Лабораторна робота № 11. Авторизація

Мета: навчитися використовувати JWT для API-авторизації.

Завдання

1) Ознайомитися з матеріалом лекції № 11;
2) Виконати пункти 1- ходу роботи;
3) У звіті відобразити кроки 1- ходу роботи та додати скріни роботи застосунку (відображення, створення, оновлення, видалення, пошук, фільтрація, навігація, реєстрація, вхід, відновлення пароля);
4) Надати відповіді на контрольні питання.

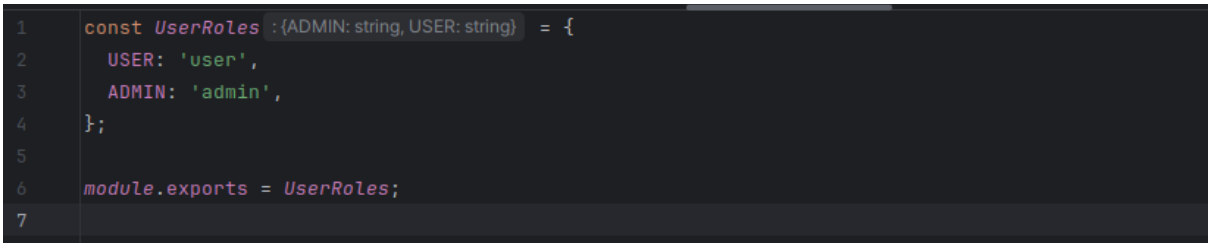
Хід роботи

1) Почнемо реалізацію зі сторони серверу. Перед нами стоять наступні задачі:
    a) Створити модель User;
    b) Зв’язати колекції tasks та users;
    c) Створити контролер для реєстрації, входу, відновлення паролю та підтвердження реєстрації;
    d) Створити сервіс відправки повідомлень на пошту;
    e) Створити middleware для захисту маршрутів;
    f) Оновити правила валідації для обробки форм реєстрації/входу/відновлення пароля.

Перед початком встановимо необхідні залежності, а саме bcrypt та jsonwebtoken (npm install package_name).

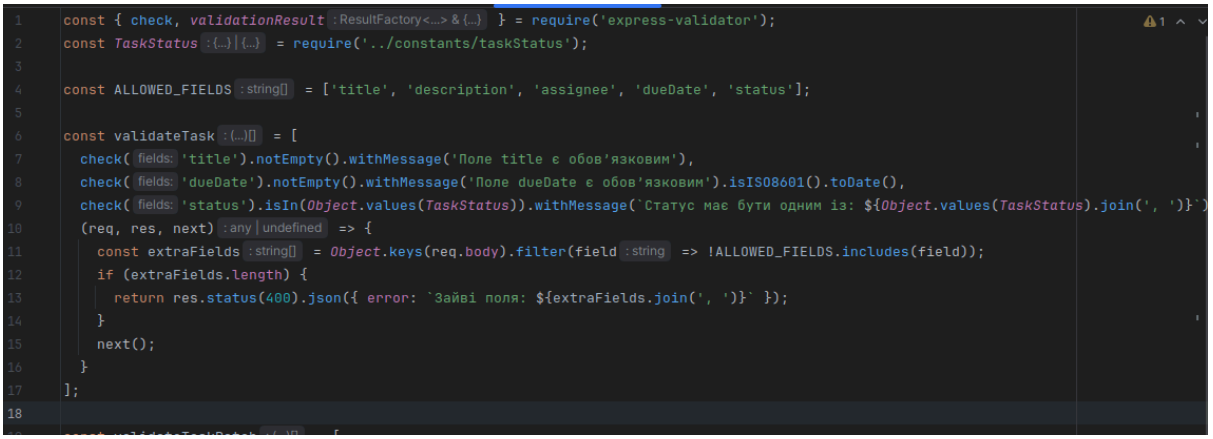
Почнемо зі створення нової моделі, у теці **models** створюємо файл **user.model.js**. Створюємо саму модель, вона буде містити поля: email, password, role, isConfirmed.

А також створимо константу UserRoles



Далі оновимо модель Task
Оновимо поле assignee, щоб воно було пов’язане з колекцією users

Але так як ми будемо отримувати ID користувача з токена (та тільки емулюємо рольову модель), а не з Angular, то для звичайного користувача це поле буде зайвим. Але у нас є логіка валідації, яка каже, що поле “assignee” є обов’язковим, тому ми його прибираємо.



```
const { check, validationResult } = require('express-validator');
const TaskStatus = require('../constants/taskStatus');

const ALLOWED_FIELDS = ['title', 'description', 'assignee', 'dueDate', 'status'];

const validateTask = [
  check('title').notEmpty().withMessage('Поле title є обов'язковим'),
  check('dueDate').notEmpty().withMessage('Поле dueDate є обов'язковим').isISO8601().toDate(),
  check('status').isIn(Object.values(TaskStatus)).withMessage(`Статус має бути одним із: ${Object.values(TaskStatus).join(', ')}`),
  (req, res, next) => {
    const extraFields = Object.keys(req.body).filter(field => !ALLOWED_FIELDS.includes(field));
    if (extraFields.length) {
      return res.status(400).json({ error: `Зайві поля: ${extraFields.join(', ')}` });
    }
    next();
  }
];
```

Далі реалізуємо логіку створення нового користувача. Для цього створимо файл **auth.controller.js**. Він буде відповідати за логіку реєстрації/входу/підтвердження/оновлення пароля/отримання поточного користувача.

Перед реалізацією нам необхідно зроби дві додаткові дії, а саме: встановити nodemailer та створити сервіс для відправки повідомлень на пошту. Спочатку встановимо необхідні залежності:

```
{
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node src/server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cors": "^2.8.5",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "express-validator": "^7.2.1",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.12.1",
    "nodemailer": "^6.10.0"
  }
}
```

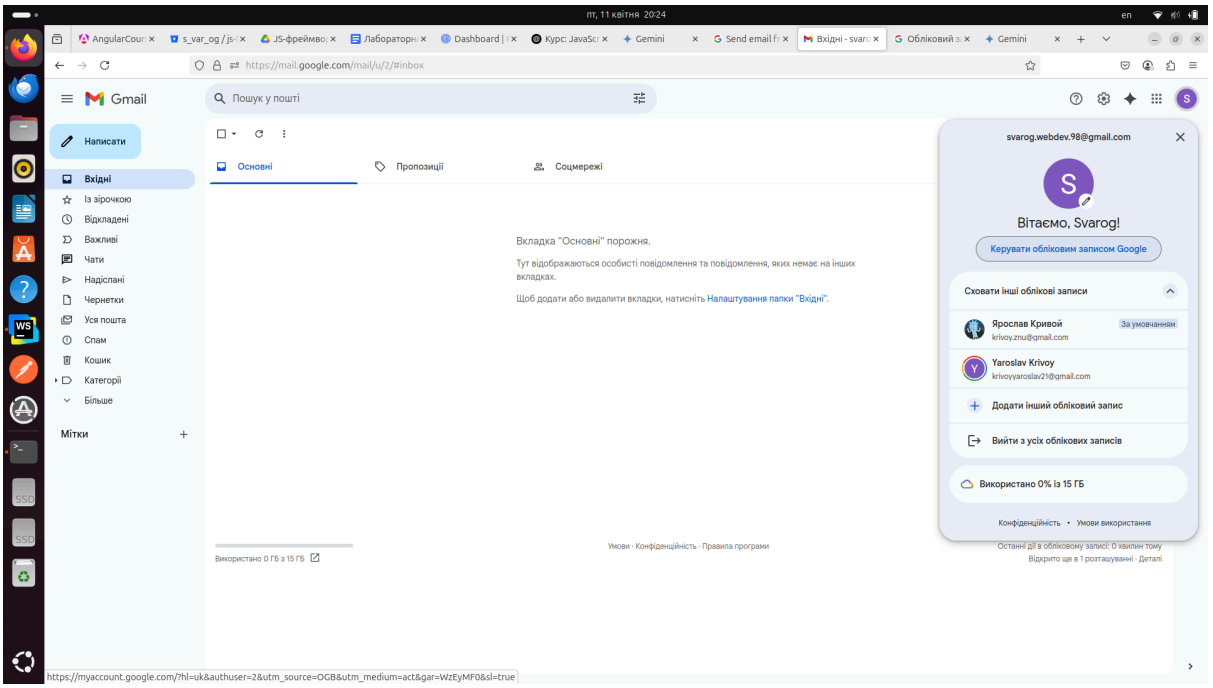
У файлі .env створимо наступні змінні

```
// основний токен для реєстрації/входу
JWT_SECRET = 'your_token'
// токен для підтвердження реєстрації
JWT_CONFIRM_SECRET = ''
// токен для оновлення пароля
JWT_RESET_SECRET=''

// credential для відправки листів
SMTP_EMAIL=''
SMTP_PASS=''

// URL клієнтської частини
FRONTEND_URL = 'http://localhost:4200'
```

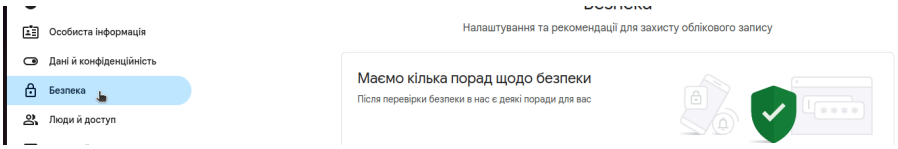
Згенерувати токен можна за допомогою команди **openssl rand -hex 32**, або можна використати сервіси для генерації https://jwtsecret.com/generate.

Далі необхідно отримати credential для SMTP. Для цього будемо використовувати Gmail (можна використати credential, які є у лабораторній).

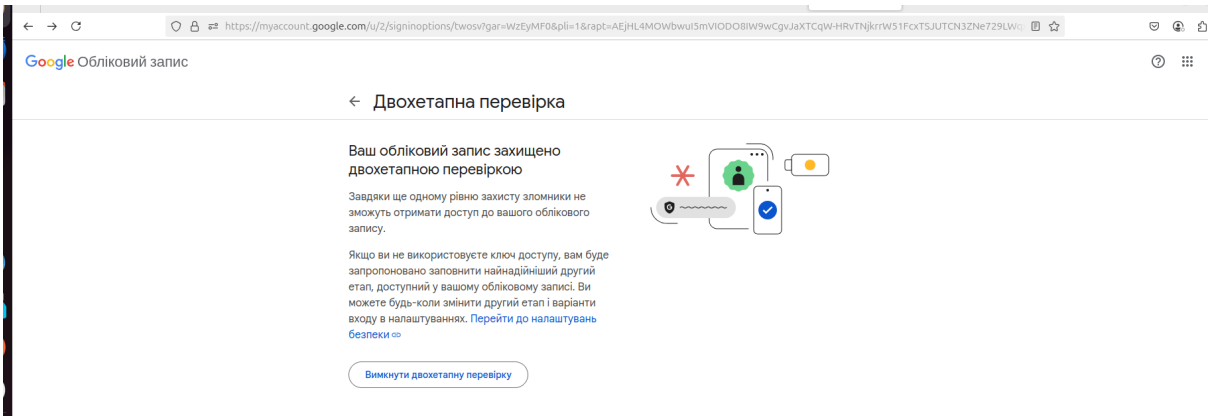
Переходимо до налаштувань профіля.



