMongoConnectionState();

    try {
        const message = await Message.findById(req.params.id);
        if (!message) {
            console.log(`Xat '${req.params.id}' no trobat`);
            return res.status(404).json({ message: "Xat no trobat" });
        }

        const messageIndex = message.interaction.findIndex(
                        (interaction) => interaction._id.toString() ===
req.params.messageId
        );

        if (messageIndex === -1) {
              console.log(`Missatge '${req.params.messageId}' no trobat
al xat '${req.params.id}'`);
             return res.status(404).json({ message: "Missatge no trobat"
});
        }

        const deletedMessage = message.interaction[messageIndex];
        message.interaction.splice(messageIndex, 1);

        const updatedMessage = await message.save();
        console.log("Missatge eliminat amb èxit de la base de dades");

        const io = getIo(req);
        if (io) {
            io.to(req.params.id).emit('message_deleted', {
                chatId: req.params.id,
                messageId: req.params.messageId,
                timestamp: new Date(),
                messageContent: deletedMessage.message,
                messageDate: deletedMessage.date,
                teacherId: deletedMessage.teacherId,
                deleted: true
            });
        }
```

```
        res.json(updatedMessage);
    } catch (error) {
        console.error("Error al eliminar missatge:", error);
        res.status(500).json({ message: error.message });
    }
});
```

Index (Pincipal)

```
/**
 * Obté l'estat de tots els serveis registrats
 * @route GET /api/services
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista amb l'estat de tots els serveis
 * @returns {string} Object[].name - Nom del servei
 * @returns {string} Object[].description - Descripció del servei
 * @returns {string} Object[].script - Nom del fitxer script del servei
 * @returns {string} Object[].state - Estat del servei ("running", "stopped", "crashed", etc.)
 * @returns {string} Object[].tech - Tecnologia utilitzada pel servei
 * @returns {number|null} Object[].port - Port on s'executa el servei (si aplica)
 * @returns {Date|null} Object[].lastStart - Data i hora de l'última vegada que es va iniciar
 * @returns {Date|null} Object[].lastStop - Data i hora de l'última vegada que es va aturar
 * @returns {string|null} Object[].error - Missatge d'error (si hi ha algun problema)
 */
app.get("/api/services", verifyTokenMiddleware, (req, res) => {
    res.json(getAllServicesStatus());
});

/**
 * Obté l'estat d'un servei específic
 * @route GET /api/services/:serviceName
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {string} serviceName - Nom del servei
 * @returns {Object} - Estat del servei sol·licitat
 * @returns {string} Object.name - Nom del servei
 * @returns {string} Object.description - Descripció del servei
 * @returns {string} Object.script - Nom del fitxer script del servei
 * @returns {string} Object.state - Estat del servei ("running", "stopped", "crashed", etc.)
 * @returns {string} Object.tech - Tecnologia utilitzada pel servei
 * @returns {number|null} Object.port - Port on s'executa el servei (si aplica)
 * @returns {Date|null} Object.lastStart - Data i hora de l'última vegada que es va iniciar
 * @returns {Date|null} Object.lastStop - Data i hora de l'última vegada que es va aturar
 * @returns {string|null} Object.error - Missatge d'error (si hi ha algun problema)
```

```
 */
app.get("/api/services/:serviceName", verifyTokenMiddleware, (req, res)
=> {
    const status = getServiceStatus(req.params.serviceName);
    if (status) {
        res.json(status);
    } else {
        res.status(404).json({ error: "Servei no trobat" });
    }
});

/**
 * Inicia tots els serveis registrats
 * @route POST /api/services/start-all
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object} - Resultat de l'operació d'inici de tots els
serveis
 * @returns {string} Object.message - Missatge indicant que s'estan
iniciant els serveis
 * @returns {Object[]} Object.results - Resultats de l'operació per
cada servei
 * @returns {string} Object.results[].name - Nom del servei
 * @returns {boolean} Object.results[].success - Indica si l'operació
va tenir èxit
 * @returns {string} Object.results[].message - Missatge de resultat
per aquest servei
 */
app.post("/api/services/start-all", verifyTokenMiddleware, (req, res)
=> {
    const results = startAllServices();
    res.json({ message: "Iniciant tots els serveis", results });
});

/**
 * Atura tots els serveis registrats
 * @route POST /api/services/stop-all
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @returns {Object} - Resultat de l'operació d'aturada de tots els
serveis
 * @returns {string} Object.message - Missatge indicant que s'estan
aturant els serveis
 * @returns {Object[]} Object.results - Resultats de l'operació per
cada servei
 * @returns {string} Object.results[].name - Nom del servei
 * @returns {boolean} Object.results[].success - Indica si l'operació
va tenir èxit
 * @returns {string} Object.results[].message - Missatge de resultat
per aquest servei
 */
app.post("/api/services/stop-all", verifyTokenMiddleware, (req, res) =>
{
    const results = stopAllServices();
    res.json({ message: "Aturant tots els serveis", results });
});

/**
```

```
 * Inicia un servei específic
 * @route POST /api/services/:serviceName/start
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {string} serviceName - Nom del servei a iniciar
 * @returns {Object} - Resultat de l'operació d'inici
 * @returns {boolean} Object.success - Indica si l'operació va tenir èxit
 * @returns {string} Object.message - Missatge descriptiu del resultat
 * @returns {Object} [Object.service] - Informació actualitzada del servei (si té èxit)
 * @returns {string} [Object.service.state] - Nou estat del servei després de la operació
 * @returns {Date} [Object.service.lastStart] - Data i hora d'inici del servei
 */
app.post("/api/services/:serviceName/start", verifyTokenMiddleware, (req, res) => {
    const result = startService(req.params.serviceName);
    if (result.success) {
        res.json(result);
    } else {
        res.status(400).json(result);
    }
});

/**
 * Atura un servei específic
 * @route POST /api/services/:serviceName/stop
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @param {string} serviceName - Nom del servei a aturar
 * @returns {Object} - Resultat de l'operació d'aturada
 * @returns {boolean} Object.success - Indica si l'operació va tenir èxit
 * @returns {string} Object.message - Missatge descriptiu del resultat
 * @returns {Object} [Object.service] - Informació actualitzada del servei (si té èxit)
 * @returns {string} [Object.service.state] - Nou estat del servei després de la operació
 * @returns {Date} [Object.service.lastStop] - Data i hora d'aturada del servei
 */
app.post("/api/services/:serviceName/stop", verifyTokenMiddleware, (req, res) => {
    const result = stopService(req.params.serviceName);
    if (result.success) {
        res.json(result);
    } else {
        res.status(400).json(result);
    }
});

/**
 * Registra un nou servei
 * @route POST /api/services
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
```

```
 * @param {Object} req.body - Dades del nou servei
 * @param {string} req.body.name - Nom del servei (obligatori)
 * @param {string} req.body.script - Nom del fitxer script (obligatori)
 * @param {string} [req.body.description] - Descripció del servei
 * @param {string} [req.body.tech] - Tecnologia utilitzada (per defecte
'Node.js 18.x')
 * @param {number} [req.body.port] - Port on s'executarà el servei
 * @param {boolean} [req.body.autoStart] - Indica si el servei s'ha
d'iniciar automàticament després de crear-lo
 * @returns {Object} - Resultat de l'operació de registre
 * @returns {boolean} Object.success - Indica si l'operació va tenir
èxit
 * @returns {string} Object.message - Missatge descriptiu del resultat
 * @returns {Object} [Object.service] - Informació del servei registrat
(si té èxit)
 */
app.post("/api/services", verifyTokenMiddleware, (req, res) => {
      const { name, script, description, tech, port, autoStart } =
req.body;

    if (!name || !script) {
         return res.status(400).json({ success: false, message: "El nom
i l'script són obligatoris" });
    }

    try {
        const result = addService(name, script, {
            name: description || `Servei ${name}`,
            tech: tech || 'Node.js 18.x',
            port: port || null
        });

        if (result.success && autoStart) {
            startService(name);
        }

        res.status(201).json(result);
    } catch (error) {
          res.status(500).json({ success: false, message: error.message
});
    }
});

/**
 * Elimina un servei
 * @route DELETE /api/services/:serviceName
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui
autenticat
 * @param {string} serviceName - Nom del servei a eliminar
 * @returns {Object} - Resultat de l'operació d'eliminació
 * @returns {boolean} Object.success - Indica si l'operació va tenir
èxit
 * @returns {string} Object.message - Missatge descriptiu del resultat
 */
app.delete("/api/services/:serviceName", verifyTokenMiddleware, async
(req, res) => {
    const serviceName = req.params.serviceName;
    try {
```

```
        const service = getServiceStatus(serviceName);
        if (service && service.state === 'running') {
            stopService(serviceName);
        }

        const { deleteService } = await import("./serviceManager.js");
        const result = deleteService(serviceName);

        if (result.success) {
            res.json(result);
        } else {
            res.status(400).json(result);
        }
    } catch (error) {
        res.status(500).json({
            success: false,
            message: `Error en eliminar el servei: ${error.message}`
        });
    }
});

/**
 * Carrega un fitxer JavaScript com a nou servei
 * @route POST /api/services/upload
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @middleware upload.single('file') - Processa la càrrega del fitxer JavaScript
 * @param {Object} req.body - Dades del nou servei
 * @param {string} req.body.name - Nom del servei (obligatori)
 * @param {string} [req.body.description] - Descripció del servei
 * @param {string} [req.body.tech] - Tecnologia utilitzada (per defecte 'Node.js 18.x')
 * @param {string} [req.body.autoStart] - Indica si el servei s'ha d'iniciar automàticament després de crear-lo ('true' o 'false')
 * @param {File} req.file - Fitxer JavaScript a carregar com a servei
 * @returns {Object} - Resultat de l'operació de càrrega i registre
 * @returns {boolean} Object.success - Indica si l'operació va tenir èxit
 * @returns {string} Object.message - Missatge descriptiu del resultat
 * @returns {Object} [Object.service] - Informació del servei registrat (si té èxit)
 */
app.post("/api/services/upload",                    verifyTokenMiddleware,
upload.single('file'), async (req, res) => {
    try {
        const { name, description, tech, autoStart } = req.body;

        if (!name || !req.file) {
            return res.status(400).json({
                success: false,
                message: "El nom i l'arxiu són obligatoris"
            });
        }

        const filename = req.file.filename;
        const result = addService(name, filename, {
            name: description || `Servei ${name}`,
```

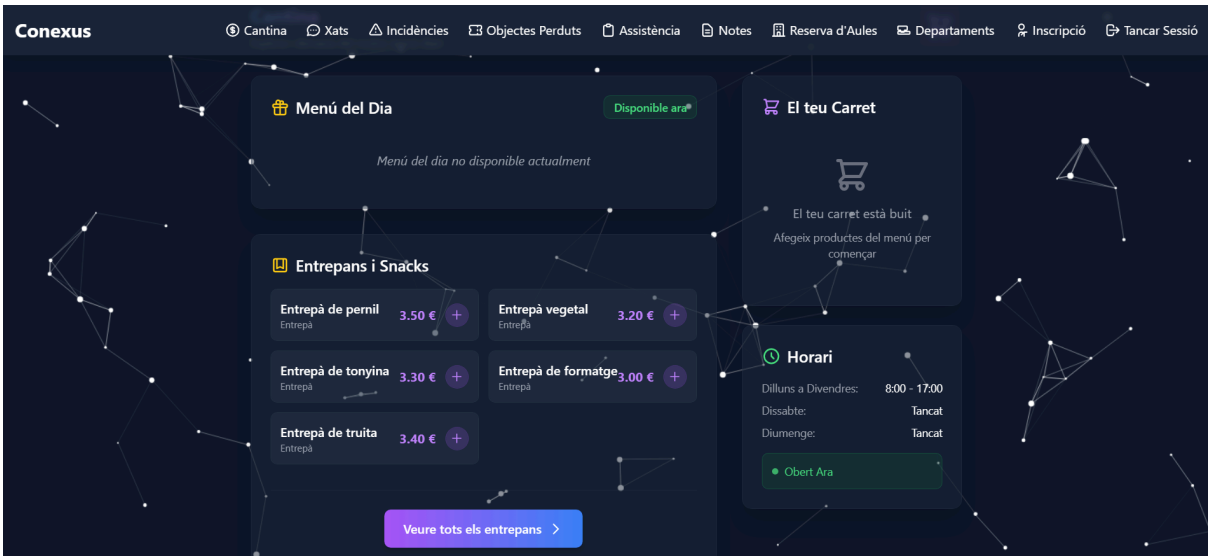
```
            tech: tech || 'Node.js 18.x'
        });

        if (result.success && autoStart === 'true') {
            startService(name);
        }

        res.status(201).json(result);
    } catch (error) {
        console.error("Error en la pujada de l'arxiu:", error);
        res.status(500).json({
            success: false,
            message: `Error en crear el servei: ${error.message}`
        });
    }
});

/**
 * Obté les últimes activitats de diverses entitats del sistema
 * @route GET /api/activities
 * @middleware verifyTokenMiddleware - Verifica que l'usuari estigui autenticat
 * @returns {Object[]} - Llista d'activitats recents ordenades per data (més recents primer)
 * @returns {string} Object[].type - Tipus d'activitat ("user", "course", "report", "lostObject", "roomReservation", "task")
 * @returns {Object} Object[].data - Dades de l'entitat relacionada amb l'activitat
 * @returns {number|string} Object[].data.id - Identificador de l'entitat
 * @returns {string|null} [Object[].data.name] - Nom de l'entitat (per a usuaris i cursos)
 * @returns {string|null} [Object[].data.room_name] - Nom de l'aula (per a reserves d'aula)
 * @returns {string|null} [Object[].data.title] - Títol (per a objectes perduts)
 * @returns {string|null} [Object[].data.report] - Descripció de l'informe (per a informes)
 * @returns {string|null} [Object[].data.task_name] - Nom de la tasca (per a tasques)
 * @returns {Date} Object[].data.createdAt - Data de creació de l'entitat (utilitzada per ordenar)
 * @returns {Object|null} [Object[].data.User] - Informació addicional de l'usuari relacionat (si aplica)
 * @returns {Object|null} [Object[].data.Room] - Informació addicional de l'aula relacionada (si aplica)
 */
app.get("/api/activities", verifyTokenMiddleware, async (req, res) => {
    try {
        const activities = [];

        try {
            const latestUser = await getLatestUser();
            if (latestUser) {
                activities.push({
                    type: "user",
                    data: latestUser
                });
```

```
            }
        } catch (error) {
                    console.error("Error en obtenir usuari recent:",
error.message);
        }

        try {
            const latestCourse = await getLatestCourse();
            if (latestCourse) {
                activities.push({
                    type: "course",
                    data: latestCourse
                });
            }
        } catch (error) {
                     console.error("Error en obtenir curs recent:",
error.message);
        }

        try {
            const latestReport = await getLatestReport();
            if (latestReport) {
                activities.push({
                    type: "report",
                    data: latestReport
                });
            }
        } catch (error) {
                  console.error("Error en obtenir incidència recent:",
error.message);
        }

        try {
            const latestLostObject = await getLatestLostObject();
            if (latestLostObject) {
                activities.push({
                    type: "lostObject",
                    data: latestLostObject
                });
            }
        } catch (error) {
                console.error("Error en obtenir objecte perdut recent:",
error.message);
        }

        try {
                                const latestRoomReservation = await
getLatestRoomReservation();
            if (latestRoomReservation) {
                activities.push({
                    type: "roomReservation",
                    data: latestRoomReservation
                });
            }
        } catch (error) {
                    console.error("Error en obtenir reserva recent:",
error.message);
        }
```

```
        try {
            const latestTask = await getLatestTask();
            if (latestTask) {
                activities.push({
                    type: "task",
                    data: latestTask
                });
            }
        } catch (error) {
                    console.error("Error en obtenir tasca recent:",
error.message);
        }

        activities.sort((a, b) => {
            const dateA = new Date(a.data.createdAt || 0);
            const dateB = new Date(b.data.createdAt || 0);
            return dateB - dateA;
        });

        res.json(activities);
    } catch (error) {
        console.error("Error obtenint últimes activitats:", error);
        res.status(500).json({
            message: "Error obtenint últimes activitats",
            error: error.message
        });
    }
});
```

3. Sockets