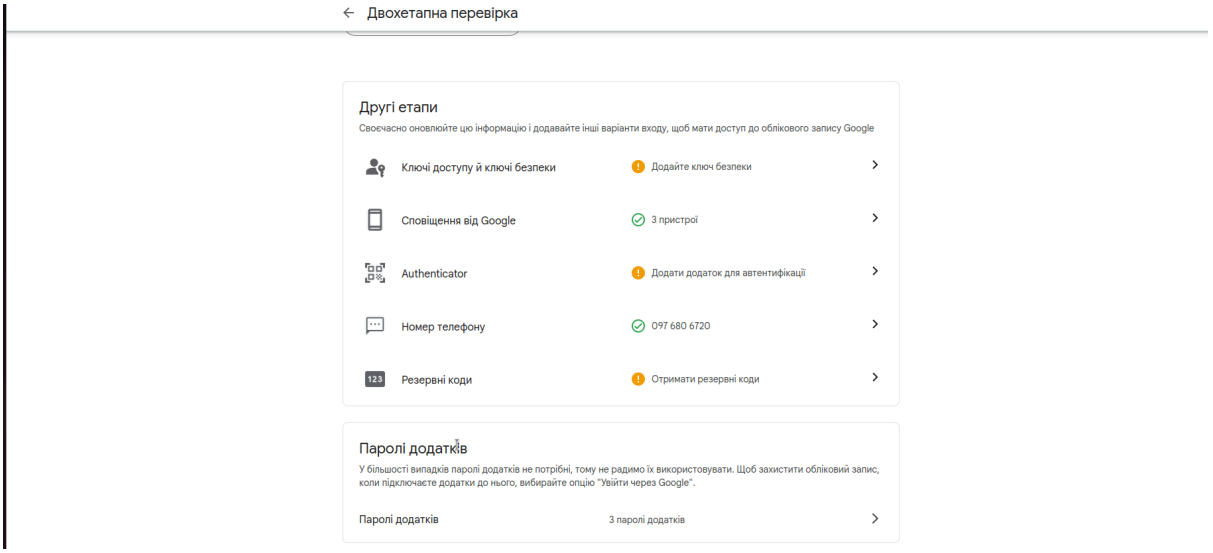
Обираємо “Безпека”



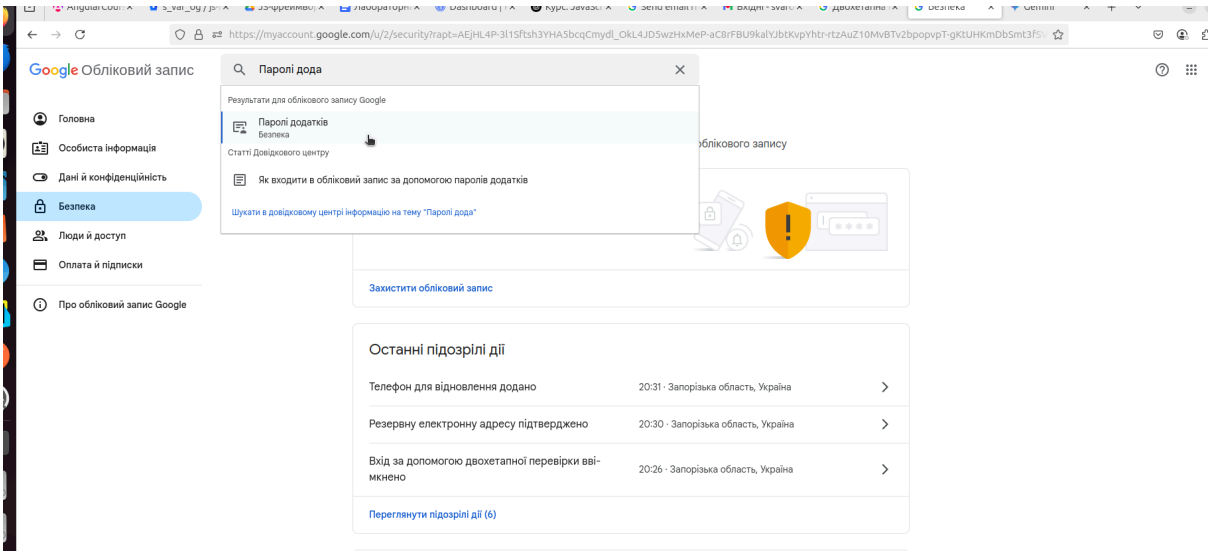
Налаштовуємо двоетапну перевірку

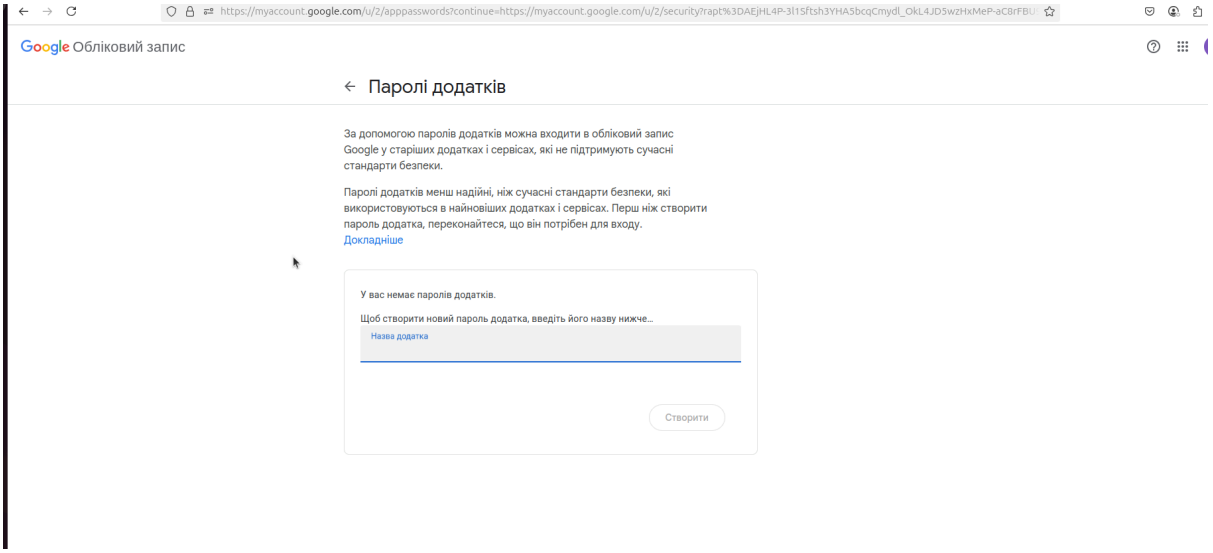


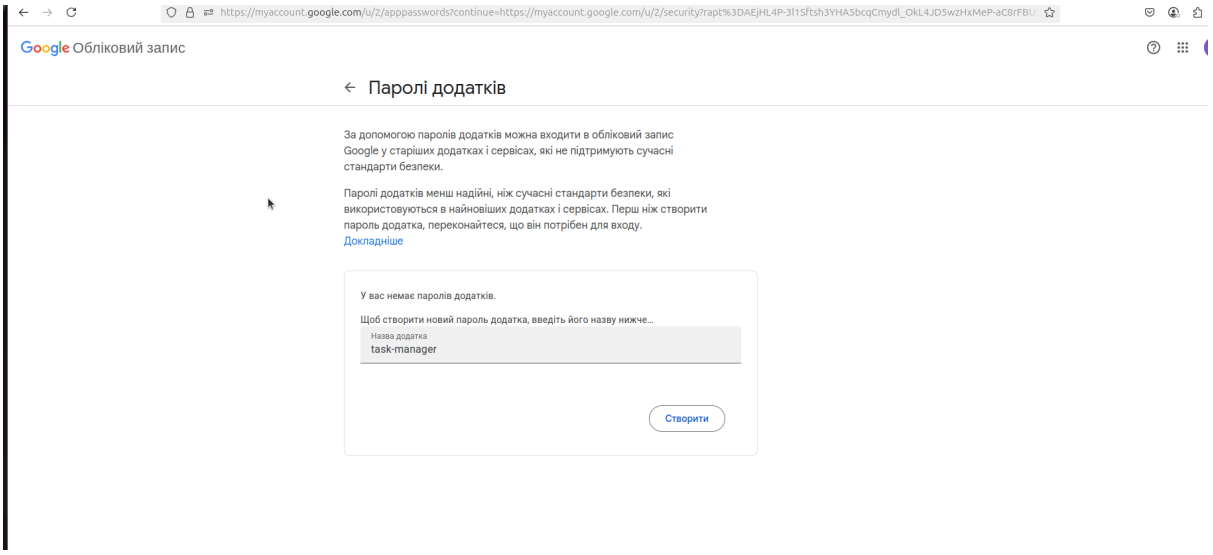
Після налаштування, нам необхідно перейти до пункту “Паролі додатків”



Якщо ви його у себе не бачите, то наберіть у пошуку







Та зберігаємо пароль додатка у .env

```
// credential для відправки листів
SMTP_EMAIL='svarog.webdev.98@gmail.com'
SMTP_PASS='dbst xrmt nxcq prxh'

// URL клієнтської частини
FRONTEND_URL = 'http://localhost:4200'
```

Переходимо до створення email-сервісу, у теці server/src/services створюємо файл email.service.js

```
require('dotenv').config();
const nodemailer :{} = require('nodemailer');

const transporter = nodemailer.createTransport({
  host: 'smtp.gmail.com',
  port: 587,
  secure: false,
  auth: {
    user: process.env.SMTP_EMAIL,
    pass: process.env.SMTP_PASS
  }
});

exports.sendConfirmationEmail = async (email, token) :Promise<void> => {
  const url :string = `${process.env.FRONTEND_URL}/${token}`;

  try {
    await transporter.sendMail({
      from: `"Task App" <${process.env.SMTP_EMAIL}>`,
      to: email,
      subject: 'Підтвердження реєстрації',
      html: `<p>Натисніть <a href="${url}">тут</a>, щоб підтвердити свій акаунт.</p>`
    });
    console.log('Email sent successfully');
  } catch (error) {
    console.error('Error sending email:', error);
  }
};

```

Повертаємося до контролера, почнемо з реєстрації

```
const bcrypt :{...} = require('bcrypt');
const jwt :{} = require('jsonwebtoken');
const UserModel :{} = require('../models/user.model');
const {sendConfirmationEmail} = require('../services/email.service');

const JWT_CONFIRM_SECRET :string = process.env.JWT_CONFIRM_SECRET;

const register = async (req, res) :Promise<...> => {  Show usages
  try {
    const {email, password} = req.body;

    if (!email || !password) {
      return res.status(400).send({message: 'Email і пароль обов'язкові'})
    }

    const existingUser = await UserModel.findOne({ email });

    if (existingUser) return res.status(400).json({ message: 'Користувач уже існує' });

    const passwordHash = await bcrypt.hash(password, salt: 10);

    const user = new UserModel({email, password: passwordHash});
    await user.save();

    const token = jwt.sign({ id: user._id }, JWT_CONFIRM_SECRET, { expiresIn: '1d' });

    await sendConfirmationEmail(email, token);

    res.status(201).json({ message: 'Реєстрація успішна. Перевірте пошту для підтвердження.' });
  } catch (error) {
    res.status(500).send({message: 'Виникла помилка при реєстрації.', errors: error})
  }
}
```

Відразу створимо auth.routes.js та зареєструємо його у app,js

```
const express :e | () => core.Express = require('express');
const router :Router = express.Router();

const authController :{...} | {...} = require('../controllers/auth.controller');

router.post( path: '/register', authController.register);

module.exports = router;
```

```
const authRoutes :Router | {...} = require('./routes/auth.routes');

connectDB();

const app :any | Express = express();

app.use(cors());

app.use(express.json());
app.use(express.urlencoded( options: { extended: true }));


app.use('/api/v1/tasks', taskMockRoutes);
app.use('/api/v2/tasks', taskRoutes);
app.use('/api/v2/auth', authRoutes);

module.exports = app;
```

Також відразу необхідно реалізувати підтвердження за email

```
const confirmEmail = async (req, res) : Promise<...> => {  Show usages
  try {
    const {token} = req.params;
    const decode = jwt.verify(token, JWT_CONFIRM_SECRET);

    const user = await UserModel.findById(decode.id);
    if (!user) return res.status(404).json({ message: 'Користувача не знайдено' });

    if(user.isConfirmed) return res.status(400).json({ message: 'Користувач уже підтверджений' });

    user.isConfirmed = true;
    await user.save();

    return res.status(200).json({ message: 'Електронну пошту підтверджено успішно!' });
  } catch (error) {
    return res.status(400).json({ message: 'Недійсний або прострочений токен' });
  }
}

module.exports = {register, confirmEmail};
```

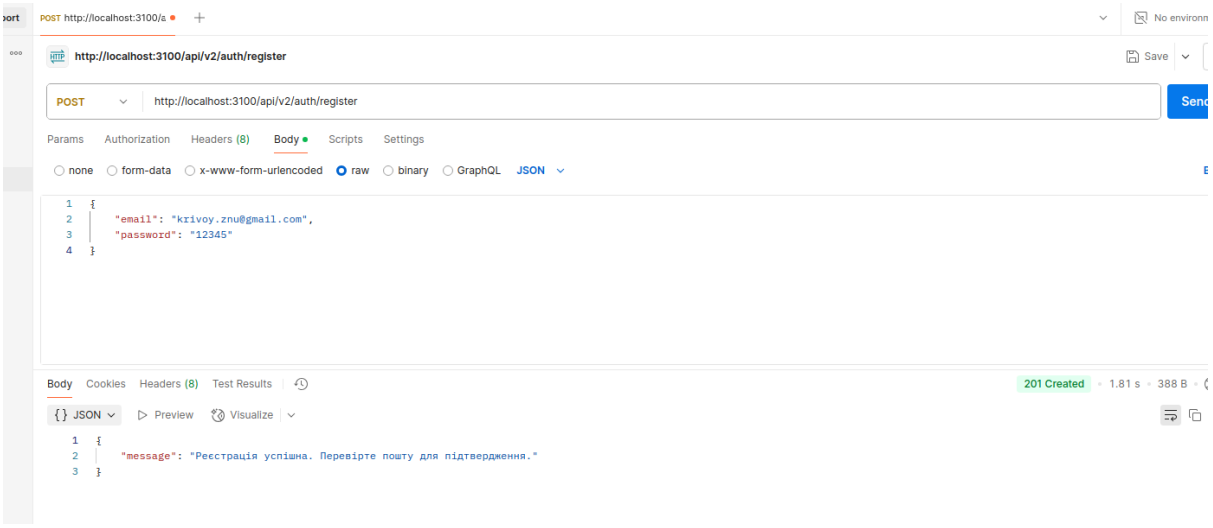
Оновлюємо файл роутів

task-list.component.ts | task.controller.js | auth.controller.js | auth.routes.js | app.js | task.ro

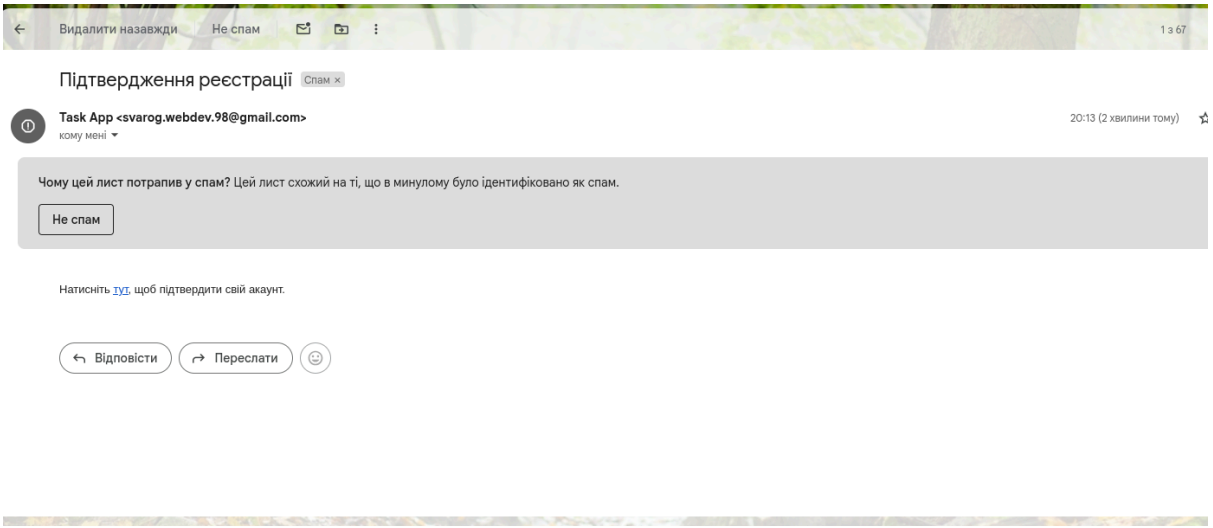
```
const express = require('express');
const router = express.Router();

const authController = require('../controllers/auth.controller');

router.post( path: '/register', authController.register);
router.post( path: '/confirm/:token', authController.confirmEmail);

module.exports = router;
```

Перевіримо реєстрацію, для цього використаємо Postman (поки у нас немає фронт-частини для реєстрації)

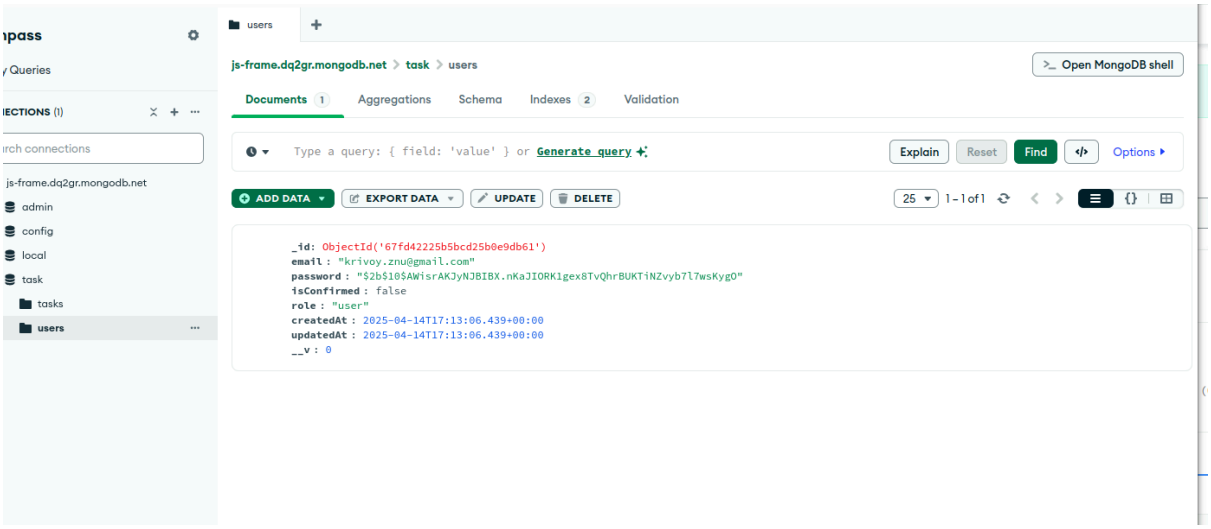
Реєструємося



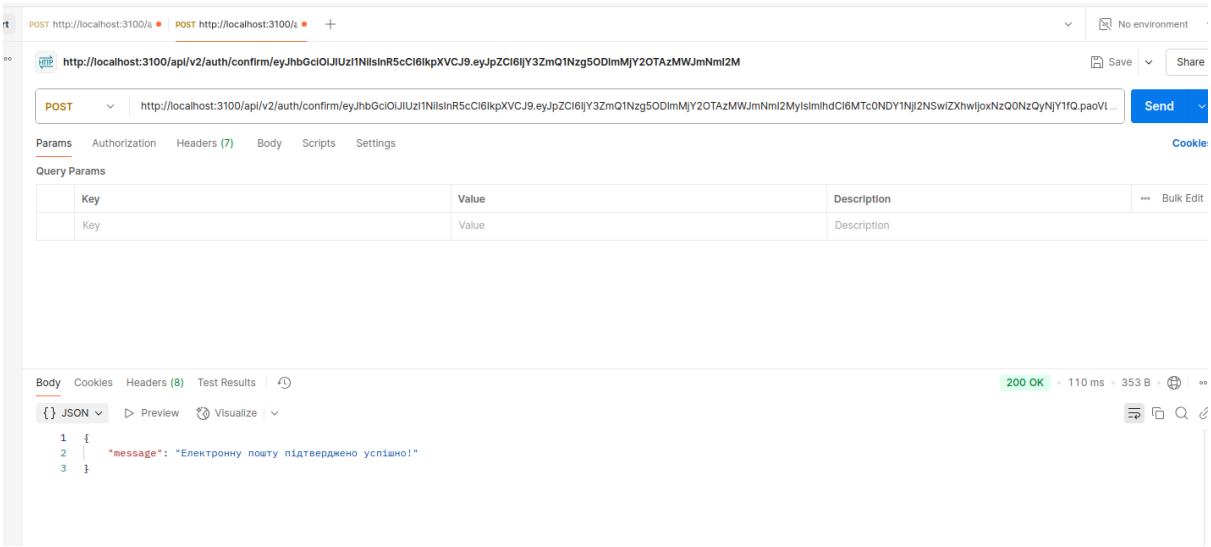
Перевіримо пошту (повідомлення скоріше за все буде у СПАМ)



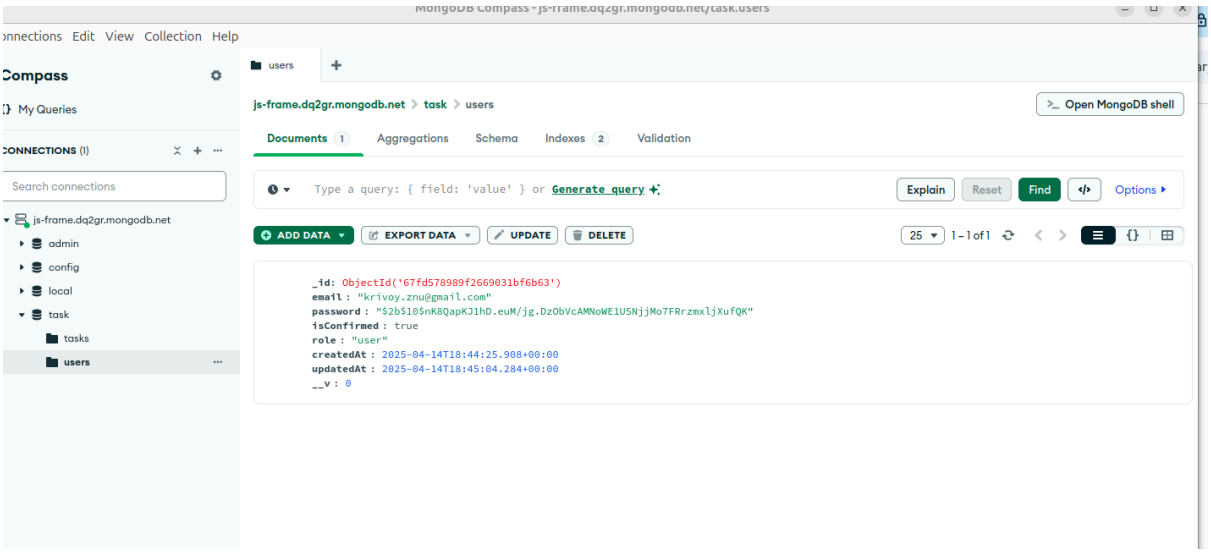
А також перевіримо БД



Чудово, реєстрація успішна. Тепер перевіримо підтвердження email. Для цього з посилання у повідомленні скопіюємо токен та зробимо запит



Перевіримо БД



Увага!!! Якщо надіслати повідомлення з підтвердженням, а потім перезапустити сервер, то токен буде невалідний.