```
io.on("connection", (socket) => {

    socket.on("register_user", (userData) => {
        const { userId, userName } = userData;
        console.log(`Usuari ${userName} (${userId}) s'ha registrat`);
        connectedUsers.set(userId, socket.id);
        userSockets.set(socket.id, userId);
        socket.join(`user:${userId}`);
        console.log(`Usuari ${userName} (${userId}) unit a sala personal user:${userId}`);
        socket.emit("registered", {
            success: true,
            userId,
            message: "Connectat correctament"
        });
    });

    socket.on("join_chat", (data) => {
        const { chatId, userId, userName } = data;
        console.log(`Usuari ${userName} (${userId}) s'uneix al xat ${chatId}`);
        socket.join(chatId);
        if (!activeRooms.has(chatId)) {
            activeRooms.set(chatId, new Set());
        }
        activeRooms.get(chatId).add(socket.id);
        socket.to(chatId).emit("user_joined", {
            userId,
            userName,
            chatId,
            timestamp: new Date()
        });
    });

    socket.on("send_message", async (data) => {
        const { chatId, message, userId, userName } = data;
        console.log(`Missatge de ${userName} al xat ${chatId}: ${message}`);
```



Tenim un servei “Chat” que és on s’utilitzen els sockets. Els sockets s’encarreguen de crear les rooms, per cada conversa individual. Ajuda al temps real de l’aplicació, ja que reps i envies missatges en temps real. A més està la funció d'eliminar missatge.

Com a part destacable hi existeix un xat amb cantina, on es fa les comandes a partir d'un menú visual i un tiquet virtual. Les teves comandes s'afegeixen al cistell i a l'hora d'enviar-ho ho fa utilitzant un xat privat teu amb cantina.

Principals Funcionalitats dels Sockets:

- **Registre d'Usuaris:** Els professors es registren al sistema de xat i se'ls assigna un socket per mantenir una connexió persistent.
- **Gestió de Sales de Xat:** Els usuaris poden unir-se a diferents sales de xat i rebre notificacions quan altres usuaris entren o surten.
- **Missatgeria Instantània:** Permet enviar i rebre missatges en temps real, amb notificacions quan algú està escrivint.
- **Notificacions de Primer Missatge:** Sistema especial d'alertes quan s'inicia una nova conversa, fins i tot per a professors que no estan connectats a la sala.
- **Gestió de Contingut:** Els usuaris poden eliminar missatges individuals o converses senceres, amb notificacions immediates a tots els participants.
- **Control de Presència:** Seguiment de quins usuaris estan connectats i en quines sales es troben en tot moment.
- **Emmagatzematge Persistent:** Tots els missatges es desen a MongoDB per mantenir l'historial de converses.
- **Gestió de Desconnexions:** Administra correctament quan els usuaris es desconnecten, notificant-ho als altres participants.

4. Base de dades

Sequelize → MySQL

Sequelize ofereix nombrosos beneficis i avantatges per a la gestió de bases de dades en aplicacions Node.js. Simplifica la interacció amb bases de dades relacionals mitjançant models Javascript, cosa que permet definir estructures de dades i relacions de manera intuïtiva. Facilita la realització d'operacions CRUD (Crear, Llegir, Actualitzar, Eliminar) amb una sintaxi clara i concisa, reduint la quantitat de codi necessari en comparació amb l'ús directe de consultes SQL.

Un dels avantatges clau de Sequelize és la seva compatibilitat amb múltiples sistemes de gestió de bases de dades (com PostgreSQL, MySQL, SQLite i MariaDB), oferint flexibilitat per triar la base de dades més adequada per a cada projecte sense haver de modificar significativament la lògica d'accés a les dades. A més, proporciona un potent sistema de migracions que facilita l'evolució de l'esquema de la base de dades al llarg del temps, cosa que permet aplicar canvis de manera controlada i consistent entre diferents entorns.

Sequelize també incorpora funcionalitats avançades com associacions (un a un, un a molts, molts a molts), validacions de dades en l'àmbit de model, transaccions per garantir la integritat de les operacions, i la possibilitat de definir hooks (ganxos) per executar lògica personalitzada en diferents moments del cicle de vida dels models (abans o després de la creació, actualització, eliminació, etc.). Així mateix, ofereix eines per a la realització de consultes complexes amb filtres, ordenació, paginació i càrrega eager (carregant dades relacionades en una sola consulta), optimitzant el rendiment de l'aplicació. L'ús de Sequelize contribueix a un desenvolupament més ràpid, mantenible i segur d'aplicacions que interactuen amb bases de dades relacionals.

Els nostres models de Sequelize són els següents:

Assistence

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Assistence = sequelize.define('Assistence', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    user_id: { type: DataTypes.INTEGER, allowNull: false },
    course_id: { type: DataTypes.INTEGER, allowNull: false },
    hour: { type: DataTypes.STRING, allowNull: false },
    day: { type: DataTypes.DATE, allowNull: false },
    assisted: { type: DataTypes.ENUM('yes', 'unjustified', 'justified', 'not selected', 'late'), defaultValue: 'unjustified' },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Assistence',
    timestamps: true,
});

export default Assistence;
```

Segueix l'assistència dels estudiants als cursos amb camps per a data, hora i estat d'assistència (sí, injustificat, justificat, etc.).

Canteen Item

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const CanteenItem = sequelize.define('CanteenItem', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    product_name: { type: DataTypes.TEXT, allowNull: false, },
    product_price: { type: DataTypes.DECIMAL(10, 2), allowNull: false, },
    product_enabled: { type: DataTypes.BOOLEAN, allowNull: false, defaultValue: true },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'CanteenItems',
    timestamps: true,
});
export default CanteenItem;
```

Representa productes disponibles a la cantina amb camps per a nom, preu i estat de disponibilitat.

Course

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Course = sequelize.define('Course', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    course_name: { type: DataTypes.TEXT, allowNull: false, },
    course_hours_available: { type: DataTypes.JSON, allowNull: true, defaultValue: null, },
    course_description: { type: DataTypes.TEXT, allowNull: false, },
    course_teacher_id: {
        type: DataTypes.INTEGER,
        allowNull: true,
    },
    course_department_id: { type: DataTypes.INTEGER, allowNull: false },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Courses',
    timestamps: true,
});
export default Course;
```

Representa cursos acadèmics amb camps per a nom, hores disponibles, descripció i associacions amb professors, departaments, tasques i inscripcions d'estudiants.

Department

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Department = sequelize.define('Department', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    name: { type: DataTypes.STRING(255), allowNull: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Departments',
    timestamps: true,
});

export default Department;
```

Representa departaments acadèmics amb un camp de nom. Té relacions amb Users i Courses.

Grade

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Grade = sequelize.define('Grade', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    user_id: { type: DataTypes.INTEGER, allowNull: false },
    task_id: { type: DataTypes.INTEGER, allowNull: false },
    grade: { type: DataTypes.FLOAT, allowNull: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Grades',
    timestamps: true,
});

export default Grade;
```

Emmagatzema les qualificacions dels estudiants per a tasques específiques, connectant usuaris i tasques amb un valor numèric de qualificació.

Lost Objects

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const LostObject = sequelize.define('LostObject', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    title: { type: DataTypes.STRING(255), allowNull: false, },
    description: { type: DataTypes.TEXT, allowNull: false, },
    image: { type: DataTypes.STRING(255), allowNull: true, },
    user_id: { type: DataTypes.INTEGER, allowNull: false },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'LostObjects',
    timestamps: true,
});

export default LostObject;
```

Gestiona els objectes perduts amb títol, descripció, imatge i associació amb l'usuari que ho va reportar.

Reports

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';
import Room from './Room.js';
import User from './User.js';


const Report = sequelize.define('Report', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    user_id: { type: DataTypes.INTEGER, allowNull: true },
    report: { type: DataTypes.TEXT, allowNull: false },
    status: { type: DataTypes.ENUM('pending', 'revising', 'revised'), defaultValue: 'pending' },
    image: { type: DataTypes.TEXT, allowNull: true },
    room_id: { type: DataTypes.INTEGER, allowNull: false },
    user_assigned: { type: DataTypes.INTEGER, allowNull: true },
    note: { type: DataTypes.TEXT, allowNull: true },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Reports',
    timestamps: true,
});

Report.belongsTo(Room, { foreignKey: 'room_id', onDelete: 'CASCADE' });
Report.belongsTo(User, { foreignKey: 'user_id', as: 'User', onDelete: 'SET NULL' });
Report.belongsTo(User, { foreignKey: 'user_assigned', as: 'AssignedUser', onDelete: 'SET NULL' });

export default Report;
```

Permet als usuaris informar de problemes relacionats amb les aules, incloent seguiment d'estat, adjunts d'imatges i assignació al personal.