Далі реалізуємо вхід до системи. Для цього підключаємо у контролер JWT_SECRET

```
const JWT_CONFIRM_SECRET :string = process.env.JWT_CONFIRM_SECRET;
const JWT_SECRET = process.env.JWT_SECRET;

const register = async (req, res) :Promise<...> => {  Show usages
```

І реалізуємо функцію входу

```
const login = async (req, res) : Promise<...> => {  Show usages
 try {
   const {email, password} = req.body;
   if (!email || !password) {
     return res.status(400).json({ message: 'Email і пароль обов'язкові' });
   }

   const user = await UserModel.findOne({ email });

   if (!user) return res.status(404).json({ message: 'Користувача не знайдено' });

   if(!user.isConfirmed) return res.status(400).json({ message: 'Будь ласка, підтвердіть електронну пошту' });

   const passwordMatch = await bcrypt.compare(password, user.password);

   if (!passwordMatch) return res.status(401).json({ message: 'Неправильний пароль' });

   const token = jwt.sign(
     {
       id: user._id,
       email: user.email,
       role: user.role
     },
     JWT_SECRET,
     { expiresIn: '2h' }
   );

   res.status(200).json({
     token,
     user: {
       id: user._id,
       email: user.email,
       role: user.role,
       isConfirmed: user.isConfirmed
     }
   });

 } catch (error) {
   res.status(500).send({message: 'Виникла помилка при вході до системи.', errors: error})
 }
}
```

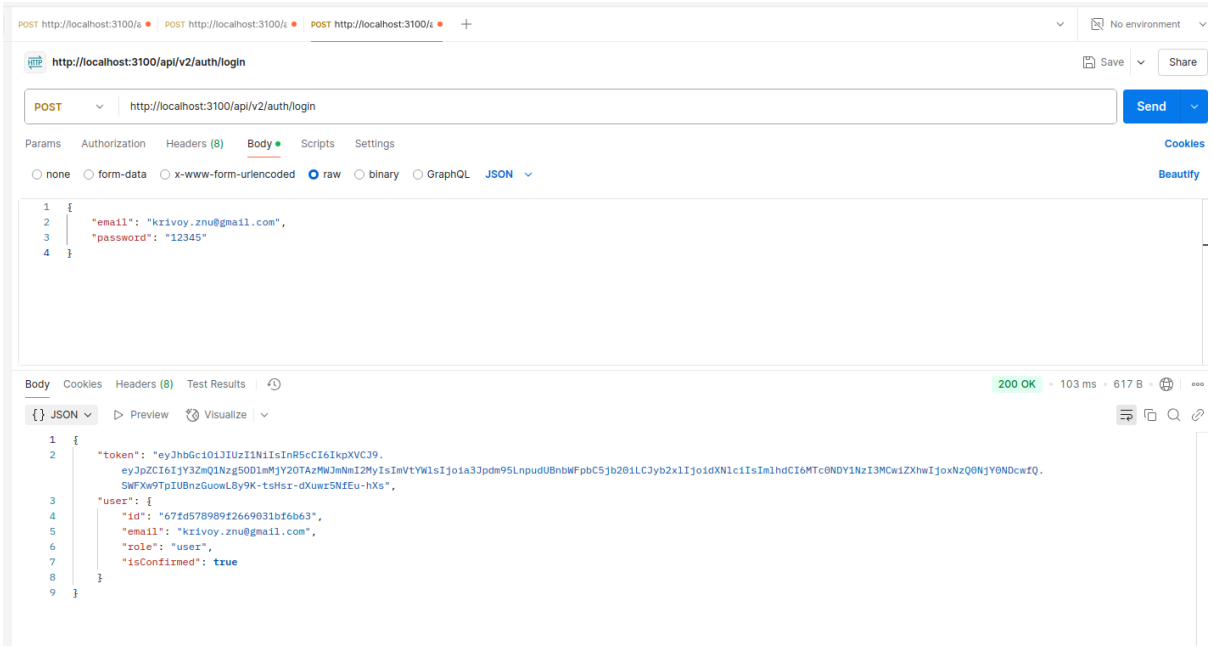
Не забуваємо експортувати функцію

```
module.exports = {register, confirmEmail, login};
```

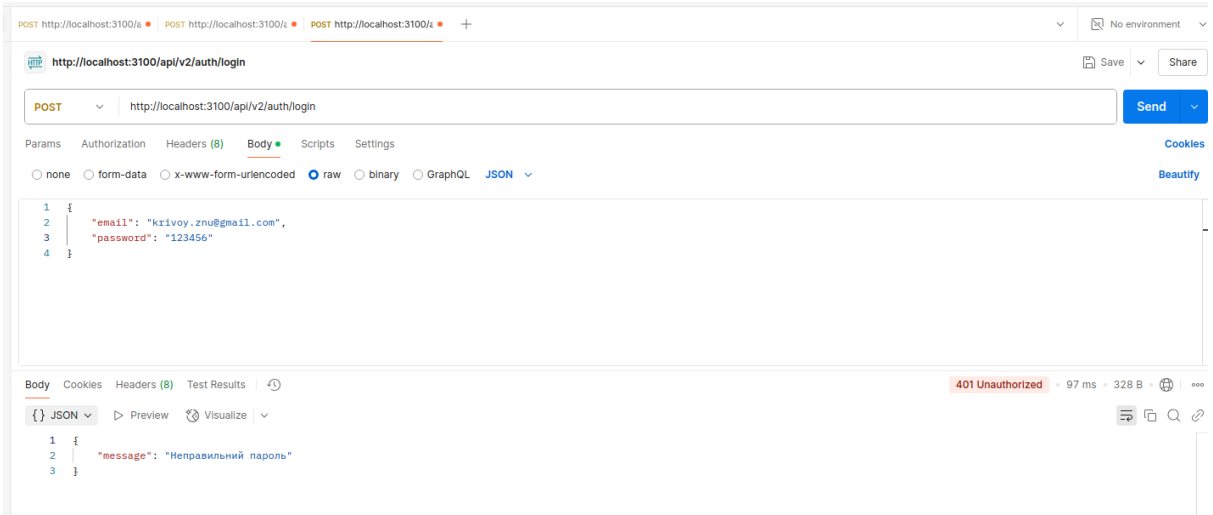
Оновлюємо файл роутів

```
const express : e | () => core.Express  = require('express');
const router : Router  = express.Router();

const authController : {...} | {...}  = require('../controllers/auth.controller');

router.post( path: '/register', authController.register);
router.post( path: '/confirm/:token', authController.confirmEmail);
router.post( path: '/login', authController.login);

module.exports = router;
```

Перевіримо вхід до системи



Якщо ввести хибний пароль



Переходимо до відновлення пароля. Спочатку нас необхідно реалізувати функцію надсилання запиту на відновлення пароля і прописати для нього функцію в email.service.js.

Для початку підключаємо JWT_RESET_SECRET

```
const JWT_CONFIRM_SECRET : string  = process.env.JWT_CONFIRM_SECRET;
const JWT_SECRET : string  = process.env.JWT_SECRET;
const JWT_RESET_SECRET = process.env.JWT_RESET_SECRET;
```

Далі реалізуємо функцію запиту на відновлення

```
97
98    const requestPasswordReset = async (req, res) : Promise<...>  => {  Show usages
99     try {
100      const { email } = req.body;
101
102      if (!email) return res.status(400).json({ message: 'Email обов'язковий' });
103
104      const user = await UserModel.findOne({ email });
105
106      if (!user) return res.status(404).json({ message: 'Користувача не знайдено' });
107
108      const token = jwt.sign({ id: user._id }, JWT_RESET_SECRET, { expiresIn: '30m' });
109
110      await sendResetPasswordEmail(email, token);
111
112      res.status(200).json({ message: 'Інструкції надіслано на пошту' });
113    } catch (error) {
114      res.status(500).send({message: 'Виникла помилка при запиті на відновлення пароля.', errors: error});
115    }
116   }
```

І функцію у email-сервісі

```
exports.sendResetPasswordEmail = async (email, token) : Promise<void>  => {
  const url : string  = `${process.env.FRONTEND_URL}/${token}`;
  try {
    await transporter.sendMail({
      from: `"Task App" <${process.env.SMTP_EMAIL}>`,
      to: email,
      subject: 'Відновлення пароля',
      html: `<p>Натисніть <a href="${url}">тут</a>, щоб задати новий пароль.</p>`
    });
    console.log('Email sent successfully');
  } catch (error) {
    console.error('Error sending email:', error);
  }
}
```

URL поки залишаємо базовий, як реалізуємо логіку зі сторони Angular, то пропишемо відповідні до навігації шляхи.

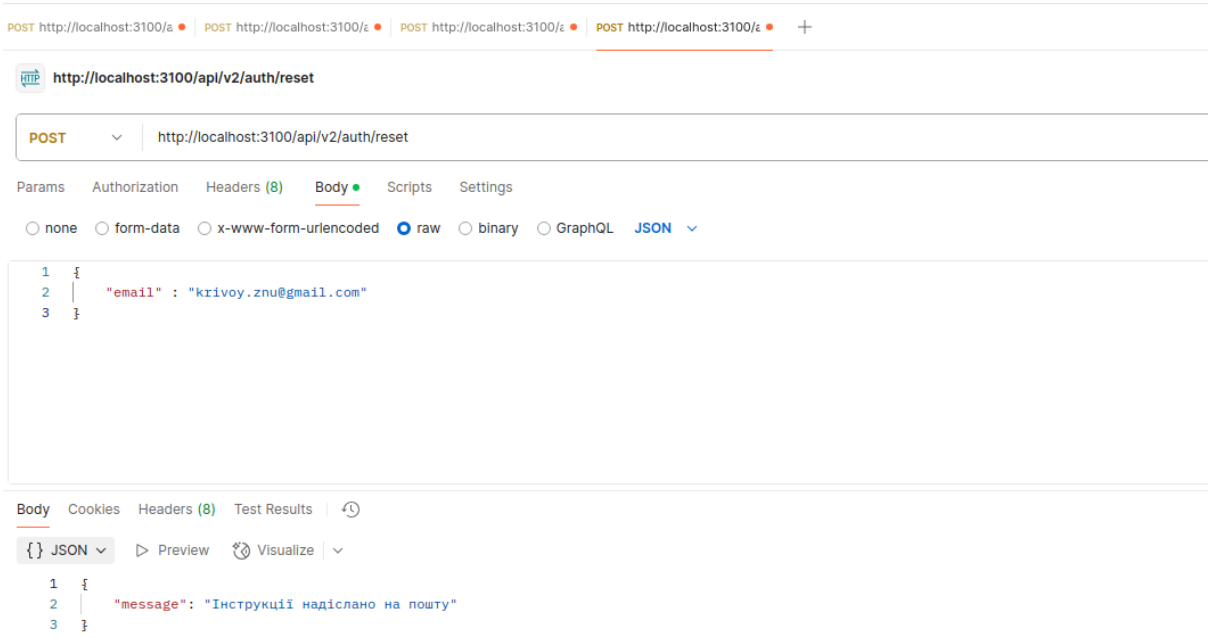
Далі реалізуємо функцію для скидання пароля.

```
const resetPassword = async (req, res) : Promise<...> => {  Show usages
  try {
    const { token } = req.params;
    const { password } = req.body;

    if (!password) return res.status(400).json({ message: 'Пароль обов'язковий' });
    const decoded = jwt.verify(token, JWT_RESET_SECRET);
    const user = await UserModel.findById(decoded.id);
    if (!user) return res.status(404).json({ message: 'Користувача не знайдено' });

    const passwordHash = await bcrypt.hash(password, salt: 10);
    user.password = passwordHash;
    await user.save();

    return res.status(200).json({ message: 'Пароль успішно змінено' });
  } catch (err) {
    return res.status(400).json({ message: 'Недійсний або прострочений токен' });
  }
};
```