Response

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Response = sequelize.define('Response', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    user_id: { type: DataTypes.INTEGER, allowNull: false },
    lostAndFound_id: { type: DataTypes.INTEGER, allowNull: false },
    comment: { type: DataTypes.TEXT, allowNull: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Responses',
    timestamps: true,
});

export default Response;
```

Emmagatzema comentaris/respostes a publicacions d'objectes perduts, connectant usuaris amb llistats específics d'objectes perduts.

Room

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Room = sequelize.define('Room', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    room_name: { type: DataTypes.TEXT, allowNull: false },
    room_hours_available: { type: DataTypes.JSON, allowNull: true, defaultValue: null },
    room_description: { type: DataTypes.TEXT, allowNull: false },
    available: { type: DataTypes.BOOLEAN, allowNull: false, defaultValue: true },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Rooms',
    timestamps: true,
});

export default Room;
```

Representa espais físics amb camps per a nom, descripció, hores disponibles i estat de disponibilitat.

Room Reservation

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const RoomReservation = sequelize.define('RoomReservation', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    user_id: { type: DataTypes.INTEGER, allowNull: false },
    room_id: { type: DataTypes.INTEGER, allowNull: false },
    start_time: { type: DataTypes.DATE, allowNull: false, },
    end_time: { type: DataTypes.DATE, allowNull: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'RoomReservations',
    timestamps: true,
});

export default RoomReservation;
```

Gestiona les reserves d'aules amb hores d'inici i finalització, connectant usuaris amb aules per a franges horàries específiques

Task

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const Task = sequelize.define('Task', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    course_id: { type: DataTypes.INTEGER, allowNull: false },
    task_name: { type: DataTypes.TEXT, allowNull: false, },
    task_description: { type: DataTypes.TEXT, allowNull: false, },
    task_ended: { type: DataTypes.BOOLEAN, defaultValue: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Tasks',
    timestamps: true,
});

export default Task;
```

Representa tasques o activitats dins d'un curs amb camps per a nom, descripció i estat de finalització.

Type Users

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const TypeUser = sequelize.define('TypeUser', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    name: { type: DataTypes.STRING(255), allowNull: false, },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'TypeUsers',
    timestamps: true,
});

export default TypeUser;
```

Defineix els tipus/rols d'usuaris al sistema (com a estudiant, professor, administrador). Conté un camp nom i està relacionat amb el model User.

User

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';
import { hashPassword } from '../routes/userRoutes.js';

const User = sequelize.define('User', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    typeUsers_id: { type: DataTypes.INTEGER, allowNull: false, defaultValue: 1 },
    name: { type: DataTypes.STRING(255), allowNull: false },
    email: { type: DataTypes.STRING(255), allowNull: false, unique: true },
    password: { type: DataTypes.STRING(255), allowNull: false },
    profile: { type: DataTypes.STRING(255), allowNull: true },
    department_id: { type: DataTypes.INTEGER, allowNull: true },
    description: { type: DataTypes.TEXT, allowNull: true, defaultValue: null },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'Users',
    timestamps: true,
});

User.beforeCreate(async (user) => {
    user.password = await hashPassword(user.password);
});

export default User;
```

Model que representa els usuaris del sistema amb relacions amb TypeUser (per a rols) i Department. Els usuaris poden tenir cursos, qualificacions, informes, registres d'assistència i altres entitats associades.

User Course

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/database.js';

const UserCourse = sequelize.define('UserCourse', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, },
    user_id: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'Users',
            key: 'id',
        },
        onDelete: 'CASCADE',
    },
    course_id: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'Courses',
            key: 'id',
        },
        onDelete: 'CASCADE',
    },
    createdAt: { field: 'created_at', type: DataTypes.DATE },
    updatedAt: { field: 'updated_at', type: DataTypes.DATE },
}, {
    tableName: 'UserCourses',
    timestamps: true,
});

export default UserCourse;
```

Una taula d'unió que connecta Users amb Courses, establint relacions d'inscripció entre estudiants i cursos.

Mongoose → MongoDB

Mongoose proporciona una solució elegant i potent per interactuar amb bases de dades MongoDB en entorns Node.js. Un dels seus principals avantatges rau en la definició d'esquemes de dades. Aquests esquemes permeten modelar les dades de manera clara i concisa, definint els tipus de camps, les validacions i els comportaments per defecte. Aquesta estructuració facilita la comprensió i el manteniment del codi, a més de proporcionar una capa de validació que assegura la integritat de les dades abans de ser emmagatzemades a la base de dades.

Un altre benefici clau de Mongoose és el seu Object Data Modeling (ODM). A través dels models definits a partir dels esquemes, Mongoose permet interactuar amb les col·leccions de MongoDB d'una manera orientada a objectes. Això significa que es poden realitzar operacions de creació, lectura, actualització i eliminació (CRUD) utilitzant mètodes i instàncies de model, en lloc de construir consultes de manera manual. Aquesta abstracció simplifica enormement el desenvolupament i redueix la probabilitat d'errors.

Finalment, la gran comunitat i l'extensa documentació que envolten Mongoose el converteixen en una opció sòlida per a desenvolupadors que treballen amb Node.js i MongoDB. La disponibilitat de nombrosos recursos, exemples i llibreries complementàries facilita la resolució de problemes i l'aprenentatge de la llibreria, accelerant el procés de desenvolupament i augmentant la productivitat.

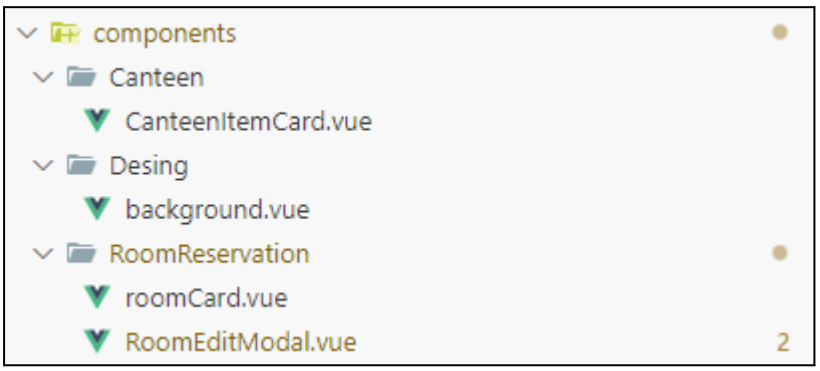
El nostre model amb Mongoose és el següent:

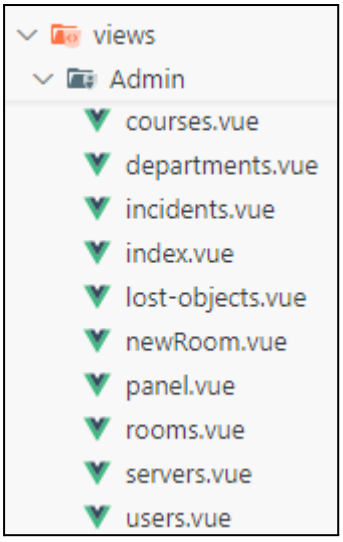
Message

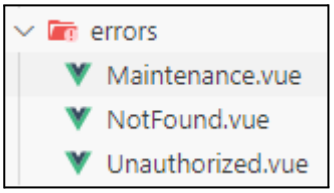
```
import mongoose from 'mongoose';

const interactionSchema = new mongoose.Schema({
    teacherId: {
        type: String,
        required: true
    },
    message: {
        type: String,
        required: true
    },
    hasLinks: {
        type: Boolean,
        default: false
    },
    links: {
        type: [String],
        default: []
    },
    linkPreviews: {
        type: [{
            url: String,
            title: String,
            description: String,
            image: String,
            siteName: String
        }],
        default: []
    },
    date: {
        type: Date,
        default: Date.now
    }
});

const messageSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    teachers: {
        type: [Number],
        default: []
    },
    interaction: {
        type: [interactionSchema],
        default: []
    },
    deletedFor: {
        type: [Number],
        default: []
    }
}, {
    timestamps: true,
    collection: 'messages'
});

const Message = mongoose.model("Message", messageSchema);

export default Message;
```

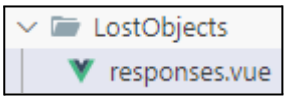
Utilitza MongoDB (no Sequelize) per emmagatzemar dades de xat/missatgeria entre professors, incloent contingut del missatge, enllaços i marques de temps.

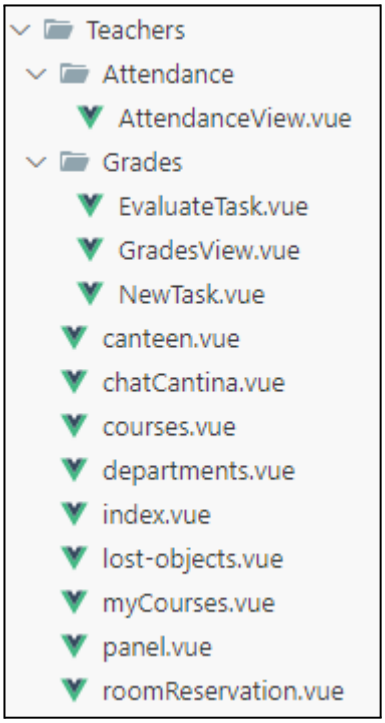
5. Esquema de components

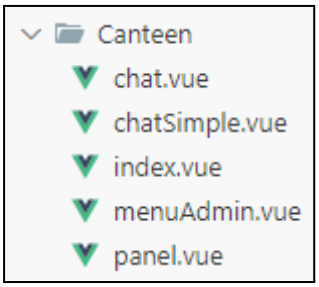


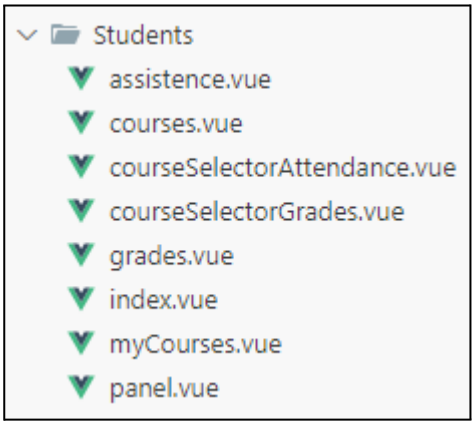


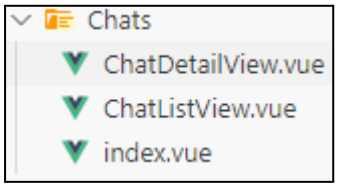












La nostra aplicació Conexus Hub està organitzada en diferents vistes segons el rol de l'usuari:

1. Vistes d'Usuari per Rol:

- **Admin:** Panel d'administració amb accés a gestió d'usuaris, departaments, cursos, informes i configuració general.
- **Teacher (Professor):** Espai per gestionar cursos, qualificacions, assistència, departaments, incidències i comunicació.
- **Student (Estudiant):** Interfície per veure notes, assistència, cursos disponibles, incidències i objectes perduts.
- **Technician (Tècnic):** Eina per gestionar incidències, assignacions, solucions tècniques i anàlisi.
- **Canteen (Cantina):** Gestió del menú, comandes i comunicació amb els professors.

2. Vistes de Funcionalitat:

- **Panel:** Cada rol té el seu propi dashboard amb resum d'informació rellevant i accions ràpides.
- **Chats:** Sistema de missatgeria entre professors i també amb la cantina.
- **Incidents:** Gestió d'incidències tècniques, amb vistes diferents per a cada rol.
- **LostObjects:** Gestió d'objectes perduts amb publicació i respostes.
- **Grades & Assistance:** Gestió i visualització de notes i assistència per a professors i estudiants.
- **Rooms:** Gestió i reserva d'aules i espais.

3. Vistes d'Error:

- **NotFound:** Pàgina 404 personalitzada.
- **Unauthorized:** Per a intents d'accés a àrees restringides.
- **Maintenance:** Mostra informació quan un servei està en manteniment.

Components Principals

L'aplicació utilitza diversos components reutilitzables:

Components de Disseny:

- Fons amb efectes visuals (background.vue).
- Barres laterals responsives a tots els layouts.

Components Funcionals:

- **CanteenItemCard:** Mostra productes de la cantina.
- **RoomCard:** Targetes d'informació d'aules.
- **RoomEditModal:** Modal per editar aules.

Components d'Estructura:

- Barres de navegació adaptatives (escriptori/mòbil).
- Headers específics per a cada secció.
- Formularis i modals estandarditzats.

Sistema de Routing

El router de l'aplicació (index.js) gestiona la navegació i els permisos:

Característiques principals:

Rutes Anidades: Estructura jeràrquica amb rutes pare (roles) i filles (funcionalitats).

- Exemple: /teachers/grades, /students/incidents.

Control d'Accés:

- Meta-dades a cada ruta amb requiresAuth i allowedRoles.
- Interceptors que comproven autenticació i permisos abans de permetre navegació.

Detecció de Serveis:

- Comprova l'estat dels serveis del backend (com chat, grades, canteen).
- Redirigeix a la pàgina de manteniment quan un servei no està disponible.

Implementació Lazy-loading:

- Les vistes es carreguen sota demanda per millorar el rendiment.

Flux d'Interacció

1. L'usuari s'autentica a la pàgina principal.
2. El router detecta el seu rol i el dirigeix al dashboard corresponent.
3. Els menús i opcions disponibles s'adapten segons els seus permisos.
4. Quan navega, el router comprova els permisos i l'estat dels serveis.
5. Les vistes carreguen dinàmicament els components necessaris.

Aquesta arquitectura permet una experiència fluïda i adaptada a cada tipus d'usuari, assegurant que només puguin accedir a les funcionalitats pertinents al seu rol dins de l'ecosistema Conexus Hub.

6. Documentació de codi front-end

```
<template>
    <!-- Contingut Principal -->
    <div class="animate-fade-in mt-8 mb-8">
        <!-- Secció de Capçalera -->
        <div class="mb-12 text-center">
            <h1 class="text-4xl font-bold bg-clip-text text-transparent
bg-gradient-to-r from-blue-400 via-purple-400 to-pink-400">
                Reserva de Sales
            </h1>
            <p class="text-gray-300 mt-3 text-lg">Selecciona una sala i
fes la teva reserva</p>
        </div>

        <!-- Llistat de Sales Disponibles -->
        <div class="max-w-4xl mx-auto px-4 space-y-6">
                <!-- Per cada sala disponible, crearem una targeta amb
animació d'entrada escalonada -->
                        <div v-for="room in rooms" :key="room.id"
class="animate-fade-in"
                    :style="{ animationDelay: `${rooms.indexOf(room) *
150}ms` }">
                <!-- Component roomCard que mostra informació de cada
sala -->
                                                <roomCard :room="room"
@open-reservation="openReservationModal" />
            </div>
        </div>

         <!-- Estat de buit: Es mostra quan no hi ha sales disponibles
-->
        <div v-if="rooms.length === 0" class="text-center py-16">
                <div class="bg-slate-800/50 rounded-2xl p-8 max-w-md
mx-auto">
                    <svg class="w-16 h-16 mx-auto mb-4 text-slate-600"
fill="none" stroke="currentColor" viewBox="0 0 24 24">
                                        <path stroke-linecap="round"
stroke-linejoin="round" stroke-width="1"
                            d="M9.172 16.172a4 4 0 015.656 0M9 10h.01M15
10h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z" />
                </svg>
                        <p class="text-xl text-gray-400">No hi ha sales
disponibles en aquest moment</p>
            </div>
        </div>
    </div>

     <!-- Modal de Reserva - Utilitzem Teleport per col·locar-lo fora
del flux normal del DOM -->
    <Teleport to="body">
        <div v-if="showModal" class="modal">
                <!-- Fons fosc amb efecte de desenfocament que es pot
clicar per tancar -->
            <div class="modal-backdrop" @click="closeModal"></div>
```

```
            <!-- Contingut del Modal -->
            <div class="modal-content">
                <!-- Capçalera del Modal amb títol i botó de tancar -->
                <div class="modal-header">
                            <h3 class="modal-title">Nova Reserva - {{
selectedRoom?.room_name }}</h3>
                    <button class="close-button" @click="closeModal">
                                        <svg class="w-5 h-5" fill="none"
stroke="currentColor" viewBox="0 0 24 24">
                                            <path stroke-linecap="round"
stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12" />
                        </svg>
                    </button>
                </div>

                <!-- Cos del Modal amb formulari de reserva -->
                <div class="modal-body">
                      <!-- Selector de data amb event que filtra hores
disponibles -->
                    <div class="form-group">
                        <label class="form-label">Data</label>
                            <input type="date" v-model="selectedDate"
@change="filterHoursByDate"
                                class="form-input" />
                    </div>

                       <!-- Selector d'hores disponibles segons la data
seleccionada -->
                    <div class="form-group">
                        <label class="form-label">Hora</label>
                                        <select v-model="selectedHour"
class="form-select">
                                    <option value="" disabled>Selecciona una
hora</option>
                                       <option v-for="hour in filteredHours"
:key="hour" :value="hour">
                                        {{ hour }}
                                </option>
                        </select>
                    </div>
                </div>