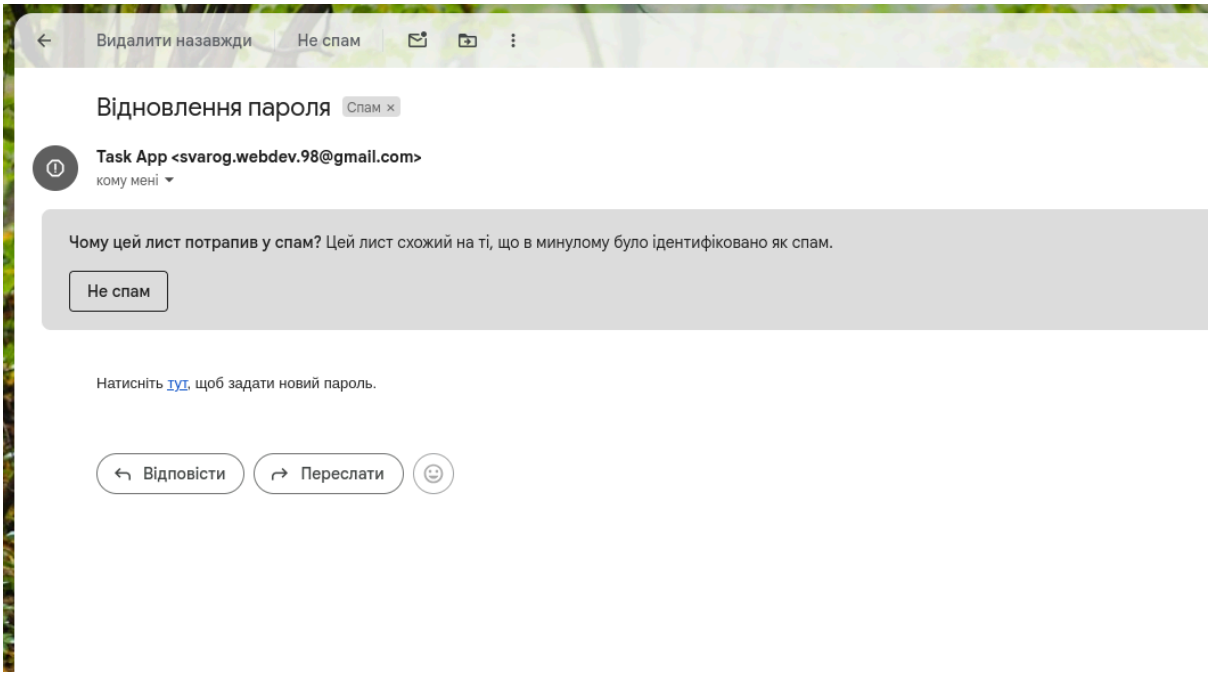
Доповнюємо наш файл роутів

```
const express : e | () => core.Express  = require('express');
const router : Router  = express.Router();

const authController : {...} | {...}  = require('../controllers/auth.controller');

router.post( path: '/register', authController.register);
router.post( path: '/confirm/:token', authController.confirmEmail);
router.post( path: '/login', authController.login);
router.post( path: '/reset', authController.requestPasswordReset);
router.post( path: '/reset-password/:token', authController.resetPassword);

module.exports = router;
```

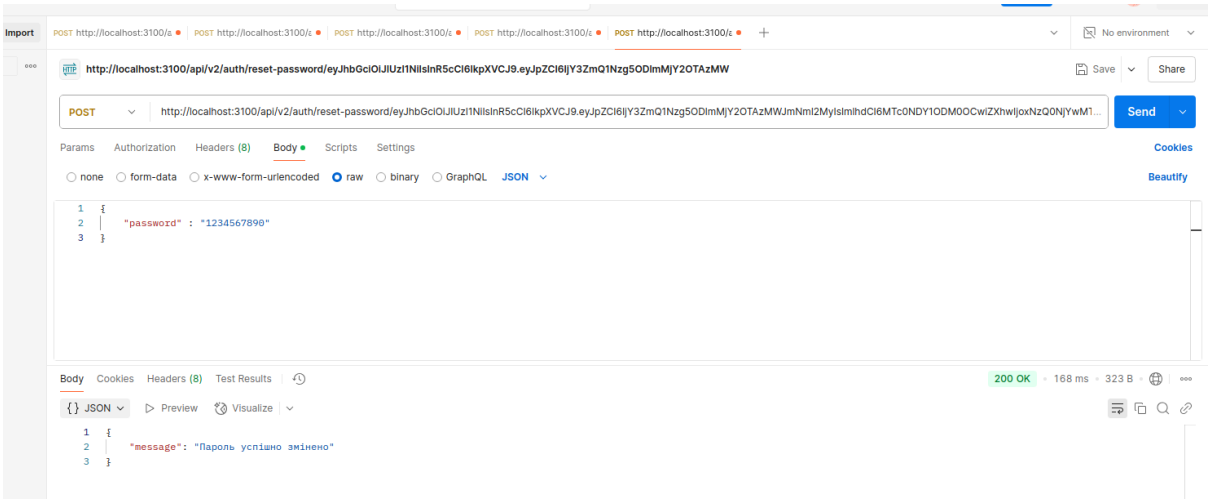
Перевіримо роботу



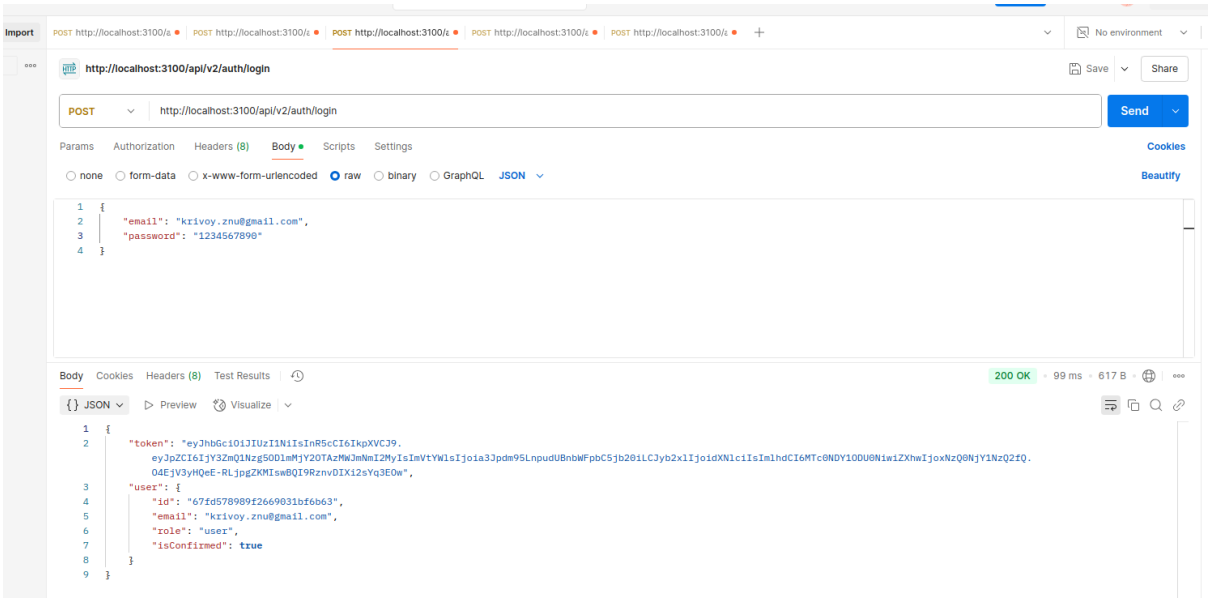
На пошті отримали повідомлення



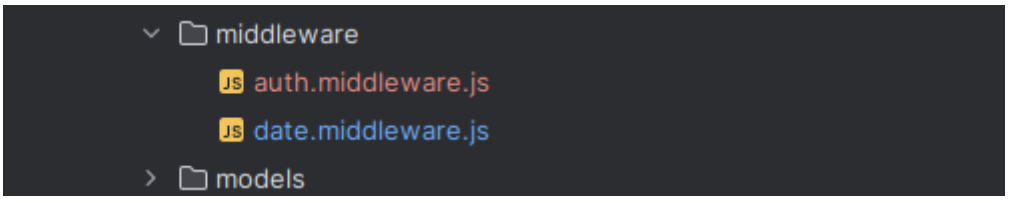
Візьмемо з нього токен та спробуємо відновити пароль