                <!-- Peu del Modal amb botons d'acció -->
                <div class="modal-footer">
                    <button @click="closeModal" class="cancel-button">
                        Cancel·lar
                    </button>
                       <!-- Botó de confirmació només actiu quan hi ha
data i hora seleccionades -->
                     <button @click="reserveRoom" v-if="selectedDate &&
selectedHour" class="confirm-button">
                        Confirmar Reserva
                    </button>
                </div>
            </div>
        </div>
    </Teleport>
</template>
```

```
<script setup>
// Importacions necessàries
import roomCard from "@/components/RoomReservation/roomCard.vue"; // Component de targeta de sala
import { onMounted, ref, computed } from "vue"; // Funcions de Vue 3 Composition API
import { getAllRooms, getReservationsFromRoom, createNewReservation } from "@/services/communicationsScripts/roomReservationsComManager"; // Serveis de comunicació API
import { useAppStore } from '@/stores/index.js'; // Store Pinia per gestionar l'estat de l'aplicació

// Inicialització de l'store per accedir a les dades de l'usuari
const appStore = useAppStore();

// Estats reactius
const rooms = ref([]); // Llista de sales disponibles
const showModal = ref(false); // Control de visibilitat del modal
const selectedRoom = ref(null); // Sala seleccionada per a reservar
const selectedDate = ref(""); // Data seleccionada en el formulari
const selectedHour = ref(""); // Hora seleccionada en el formulari
const filteredHours = ref([]); // Hores disponibles després de filtrar
const reservations = ref([]); // Reserves existents per a la sala seleccionada

/**
 * Carrega la llista de sales disponibles quan es munta el component
 */
onMounted(async () => {
    // Obté totes les sales des de l'API
    rooms.value = await getAllRooms();
    // Filtra només les sales disponibles
    rooms.value = rooms.value.filter((room) => room.available == true);
});

/**
 * Obre el modal de reserva per a una sala específica
 * @param {Object} room - L'objecte sala seleccionat per l'usuari
 */
function openReservationModal(room) {
    selectedRoom.value = room;
    showModal.value = true;
     loadReservations(); // Carrega les reserves existents per aquesta sala
}

/**
 * Carrega les reserves existents per a la sala seleccionada
 */
async function loadReservations() {
    if (selectedRoom.value) {
                                        reservations.value = await getReservationsFromRoom(selectedRoom.value.id);
    }
}

/**
```

```
 * Tanca el modal i reinicia tots els valors del formulari
 */
function closeModal() {
    showModal.value = false;
    selectedRoom.value = null;
    selectedDate.value = "";
    selectedHour.value = "";
    filteredHours.value = [];
}

/**
 * Filtra les hores disponibles segons la data seleccionada
 * Té en compte tant l'horari disponible de la sala com les reserves existents
 */
function filterHoursByDate() {
    if (!selectedRoom.value || !selectedDate.value) return;

    // Obté el dia de la setmana (0=diumenge, 1=dilluns, ...)
    const day = new Date(selectedDate.value).getDay();
      selectedHour.value = ""; // Reinicia l'hora seleccionada quan canvia la data

    // Selecciona les hores disponibles segons el dia de la setmana
    let availableHours = [];
    switch (day) {
        case 1: // Dilluns
                                                    availableHours = selectedRoom.value.room_hours_available_monday || [];
            break;
        case 2: // Dimarts
                                                    availableHours = selectedRoom.value.room_hours_available_tuesday || [];
            break;
        case 3: // Dimecres
                                                    availableHours = selectedRoom.value.room_hours_available_wednesday || [];
            break;
        case 4: // Dijous
                                                    availableHours = selectedRoom.value.room_hours_available_thursday || [];
            break;
        case 5: // Divendres
                                                    availableHours = selectedRoom.value.room_hours_available_friday || [];
            break;
        default: // Caps de setmana no disponibles
            availableHours = [];
    }

    // Filtra les hores que ja estan reservades en aquesta data
    filteredHours.value = availableHours.filter((hour) => {
        return !reservations.value.some((reservation) => {
            const reservedDate = new Date(reservation.start_time);
            return (
                // Compara si és el mateix dia
                                reservedDate.toDateString() === new Date(selectedDate.value).toDateString() &&
```

```
                // Compara si és la mateixa hora
                                        reservedDate.getHours()   ===
parseInt(hour.split(":")[0])
            );
        });
    });
}

/**
 * Crea una nova reserva amb les dades seleccionades
 */
function reserveRoom() {
    // Validació de dades necessàries
            if   (!selectedRoom.value   ||   !selectedDate.value   ||
!selectedHour.value) return;

    // Crea l'objecte de reserva
    const newReservation = {
        room_id: selectedRoom.value.id,
                                                start_time:      new
Date(`${selectedDate.value}T${selectedHour.value.split("-")[0]}`),
                                                 end_time:       new
Date(`${selectedDate.value}T${selectedHour.value.split("-")[1]}`),
        user_id: appStore.user.id, // Utilitza l'ID de l'usuari actual
    };

    // Envia la petició de reserva a l'API
    createNewReservation(newReservation)
        .then((response) => {
            console.log("Reserva creada:", response);
            closeModal();
                  // Aquí es podria afegir una notificació d'èxit o
recarregar les reserves
        })
        .catch((error) => {
            console.error("Error al crear la reserva:", error);
            // Aquí es podria afegir una notificació d'error
        });
}
</script>
```

7. Documentació de codi back-end

```
/**
 * Gestor de Serveis
 *
 * Aquest mòdul gestiona els microserveis de l'aplicació Conexus Hub.
 * Permet iniciar, aturar, monitoritzar i administrar serveis Node.js
 * de forma dinàmica i centralitzada.
 */

// Importacions de mòduls
import { spawn } from 'node:child_process';      // Utilitzat per crear
processos fills
import fs from 'fs';                             // Sistema de fitxers
import path from 'path';                         // Manipulació de rutes
import { fileURLToPath } from 'url';                 // Conversió URL a
rutes de fitxers
import pidusage from 'pidusage';                  // Monitorització d'ús
de CPU i memòria

// Configuració d'entorn
const isDevelopment = process.env.NODE_ENV === 'development';
const executor = isDevelopment ? 'nodemon' : 'node';   // Utilitza
nodemon en desenvolupament per hot-reload

// Resolució de ruta del directori actual en mòduls ES
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

console.log(`Executant en mode ${isDevelopment ? 'desenvolupament' :
'producció'}`);

// Fitxers que no s'han de tractar com a serveis
const excludedFiles = ['index.js', 'serviceManager.js', 'token.js',
'Dockerfile', 'prod.Dockerfile'];
const services = {};  // Registre de tots els serveis disponibles

// Descobriment automàtic de serveis a partir dels fitxers JavaScript
del directori
fs.readdirSync(__dirname)
                    .filter(file     =>     file.endsWith('.js')     &&
!excludedFiles.includes(file))
    .forEach(file => {
        // Per cada fitxer JS vàlid, registra'l com a servei disponible
        const serviceName = path.basename(file, '.js');
        services[serviceName] = {
                state: "stopped",                        // Estat inicial:
aturat
                process: null,                            // Referència al
procés quan s'executa
             script: file,                           // Nom del fitxer del
servei
                usage: 0,                                      // Ús de CPU
(s'actualitza periòdicament)
                    description: getServiceDescription(serviceName)   //
Informació descriptiva
```

```
        };
    });

/**
 * Proporciona informació descriptiva per a cada servei conegut
 * @param {string} serviceName - Nom del servei
 * @returns {Object} Objecte amb nom i tecnologia del servei
 */
function getServiceDescription(serviceName) {
    const serviceDescriptions = {
        'assistences': { name: 'Servei d\'Assistència', tech: 'Node.js
18.x' },
        'incidents': { name: 'Gestor d\'Incidències', tech: 'Node.js
18.x' },
        'grades': { name: 'Sistema de Notes', tech: 'Node.js 18.x' },
        'rooms': { name: 'Gestió d\'Aules', tech: 'Node.js 18.x' },
        'lostObjects': { name: 'Objectes Perduts', tech: 'Node.js 18.x'
},
        'canteen': { name: 'Servei de Menjador', tech: 'Node.js 18.x'
},
        'chat': { name: 'Xat', tech: 'Node.js 18.x' },
    };

    // Si no hi ha descripció personalitzada, en genera una genèrica
    return serviceDescriptions[serviceName] || { name: `Servei
${serviceName}`, tech: 'Node.js 18.x' };
}

/**
 * Actualitza les estadístiques d'ús de recursos per a tots els serveis
en execució
 * Es crida periòdicament per monitoritzar CPU i memòria
 */
async function updateServiceUsage() {
    for (const [name, service] of Object.entries(services)) {
        if (service.state === "running" && service.process &&
service.process.pid) {
            try {
                // Obtenció d'estadístiques de procés via pidusage
                const stats = await pidusage(service.process.pid);

                // Actualització de l'ús de CPU arrodonit
                service.usage = Math.round(stats.cpu);

                // Emmagatzematge d'estadístiques detallades
                service.stats = {
                    memory: Math.round(stats.memory / 1024 / 1024),  //
Convertit a MB
                    cpu: Math.round(stats.cpu),
// Percentatge d'ús de CPU
                    elapsed: stats.elapsed,
// Temps en execució
                    timestamp: Date.now()
// Marca temporal
                };
            } catch (error) {
                console.error(`Error obtenint estadístiques per
${name}:`, error);
```

```
                service.usage = 0;
                service.stats = null;

                // Comprova si el procés encara existeix
                if (service.process) {
                    try {
                                process.kill(service.process.pid, 0);  //
Senyal 0 per comprovar existència
                    } catch (e) {
                        // El procés ja no existeix, actualitza l'estat
                        service.state = "stopped";
                        service.process = null;
                    }
                }
            }
        } else {
            // El servei no s'està executant, ressetejar estadístiques
            service.usage = 0;
            service.stats = null;
        }
    }
}

// Configura l'actualització periòdica d'estadístiques cada 5 segons
setInterval(updateServiceUsage, 5000);

/**
 * Inicia un servei específic
 * @param {string} serviceName - Nom del servei a iniciar
  * @returns {Object} Resultat de l'operació amb indicació d'èxit i
missatge
 */
export function startService(serviceName) {
    // Recupera la informació del servei
    const service = services[serviceName];

    // Comprova si el servei existeix
    if (!service) {
        return { success: false, message: `Servei ${serviceName} no
trobat` };
    }

    // Comprova si el servei ja s'està executant
    if (service.state === "running") {
        return { success: true, message: `Servei ${serviceName} ja
s'està executant` };
    }

    try {
                    console.log(`Iniciant servei ${serviceName} amb
${executor}...`);
             // Inicia el procés com a fill, utilitzant nodemon en
desenvolupament
        const process = spawn(executor, [service.script, '-L']);

        // Actualitza l'estat i guarda la referència al procés
        service.process = process;
        service.state = "running";
```

```
        // Gestió de la sortida estàndard del procés
        process.stdout.on('data', data => {
            console.log(`[${serviceName}] ${data.toString().trim()}`);
        });

        // Gestió de la sortida d'error del procés
        process.stderr.on('data', data => {
                                    console.error(`[${serviceName}   ERROR]
${data.toString().trim()}`);
        });

        // Gestió de finalització del procés
        process.on('close', code => {
            // Ignora el codi 0 en desenvolupament (nodemon restart)
            if (!(isDevelopment && code === 0)) {
                    console.log(`Servei ${serviceName} tancat amb codi
${code}`);
                service.state = 'stopped';
                service.process = null;
            }
        });

         return { success: true, message: `Servei ${serviceName} iniciat
correctament amb ${executor}` };
    } catch (error) {
        console.error(`Error en iniciar ${serviceName}:`, error);
           return { success: false, message: `Error en iniciar servei
${serviceName}` };
    }
};

/**
 * Atura un servei específic
 * @param {string} serviceName - Nom del servei a aturar
  * @returns {Object} Resultat de l'operació amb indicació d'èxit i
missatge
 */
export function stopService(serviceName) {
    const service = services[serviceName];

    // Comprova si el servei existeix
    if (!service) {
        return { success: false, message: `Servei ${serviceName} no
trobat` };
    }

    // Comprova si el servei ja està aturat
    if (service.state === "stopped") {
         return { success: true, message: `Servei ${serviceName} ja està
aturat` };
    }

    try {
        console.log(`Aturant servei ${serviceName}...`);
        // Envia senyal SIGTERM al procés
        service.process.kill();
        // Neteja referències i estat
```

```
        service.process = null;
        service.state = "stopped";
        service.usage = 0;
         return { success: true, message: `Servei ${serviceName} aturat
correctament` };
    } catch (error) {
        console.error(`Error en aturar ${serviceName}:`, error);
            return { success: false, message: `Error en aturar servei
${serviceName}` };
    }
};

/**
 * Inicia tots els serveis registrats
 * @returns {Object} Diccionari amb els resultats per cada servei
 */
export function startAllServices() {
    const results = {};

    // Itera sobre tots els serveis i els inicia
    for (const serviceName of Object.keys(services)) {
        results[serviceName] = startService(serviceName);
    }

    return results;
};

/**
 * Atura tots els serveis que estan en execució
  *  @returns {Object} Diccionari amb els resultats per cada servei
aturat
 */
export function stopAllServices() {
    const results = {};

    // Itera sobre tots els serveis i atura els que estan en execució
    for (const serviceName of Object.keys(services)) {
        if (services[serviceName].state === "running") {
            results[serviceName] = stopService(serviceName);
        }
    }

    return results;
};

/**
 * Obté l'estat d'un servei específic
 * @param {string} serviceName - Nom del servei
 * @returns {Object|null} Informació de l'estat del servei o null si no
existeix
 */
export function getServiceStatus(serviceName) {
    if (serviceName && services[serviceName]) {
        return {
            name: serviceName,                                    // Nom
tècnic
            displayName: services[serviceName].description.name, // Nom
amigable
```

```
                tech: services[serviceName].description.tech,          //
Tecnologia
                state: services[serviceName].state,                    //
Estat (running/stopped)
                usage: services[serviceName].usage,                  // Ús
de CPU
                stats: services[serviceName].stats,                    //
Estadístiques detallades
                port: services[serviceName].port                       //
Port d'execució
            };
        }
        return null;
};

/**
 * Obté l'estat de tots els serveis registrats
 * @returns {Object} Diccionari amb l'estat de cada servei
 */
export function getAllServicesStatus() {
    const status = {};

    // Itera sobre tots els serveis i recopila el seu estat
    for (const [name, service] of Object.entries(services)) {
        status[name] = {
            displayName: service.description.name,
            tech: service.description.tech,
            state: service.state,
            usage: service.usage,
            stats: service.stats,
            port: service.port
        };
    }

    return status;
};

/**
 * Afegeix un nou servei al gestor
 * @param {string} serviceName - Nom del nou servei
 * @param {string} scriptName - Nom del fitxer de script
 * @param {Object} description - Informació descriptiva opcional
 * @returns {Object} Resultat de l'operació
 */
export function addService(serviceName, scriptName, description = {}) {
    // Comprova si el servei ja existeix
    if (services[serviceName]) {
         return { success: false, message: `El servei ${serviceName} ja
existeix` };
    }

    try {
        const scriptPath = path.join(__dirname, scriptName);

        // Si el script no existeix, en crea un de bàsic
        if (!fs.existsSync(scriptPath)) {
            // Genera un port aleatori si no es proporciona
```

```
                const port = description.port || Math.floor(3100 +
Math.random() * 900);

            // Plantilla bàsica per a un nou servei Express
            const basicScript = `
import express from 'express';
import cors from 'cors';

const app = express();
const PORT = process.env.PORT || ${port};

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

app.get('/', (req, res) => {
  res.json({
    service: '${serviceName}',
    status: 'running',
    time: new Date().toISOString(),
    environment: process.env.NODE_ENV || 'development'
  });
});

app.get('/health', (req, res) => {
  const memoryUsage = process.memoryUsage();
  res.json({
    status: 'ok',
    pid: process.pid,
    memory: {
      rss: Math.round(memoryUsage.rss / 1024 / 1024),
      heapTotal: Math.round(memoryUsage.heapTotal / 1024 / 1024),
      heapUsed: Math.round(memoryUsage.heapUsed / 1024 / 1024),
    },
    uptime: process.uptime()
  });
});

app.listen(PORT, () => {
  console.log(\`Servei ${serviceName} executant-se al port \${PORT}\`);
});
`;
            // Escriu el nou fitxer de script
            fs.writeFileSync(scriptPath, basicScript, 'utf8');
        }