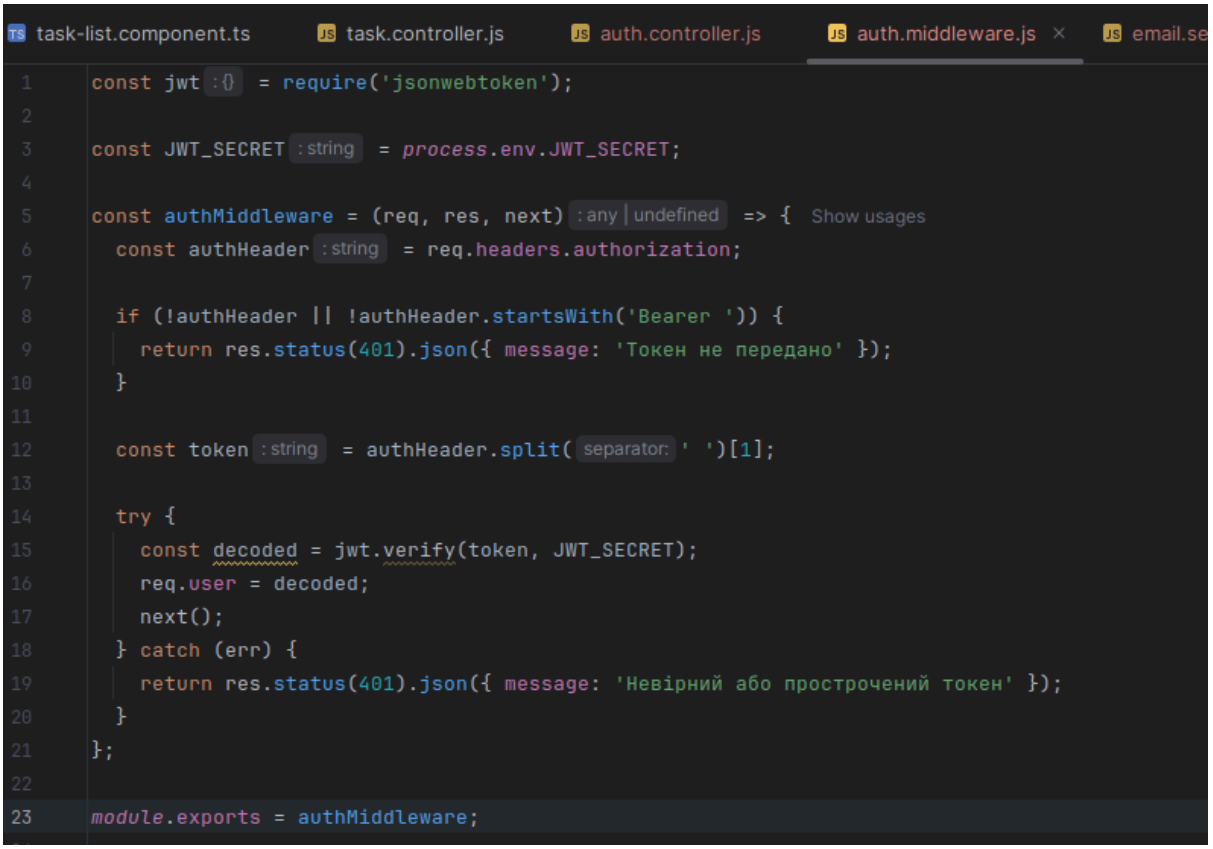
Перевіримо вхід з новим паролем



Переходимо до захисту маршрутів. Створимо auth.middleware.js



Пропишемо стандартний код такого middleware

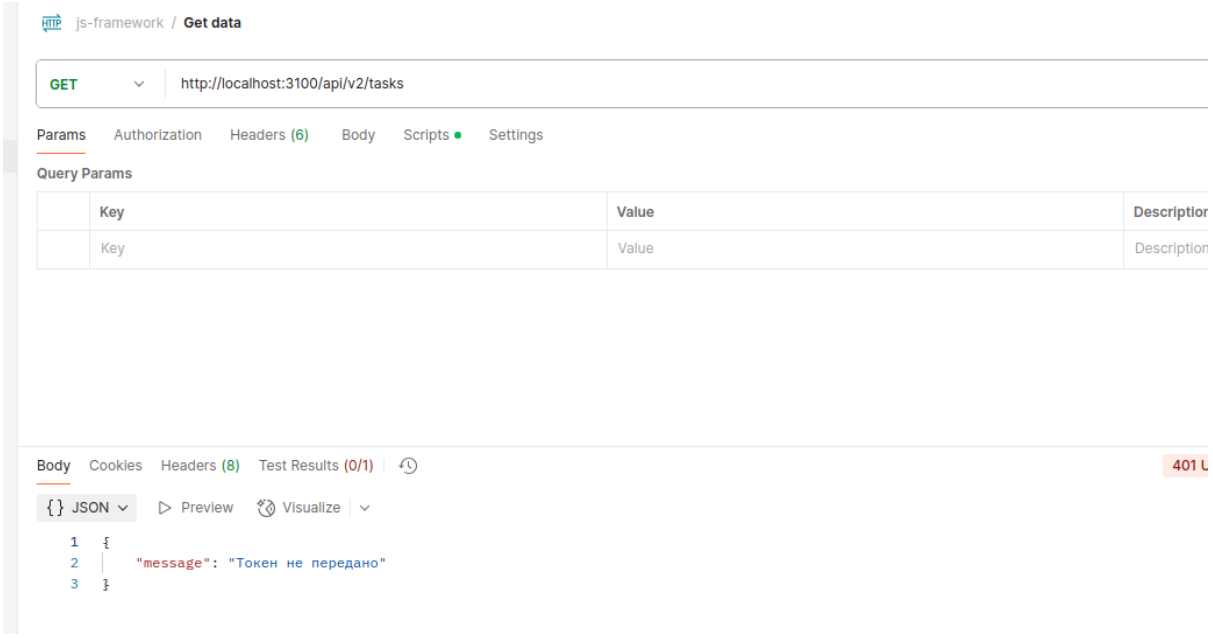


Цей middleware отримує токен з headers та перевіряє його наявність, якщо токен є, то перевіряє його, отримуючи з нього payload, який ми записуємо у req.user для того, щоб можна було використовувати у контролерах, які захищені цим middleware. Якщо токен відсутній або прострочений, то відхиляємо запит.

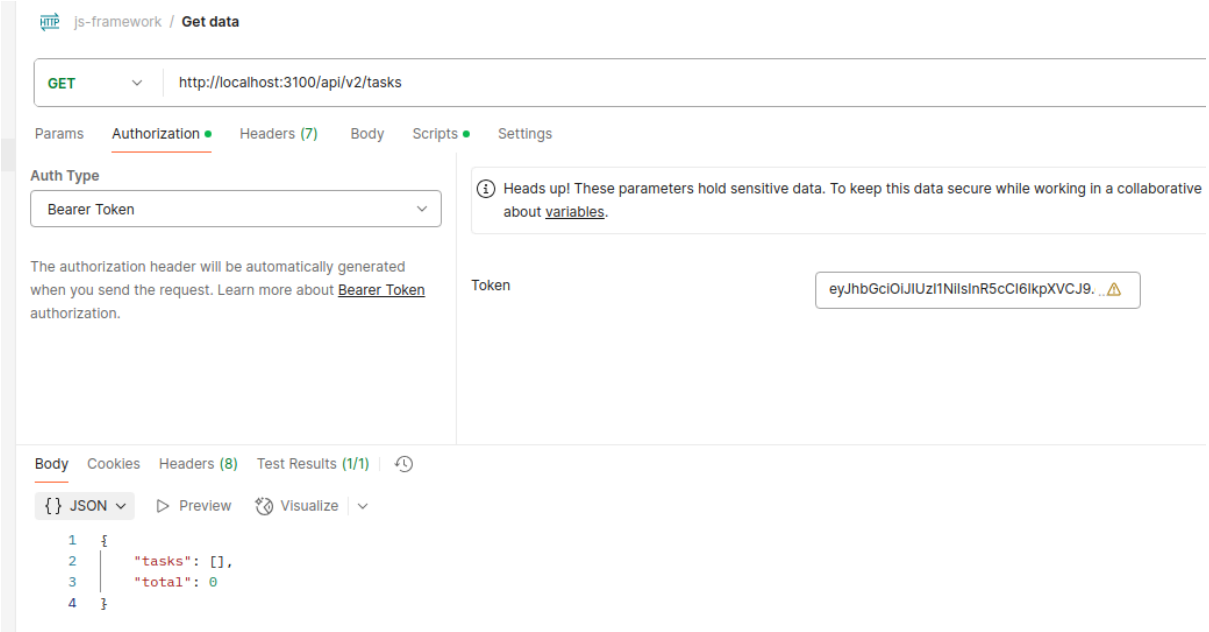
Захищаємо всі шляхи tasks за допомогою router.use

```
task.controller.js  auth.controller.js  auth.middleware.js  email.service.js  auth.routes.js  app.js  task.route

const router : Router = express.Router();

const taskController : {...} | {...} = require('../controllers/task.controller');

const {
  validateTask : [ValidationChain, ValidationCh... ,
  validateTaskPatch : [ValidationChain, function(any... ,
  handleValidationErrors
} = require('../validators/taskValidator');

const authMiddleware : function(any, any, any): (any ... | {...} = require("../middleware/auth.middleware");

router.use(authMiddleware);


router.get( path: '/', taskController.getTasks);
router.post( path: '/', validateTask, handleValidationErrors, taskController.createTask);
router.put( path: '/:id', validateTask, handleValidationErrors, taskController.updateTask);
router.patch( path: '/:id', validateTaskPatch, handleValidationErrors, taskController.patchTask);
router.delete( path: '/:id', taskController.deleteTask);

module.exports = router;
```

Перевіримо тепер отримання завдань за допомогою Postman



При вході, а потім передачі токену отримуємо



Краще не використовувати токен таким чином, а записувати його у змінну.

Переходимо до правил валідації. Нам необхідно створити 3 окремі правила: для входу/реєстрації, відновлення пароля, запит на відновлення пароля.

Реалізуємо **auth.validator.js**

middleware.js | email.service.js | auth.routes.js | taskMockController.js | app.js | auth.validator.js | task.model.js

```
const { check } = require('express-validator');

const ALLOWED_FIELDS :string[] = ['email', 'password'];

const validateAuthLogAndReg :(...)[] = [
  check( fields: 'email').trim().notEmpty().withMessage('Поле email є обов'язковим')
    .isEmail().withMessage('Неправильний email'),
  check( fields: 'password').notEmpty().withMessage('Поле пароля є обов'язковим')
    .isLength( options: { min: 8 }).withMessage('Пароль повинен містити щонайменше 8 символів'),
  (req, res, next) :any | undefined => {
    const extraFields :string[] = Object.keys(req.body).filter(field :string => !ALLOWED_FIELDS.includes(field));
    if (extraFields.length) {
      return res.status(400).json({ error: `Зайві поля: ${extraFields.join(', ')}` });
    }
    next();
  }
];
const validateAuthReqPassReset :(...)[] = [
  check( fields: 'email').trim().notEmpty().withMessage('Поле email є обов'язковим')
    .isEmail().withMessage('Неправильний email'),
  (req, res, next) :any | undefined => {
    const extraFields :string[] = Object.keys(req.body).filter(field :string => !ALLOWED_FIELDS.includes(field));
    if (extraFields.length) {
      return res.status(400).json({ error: `Зайві поля: ${extraFields.join(', ')}` });
    }
    next();
  }
];
const validateAuthPassReset :(...)[] = [
  check( fields: 'password').notEmpty().withMessage('Поле пароля є обов'язковим')
    .isLength( options: { min: 8 }).withMessage('Пароль повинен містити щонайменше 8 символів'),
  (req, res, next) :any | undefined => {
    const extraFields :string[] = Object.keys(req.body).filter(field :string => !ALLOWED_FIELDS.includes(field));
    if (extraFields.length) {
      return res.status(400).json({ error: `Зайві поля: ${extraFields.join(', ')}` });
    }
    next();
  }
];
module.exports = {validateAuthLogAndReg, validateAuthPassReset, validateAuthReqPassReset}
```

Та відкорегуємо під нього модель (додаємо minlength)

```
const mongoose :... = require('mongoose');
const UserRoles :{} = require("../constants/userRoles");

const UserSchema :Schema<any, Model<...>, {...}, {...}, {...}, {...}, {...}> = new mongoose.Schema( definition: {
  email: { type: String, trim: true, required: true, unique: true, lowercase: true, },
  password: { type: String, required: true, minlength: 8 },
  isConfirmed: { type: Boolean, default: false },
  role: { type: String, enum: Object.values(UserRoles), default: UserRoles.USER },
}, options: { timestamps: true });

const UserModel :Model<...> = mongoose.model( name: "User", UserSchema);
module.exports = UserModel;
```

Також відкорегуємо **task.validator.js**. Прибираємо з нього handleValidationErrors. Переносимо його до **validationErrorHandler.middleware.js**. Робимо це для цього, щоб можна було використовувати у різних файлах роутів.

taskMockRoutes.js | auth.validator.js | task.model.js | user.model.js | task.validator.js | validationErrorHandler.middleware.js

```
const {validationResult :ResultFactory<...> & {...}} = require("express-validator");

exports.handleValidationErrors = (req, res, next) :any | undefined => {
  const errors :Result<...> = validationResult(req);
  if (!errors.isEmpty()) {
    const valErrors :any[] = Object.values(errors.errors).map(err => err.msg);
    return res.status(400).json({ message: "Validation error", errors: valErrors });
  }
  next();
};
```

У файлі роутів tasks буде мати наступний вигляд

auth.validator.js | task.model.js | user.model.js | task.validator.js | validationErrorHandler.middleware.js | task.routes.js

```
const express :e | () => core.Express = require('express');
const router :Router = express.Router();

const taskController :{...} | {...} = require('../controllers/task.controller');

const {
  validateTask :[ValidationChain, ValidationCh...],
  validateTaskPatch :[ValidationChain, function(any...],
} = require('../validators/task.validator');

const authMiddleware :function(any, any, any): (any ... | {...} = require("../middleware/auth.middleware");

const {handleValidationErrors} =
  require("../middleware/validationErrorHandler.middleware");

router.use(authMiddleware);


router.get( path: '/', taskController.getTasks);
router.post( path: '/', validateTask, handleValidationErrors, taskController.createTask);
router.put( path: '/:id', validateTask, handleValidationErrors, taskController.updateTask);
router.patch( path: '/:id', validateTaskPatch, handleValidationErrors, taskController.patchTask);
router.delete( path: '/:id', taskController.deleteTask);

module.exports = router;
```

Підключаємо валідатори до шляхів auth

auth.routes.js | app.js | taskMockRoutes.js | auth.validator.js | task.model.js | user.model.js | task.validator.js

```
const express : e | () => core.Express = require('express');
const router : Router = express.Router();

const authController : {...} | {...} = require('../controllers/auth.controller');

const {
  validateAuthLogAndReg : [ValidationChain, ValidationCh... ,
  validateAuthReqPassReset : [ValidationChain, function(any... ,
  validateAuthPassReset : [ValidationChain, function(any...
} = require('../validators/auth.validator');

const {handleValidationErrors} =
  require("../middleware/validationErrorHandler.middleware");

router.post( path: '/register', validateAuthLogAndReg, handleValidationErrors, authController.register);
router.post( path: '/confirm/:token', authController.confirmEmail);
router.post( path: '/login', validateAuthLogAndReg, handleValidationErrors, authController.login);
router.post( path: '/reset', validateAuthReqPassReset, handleValidationErrors, authController.requestPasswordReset);
router.post( path: '/reset-password/:token', validateAuthPassReset, handleValidationErrors, authController.resetPassword);

module.exports = router;
```

2) Переходимо до фронт-частини. Вирішимо наступні задачі:
   a) Створимо компоненти для реєстрації, входу, відновлення пароля, запиту на відновлення пароля і підтвердження реєстрації;
   b) Створити розділ у Store;
   c) Захистити маршрути від неавторизованих користувачів;
   d) Створити сервіс для авторизації та Interceptor для додавання токену до запитів

Для початку створимо новий сервіс auth.service.ts

```
svarog@svarog:~/projects/angular-course$ ng g s services/auth --skip-tests
CREATE src/app/services/auth.service.ts (133 bytes)
svarog@svarog:~/projects/angular-course$
```

Інжектимо необхідні залежності

```
import {Inject, Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {AppConfig, CONFIG_TOKEN} from '../share/config/config';

@Injectable({  no usages
  providedIn: 'root'
})
export class AuthService {

  constructor(private http: HttpClient,  no usages
              @Inject(CONFIG_TOKEN) private config: AppConfig
  ) { }


}
```

Також необхідно створити інтерфейси для отримання даних авторизованого користувача у **auth.model.ts**.

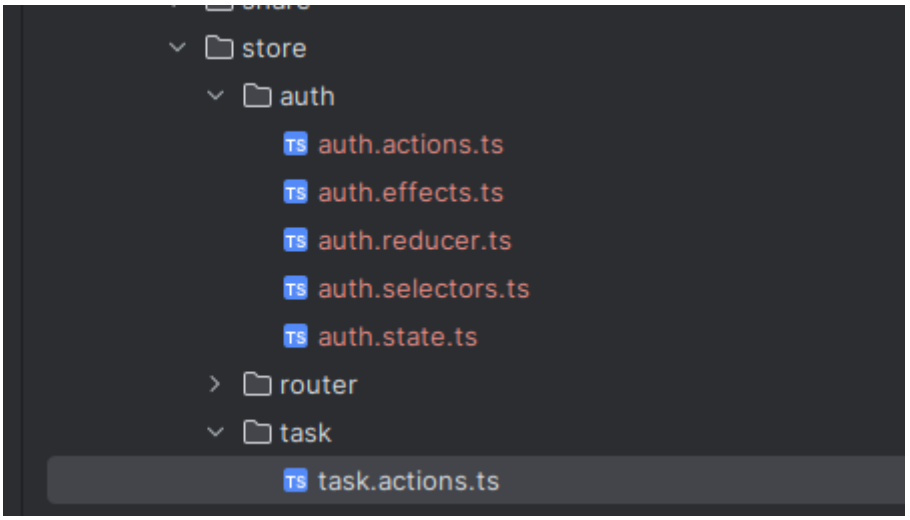
```
export interface AuthUser {  Show usages
  id: string;
  email: string;
  role: string;
  isConfirmed: boolean;
}

export interface LoginResponse {  no usages
  token: string;
  user: AuthUser;
}
```

Реалізуємо методи

```
login(email: string, password: string): Observable<LoginResponse> {
        return this.http.post<LoginResponse>(`${this.config.apiUrl}/v2/auth/login`, { email, password }).pipe(
        tap(({ token }) => this.saveToken(token))
        );
}

register(email: string, password: string): Observable<void> {
        return this.http.post<void>(`${this.config.apiUrl}/v2/auth/register`, { email, password });
}

requestPasswordReset(email: string): Observable<void> {
        return this.http.post<void>(`${this.config.apiUrl}/v2/auth/reset`, { email });
}
```

```
  confirmEmail(token: string): Observable<void> {
        return this.http.post<void>(`${this.config.apiUrl}/v2/auth/confirm/${token}`, {});
  }

  resetPassword(token: string, password: string): Observable<void> {
        return this.http.post<void>(`${this.config.apiUrl}/v2/auth/reset-password/${token}`, { password });
  }

  logout(): void {
        localStorage.removeItem('token');
  }

  private saveToken(token: string): void {
        localStorage.setItem('token', token);
  }

  getToken(): string | null {
        return localStorage.getItem('token');
  }

  isLoggedIn(): boolean {
        return !!this.getToken();
  }
```

Далі переходимо до Store. Створимо нову теку store/auth та створимо відповідні файли (state, actions тощо).



Почнемо з auth.state.ts

```ts
import {createEntityAdapter, EntityAdapter, EntityState} from '@ngrx/entity';
import {AuthUser} from '../../core/models/auth.model';

export interface AuthState extends EntityState<AuthUser>{ Show usages
  currentUserId: string | null;
  token: string | null;
  loading: boolean;
  error: string | null;
  registered: boolean;
  passwordResetEmailSent: boolean;
  confirmEmailSuccess: boolean;
  resetPasswordSuccess: boolean;
}

export const authAdapter: EntityAdapter<AuthUser> = createEntityAdapter<AuthUser>( Show usages
  {
    selectId: model :AuthUser => model.id
  }
)

export const initialAuthEntityState: AuthState = authAdapter.getInitialState({ Show usages
  currentUserId: null,
  token: null,
  loading: false,
  error: null,
  registered: false,
  passwordResetEmailSent: false,
  confirmEmailSuccess: false,
  resetPasswordSuccess: false,
});
```

Додаємо наш state до app.state.ts

```ts
import {TaskState} from './task/task.state';
import {RouterState} from './router/router.state';
import {RouterReducerState} from '@ngrx/router-store';
import {AuthState} from './auth/auth.state';


export interface AppState { Show usages  Kryvyi Yaroslav *
  tasks: TaskState,
  auth: AuthState,
  router: RouterReducerState<RouterState>
}
```

Далі переходимо до створення Actions

Реєстрація

```
export const register : ActionCreator<'[Auth] Register', (props: ... = createAction( no usages
  '[Auth] Register',
  props<{ email: string; password: string }>()
);

export const registerSuccess : ActionCreator<'[Auth] Register Success', ... = createAction( no usages
  '[Auth] Register Success',
);

export const registerFailure : ActionCreator<'[Auth] Register Failure', ... = createAction( no usages
  '[Auth] Register Failure',
  props<{ error: string }>()
)
```

Вхід

```
export const login : ActionCreator<'[Auth] Login', (props: { e... = createAction( no usages
  '[Auth] Login',
  props<{ email: string; password: string }>()
);

export const loginSuccess : ActionCreator<'[Auth] Login Success', (pr... = createAction( no usages
  '[Auth] Login Success',
  props<{ user: AuthUser, token: string }>()
);

export const loginFailure : ActionCreator<'[Auth] Login Failure', (pr... = createAction( no usages
  '[Auth] Login Failure',
  props<{ error: string }>()
);
```

Вихід

```
export const logout : ActionCreator<'[Auth] Logout', () => Acti... = createAction( no usages
  '[Auth] Logout',
);
```

Запит на відновлення пароля

```
export const requestResetPassword : ActionCreator<'[Auth] Request Reset Passw... = createAction(  no usages
  '[Auth] Request Reset Password',
  props<{ email: string }>()
);

export const requestResetPasswordSuccess : ActionCreator<'[Auth] Request Reset Passw... = createAction(  no usages
  '[Auth] Request Reset Password Success',
);

export const requestResetPasswordFailure : ActionCreator<'[Auth] Request Reset Passw... = createAction(  no usages
  '[Auth] Request Reset Password Failure',
  props<{ error: string }>()
);
```

Відновлення пароля

```
export const resetPassword : ActionCreator<'[Auth] Reset Password', (p... = createAction(  no usages
  '[Auth] Reset Password',
  props<{ token: string; password: string }>()
);

export const resetPasswordSuccess : ActionCreator<'[Auth] Reset Password Succ... = createAction(  no usages
  '[Auth] Reset Password Success'
);

export const resetPasswordFailure : ActionCreator<'[Auth] Reset Password Fail... = createAction(  no usages
  '[Auth] Reset Password Failure',
  props<{ error: string }>()
);
```

Підтвердження реєстрації

```
export const confirmEmail : ActionCreator<'[Auth] Confirm Email', (pr... = createAction(  no usages
  '[Auth] Confirm Email',
  props<{ token: string }>()
);

export const confirmEmailSuccess : ActionCreator<'[Auth] Confirm Email Succe... = createAction(  no usages
  '[Auth] Confirm Email Success'
);

export const confirmEmailFailure : ActionCreator<'[Auth] Confirm Email Failu... = createAction(  no usages
  '[Auth] Confirm Email Failure',
  props<{ error: string }>()
);
```

Скидання стану після дії

```
export const clearAuthState : ActionCreator<'[Auth] Clear Auth State', ... > = createAction(  no usages
  '[Auth] Clear Auth State'
);
```

Далі реалізуємо **auth.reducer.ts**

```
import {createReducer} from '@ngrx/store';
import {initialAuthEntityState} from './auth.state';

export const authReducer : ActionReducer<AuthState, Action<string>> = createReducer(  no usages
  initialAuthEntityState,

)
```

Відразу реєструємо його у **app.module.ts**

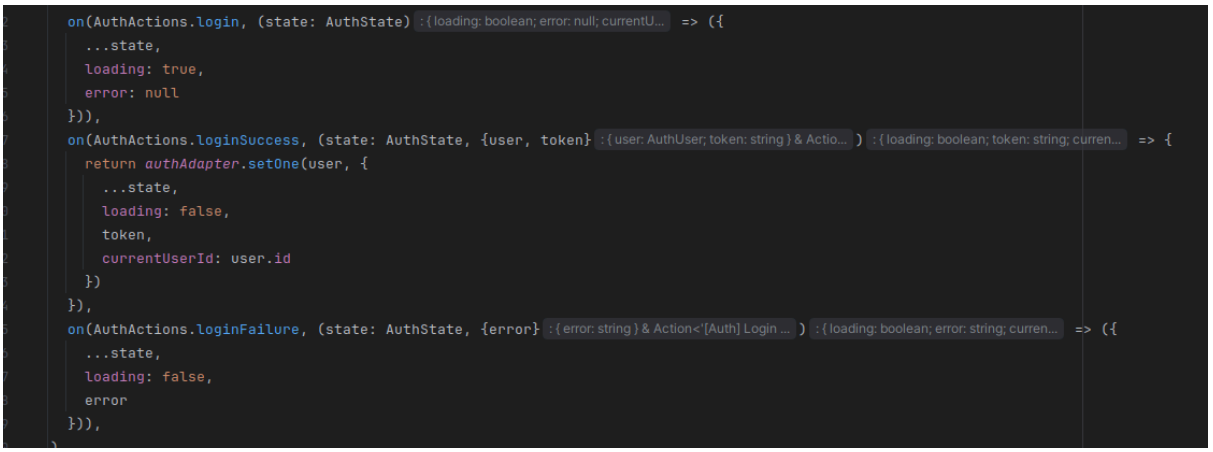
```
    MatChipsModule,
    StoreModule.forRoot<AppState>({
      tasks: taskReducer,
      auth: authReducer,
      router: routerReducer
    }),
    EffectsModule.forRoot([TaskEffects]),
```

Реалізуємо також дії у reducer

Реєстрація

```
export const authReducer : ActionReducer<AuthState, Action<string>> = createReducer(  Show usages
  initialAuthEntityState,
  on(AuthActions.register, (state: AuthState) : { loading: boolean; error: null; currentU... => ({
    ...state,
    loading: true,
    error: null
  })),
  on(AuthActions.registerSuccess, (state: AuthState) : { loading: boolean; registered: boolean; ... => ({
    ...state,
    loading: false,
    registered: true
  })),
  on(AuthActions.registerFailure, (state: AuthState, {error} : { error: string } & Action<'[Auth] Regist... ) : { loading: boolean; error: string; curren... => ({
    ...state,
    loading: false,
    error
  })),
```