        // Registra el nou servei al gestor
        services[serviceName] = {
            state: "stopped",
            process: null,
            script: scriptName,
            usage: 0,
            stats: null,
            port: description.port,
            description: {
                name: description.name || `Servei ${serviceName}`,
                tech: description.tech || 'Node.js 18.x'
            }
        };
```

```
        console.log(`Servei ${serviceName} creat correctament`);
          return { success: true, message: `Servei ${serviceName} creat
correctament` };
    } catch (error) {
        console.error(`Error en crear servei ${serviceName}:`, error);
           return { success: false, message: `Error en crear servei:
${error.message}` };
    }
}

/**
 * Elimina un servei del gestor
 * @param {string} serviceName - Nom del servei a eliminar
 * @returns {Object} Resultat de l'operació
 */
export function deleteService(serviceName) {
    const service = services[serviceName];

    // Comprova si el servei existeix
    if (!service) {
           return { success: false, message: `Servei ${serviceName} no
trobat` };
    }

    try {
        // Si el servei està en execució, l'atura primer
        if (service.state === "running") {
            stopService(serviceName);
        }

        // Elimina el fitxer del script si existeix
        const scriptPath = path.join(__dirname, service.script);
        if (fs.existsSync(scriptPath)) {
            fs.unlinkSync(scriptPath);
        }

        // Elimina el servei del registre
        delete services[serviceName];

        console.log(`Servei ${serviceName} eliminat correctament`);
             return { success: true, message: `Servei ${serviceName}
eliminat correctament` };
    } catch (error) {
             console.error(`Error en eliminar servei ${serviceName}:`,
error);
          return { success: false, message: `Error en eliminar servei:
${error.message}` };
    }
}
```

Aquest gestor de serveis actua com a orquestrador per als diferents microserveis de l'aplicació Conexus Hub. Permet descobrir, iniciar, aturar i monitoritzar cada servei de forma centralitzada, proporcionant una interfície unificada per a l'administració de l'aplicació.

8. Desplegament Contenidors

8.1 Desplegament Local

Visió General de l'Arquitectura

L'entorn de desenvolupament de Conexus Hub és una arquitectura basada en contenidors Docker que proporciona tots els components necessaris per desenvolupar i provar l'aplicació. Està compost per:

1. **Frontend (Vue.js)** - Interfície d'usuari
2. **Backend (Node.js)** - API i serveis
3. **MySQL** - Base de dades principal
4. **MongoDB** - Base de dades secundària (per al xat)
5. **Eines d'administració** - Adminer i Mongo Express

Com Aixecar l'Entorn de Desenvolupament

Requisits Previs

- Docker i Docker Compose instal·lats al teu sistema.
- Git per clonar el repositori.

Passos per Aixecar els Contenidors

1. **Clona el repositori** (si encara no ho has fet):
    a. git clone <url-del-repositori>
    b. cd prj-final-front-back-g1-conexus
2. **Configura les variables d'entorn** (ja hauria d'estar configurat en el fitxer .env)
3. **Aixeca tots els contenidors:**
    a. docker compose up --build

Accés als Serveis

Un cop els contenidors estan en marxa, pots accedir a:

- **Frontend:** http://localhost:5173
- **Backend API:** http://localhost:3000
- **Adminer (gestor MySQL):** http://localhost:8080

    - Sistema: MySQL
    - Servidor: conexus-hub-mysql
    - Usuari: root
    - Contrasenya: root
    - Base de dades: conexushub
- **Mongo Express:** http://localhost:8081

Estructura dels Contenidors

El fitxer compose.yaml defineix els següents serveis:

1. **conexus-hub-front:**
    - Serveix l'aplicació Vue.js en mode desenvolupament.
    - Port: 5173.
    - Hot-reloading activat (els canvis es veuen immediatament).
2. **conexus-hub-back:**
    - Executa el backend Node.js.
    - Ports: 3000-3007 (un per cada servei).
    - Nodemon per reiniciar automàticament quan es detecten canvis.
3. **conexus-hub-mysql:**
    - Base de dades MySQL amb scripts d'inicialització.
    - Els scripts SQL crearan automàticament l'estructura de la BD i inseriran dades inicials.
4. **conexus-hub-adminer:**
    - Eina de gestió visual per MySQL.
5. **conexus-hub-mongodb:**
    - Base de dades MongoDB per al servei de xat.
6. **conexus-hub-mongo-express:**
    - Eina de gestió visual per MongoDB.

Característiques de l'Entorn de Desenvolupament

- **Volums muntats:** El codi font està enllaçat amb els contenidors, cosa que permet veure els canvis en temps real.
- **Hot Reloading:** Tant el front-end com el back-end es reinicien automàticament quan es detecten canvis.
- **Dependències encapsulades:** Totes les dependències s'instal·len dins dels contenidors, evitant problemes de "funciona al meu ordinador".

- **Persistència de dades:** Les dades de MySQL i MongoDB es mantenen en volums Docker fins i tot quan els contenidors es reinicien.

Dockerfile back

```
FROM node:23-alpine3.20

WORKDIR /app

RUN apk add --no-cache bash

COPY package*.json ./

RUN npm ci

RUN npm install -g nodemon

COPY . .

EXPOSE 3000
```

Dockerfile front

```
FROM node:23-alpine3.20

WORKDIR /app

RUN apk add --no-cache bash

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5173

ENV NODE_ENV=development

CMD ["npm", "run", "dev"]
```

Compose

```
services:
  conexus-hub-front:
```

```
  build:
    context: ./front-vue
    dockerfile: Dockerfile
  ports:
    - "5173:5173"
  volumes:
    - ./front-vue:/app
    - /app/node_modules
  working_dir: /app
  environment:
    - NODE_ENV=development
    - CHOKIDAR_USEPOLLING=true
    - HOST=0.0.0.0
  env_file:
    - .env
  restart: unless-stopped
  networks:
    - conexus-hub-network

conexus-hub-back:
  build:
    context: ./back-node/node
    dockerfile: Dockerfile
  ports:
    - "3000:3000"
    - "3001:3001"
    - "3002:3002"
    - "3003:3003"
    - "3004:3004"
    - "3005:3005"
    - "3006:3006"
    - "3007:3007"
  volumes:
    - ./back-node/node:/app
    - /app/node_modules
  working_dir: /app
  environment:
    - NODE_ENV=development
  env_file:
    - .env
  depends_on:
    conexus-hub-mysql:
      condition: service_healthy
  command: >
    sh -c "
      echo 'Esperando a que MySQL esté disponible...'
      while ! nc -z conexus-hub-mysql 3306; do
        sleep 1
      done
      echo 'MySQL está disponible, iniciando la aplicación...'
      npm run dev
    "
  # command: npm run dev # para desarrollo
  # command: npm run start # para producción
  restart: unless-stopped
  networks:
    - conexus-hub-network
  healthcheck:
```

```
      test: [ "CMD", "mysqladmin", "ping", "-h", "conexus-hub-mysql" ]
      interval: 30s
      timeout: 10s
      retries: 5

  conexus-hub-mysql:
    image: mysql:8.0
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=conexushub
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
        command:  --default-authentication-plugin=mysql_native_password
--character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
    volumes:
      - conexus-hub-mysql-data:/var/lib/mysql
                                                                        -
./back-node/data/00-charset.sql:/docker-entrypoint-initdb.d/00-charset.
sql
                                                                        -
./back-node/data/create_db.sql:/docker-entrypoint-initdb.d/01-create.sq
l
                                                                        -
./back-node/data/insert_db.sql:/docker-entrypoint-initdb.d/02-inserts.s
ql
    restart: unless-stopped
    networks:
      - conexus-hub-network
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s

  conexus-hub-adminer:
    image: adminer
    ports:
      - "8080:8080"
    restart: unless-stopped
    networks:
      - conexus-hub-network

  conexus-hub-mongodb:
    image: mongo
    restart: always
    container_name: conexus-hub-mongodb
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: password
    volumes:
      - mongo-db:/data/db
    networks:
      - conexus-hub-network
```

```
  conexus-hub-mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: password
                                                    ME_CONFIG_MONGODB_URL:
mongodb://root:password@conexus-hub-mongodb:27017/
      ME_CONFIG_BASICAUTH: false
    depends_on:
      - conexus-hub-mongodb
    networks:
      - conexus-hub-network

volumes:
  conexus-hub-mysql-data:
  mongo-db:


networks:
  conexus-hub-network:
    driver: bridge
```