Вхід



```
on(AuthActions.login, (state: AuthState) : { loading: boolean; error: null; currentU... => ({
  ...state,
  loading: true,
  error: null
})),
on(AuthActions.loginSuccess, (state: AuthState, {user, token} : { user: AuthUser; token: string } & Actio... ) : { loading: boolean; token: string; curren... => {
  return authAdapter.setOne(user, {
    ...state,
    loading: false,
    token,
    currentUserId: user.id
  })
}),
on(AuthActions.loginFailure, (state: AuthState, {error} : { error: string } & Action<'[Auth] Login ... ) : { loading: boolean; error: string; curren... => ({
  ...state,
  loading: false,
  error
})),
```

Вихід

```
on(AuthActions.logout, (state : AuthState ) : { currentUserId: null; token: null; loadi... =>
  authAdapter.removeAll({
    ...state,
    currentUserId: null,
    token: null,
    loading: false,
    error: null,
    registered: false,
    passwordResetEmailSent: false,
    confirmEmailSuccess: false,
    resetPasswordSuccess: false,
  })
),
```

Запит на відновлення пароля

```
on(AuthActions.requestResetPassword, (state : AuthState ) : { loading: boolean; currentUserId: string... => ({
  ...state,
  loading: true
})),
on(AuthActions.requestResetPasswordSuccess, (state : AuthState ) : { loading: boolean; passwordResetEmailSen... => ({
  ...state,
  loading: false,
  passwordResetEmailSent: true
})),
on(AuthActions.requestResetPasswordFailure, (state : AuthState , { error } : { error: string } & Action<'[Auth] Reques... ) : { loading: boolean; error
  ...state,
  loading: false,
  error
})),
```

Відновлення пароля

```
  on(AuthActions.resetPassword, state : AuthState  => ({
    ...state,
    loading: true,
    error: null
  })),
  on(AuthActions.resetPasswordSuccess, state : AuthState  => ({
    ...state,
    loading: false,
    resetPasswordSuccess: true
  })),
  on(AuthActions.resetPasswordFailure, (state : AuthState , { error } : { error: string } & Action<'[Auth] Reset ... ) : { loading: boolean; error: string; curren...  => ({
    ...state,
    loading: false,
    error
  })),
```

Підтвердження реєстрації

```
  on(AuthActions.confirmEmail, state : AuthState  => ({
    ...state,
    loading: true,
    error: null
  })),
  on(AuthActions.confirmEmailSuccess, state : AuthState  => ({
    ...state,
    loading: false,
    confirmEmailSuccess: true
  })),
  on(AuthActions.confirmEmailFailure, (state : AuthState , { error } : { error: string } & Action<'[Auth] Confir... ) : { loading: boolean; error: string; curren...  => ({
    ...state,
    loading: false,
    error
  })),
)
```

Скидання стану

```
  on(AuthActions.confirmEmailFailure, (state : AuthState , { error } : { error: string } & Action<'[Auth] Confir... ) : { loading: boolean; error: string; curren...  => ({
    ...state,
    loading: false,
    error
  })),
  on(AuthActions.clearAuthState, (state : AuthState ) : { error: null; registered: boolean; passw...  => ({
    ...state,
    error: null,
    registered: false,
    passwordResetEmailSent: false,
    confirmEmailSuccess: false,
    resetPasswordSuccess: false,
  }))
)
```

Переходимо до **auth.selectors.ts**. Додамо селектори для AuthUser

```ts
import {createFeatureSelector, createSelector} from '@ngrx/store';
import {authAdapter, AuthState} from './auth.state';

export const selectAuthState : MemoizedSelector<object, AuthState, Defau... = createFeatureSelector<AuthState>('auth'); Show usages

const {
  selectAll,
  selectEntities,
  selectIds,
  selectTotal
} = authAdapter.getSelectors(selectAuthState);

export const selectCurrentUserId : MemoizedSelector<object, string | null, (... = createSelector( Show usages
  selectAuthState,
  (state : AuthState ) : string | null => state.currentUserId
);

export const selectCurrentUser : MemoizedSelector<object, AuthUser | null ... = createSelector( no usages
  selectEntities,
  selectCurrentUserId,
  (entities : Dictionary<AuthUser> , currentUserId : string | null ) : AuthUser | null | undefined => currentUserId ? entities[currentUserId] : null
);
```

Додамо селектори для стану

```ts
export const selectAuthToken : MemoizedSelector<object, string | null, (... = createSelector( no usages
  selectAuthState,
  (state : AuthState ) : string | null => state.token
);

export const selectAuthLoading : MemoizedSelector<object, boolean, (s1: Au... = createSelector( no usages
  selectAuthState,
  (state : AuthState ) : boolean => state.loading
);

export const selectAuthError : MemoizedSelector<object, string | null, (... = createSelector( no usages
  selectAuthState,
  (state : AuthState ) : string | null => state.error
);

export const selectRegistered : MemoizedSelector<object, boolean, (s1: Au... = createSelector( no usages
  selectAuthState,
  (state : AuthState ) : boolean => state.registered
);

export const selectPasswordResetEmailSent : MemoizedSelector<object, boolean, (s1: Au... = createSelector( no usages
  selectAuthState,
  (state : AuthState ) : boolean => state.passwordResetEmailSent
);

export const selectConfirmEmailSuccess : MemoizedSelector<object, boolean, (s1: Au... = createSelector( no usages
  selectAuthState,
  state : AuthState => state.confirmEmailSuccess
);

export const selectResetPasswordSuccess : MemoizedSelector<object, boolean, (s1: Au... = createSelector( no usages
  selectAuthState,
  state : AuthState => state.resetPasswordSuccess
);
```

Селектор для перевірки токену

```
export const selectIsLoggedIn : MemoizedSelector<object, boolean, (s1: st... = createSelector(  no usages
  selectAuthToken,
  (token : string | null ) : boolean  => !!token
);
```

Переходимо до **auth.effects.ts**

Інжектимо залежності

```
import {inject, Injectable} from '@angular/core';
import {Actions} from '@ngrx/effects';
import {AuthService} from '../../services/auth.service';
import {Store} from '@ngrx/store';
import {Router} from '@angular/router';

@Injectable()  no usages
export class AuthEffects {
  private actions$ : Actions<any>  = inject(Actions);
  private authService : AuthService  = inject(AuthService);
  private store : Store<any>  = inject(Store);
  private router : Router  = inject(Router);


}
```

Далі прописуємо effects для входу та редіректу після нього (поки залишимо такий, потім замінимо)

```
login$ : Observable<({ user: AuthUser; token: stri... = createEffect(() : Observable<({ user: AuthUser; token: stri... =>
  this.actions$.pipe(
    ofType(AuthActions.login),
    exhaustMap(({email, password} : { email: string; password: string } & Act... ) : Observable<({ user: AuthUser; token: stri... =>
      this.authService.login(email, password).pipe(
        map(({ user, token } : LoginResponse ) : { user: AuthUser; token: string } & Actio... => AuthActions.loginSuccess({user, token })),
        catchError(err : any => of(AuthActions.loginFailure({error: formatError(err)})))
      ),
    )
  )
);

loginRedirect$ : Observable<{ user: AuthUser; token: strin... = createEffect(
  () : Observable<{ user: AuthUser; token: strin... =>
    this.actions$.pipe(
      ofType(AuthActions.loginSuccess),
      tap(() : Promise<boolean> => this.router.navigate([{ outlets: { primary: ['tasks'], aside: ['stats'] } }]))
    ),
  { dispatch: false }
);
```

Далі прописуємо для реєстрації

```
register$ : Observable<Action<'[Auth] Register Succes... = createEffect(() : Observable<Action<'[Auth] Register Succes... =>
  this.actions$.pipe(
    ofType(AuthActions.register),
    exhaustMap(({ email, password } : { email: string; password: string } & Act... ) : Observable<Action<'[Auth] Register Succes... =>
      this.authService.register(email, password).pipe(
        map(() : Action<'[Auth] Register Success'> => AuthActions.registerSuccess()),
        catchError(err : any =>
          of(AuthActions.registerFailure({error: formatError(err)}))
        )
      )
    )
  )
);
```

Запит на відновлення пароля

```
requestPasswordReset$ : Observable<Action<'[Auth] Request Reset P... = createEffect(() : Observable<Action<'[Auth] Request Reset P... =>
  this.actions$.pipe(
    ofType(AuthActions.requestResetPassword),
    exhaustMap(({ email } : { email: string } & Action<'[Auth] Reques... ) : Observable<Action<'[Auth] Request Reset P... =>
      this.authService.requestPasswordReset(email).pipe(
        map(() : Action<'[Auth] Request Reset Password... => AuthActions.requestResetPasswordSuccess()),
        catchError(err : any =>
          of(AuthActions.requestResetPasswordFailure({error: formatError(err)}))
        )
      )
    )
  )
);
```

Відновлення пароля

```
resetPassword$ : Observable<Action<'[Auth] Reset Password ... = createEffect(() : Observable<Action<'[Auth] Reset Password ... =>
  this.actions$.pipe(
    ofType(AuthActions.resetPassword),
    exhaustMap(({ token, password } : { token: string; password: string } & Act... ) : Observable<Action<'[Auth] Reset Password ... =>
      this.authService.resetPassword(token, password).pipe(
        map(() : Action<'[Auth] Reset Password Success... => AuthActions.resetPasswordSuccess()),
        catchError(err : any =>
          of(AuthActions.resetPasswordFailure({error: formatError(err)}))
        )
      )
    )
  )
);
```

Підтвердження реєстрації

```
  );

  confirmEmail$ : Observable<Action<'[Auth] Confirm Email S... = createEffect(() : Observable<Action<'[Auth] Confirm Email S... =>
    this.actions$.pipe(
      ofType(AuthActions.confirmEmail),
      exhaustMap(({ token } : { token: string } & Action<'[Auth] Confir... ) : Observable<Action<'[Auth] Confirm Email S... =>
        this.authService.confirmEmail(token).pipe(
          map(() : Action<'[Auth] Confirm Email Success'> => AuthActions.confirmEmailSuccess()),
          catchError(err : any =>
            of(AuthActions.confirmEmailFailure({error: formatError(err)}))
          )
        )
      )
    )
  );
```

Вихід

```

  logout$ : Observable<Action<'[Auth] Logout'>> & Cre... = createEffect(() : Observable<Action<'[Auth] Logout'>> =>
      this.actions$.pipe(
        ofType(AuthActions.logout),
        tap(() : void => {
          this.authService.logout();
          this.router.navigate(['/auth/login']);
        })
      ),
    { dispatch: false }
  );
```

Реєструємо effect

```
  }),
  EffectsModule.forRoot([TaskEffects, AuthEffects]),
  StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: !isDevMode() }),
  StoreRouterConnectingModule.forRoot({
    stateKey: 'router'
  }),
```

Далі переходимо до створення компонентів, почнемо з компоненту входу до системи

```
svarog@svarog:~/projects/angular-course$ ng g s services/auth --skip-tests
CREATE src/app/services/auth.service.ts (133 bytes)
svarog@svarog:~/projects/angular-course$ ng g c components/auth/login --skip-tests
CREATE src/app/components/auth/login/login.component.scss (0 bytes)
CREATE src/app/components/auth/login/login.component.html (20 bytes)
CREATE src/app/components/auth/login/login.component.ts (217 bytes)
UPDATE src/app/app.module.ts (3444 bytes)
svarog@svarog:~/projects/angular-course$
```

Реалізуємо login.component.ts

```
export class LoginComponent implements OnInit, OnDestroy {
  loginForm!: FormGroup;
  error$!: Observable<string | null>;
  loading$!: Observable<boolean>;

  private destroy$ = new Subject<void>();

  constructor(
            private fb: FormBuilder,
            private store: Store<AppState>,
            private snackBar: MatSnackBar,
  ) {}

  ngOnInit(): void {
            this.error$ = this.store.select(AuthSelectors.selectAuthError);
            this.loading$ = this.store.select(AuthSelectors.selectAuthLoading);

            this.error$.pipe(takeUntil(this.destroy$)).subscribe(error => {
            if (error) {
            this.snackBar.open(error, 'Закрити', { duration: 10000, panelClass: ['error-snackbar'] });
            }
            });

            this.loginForm = this.fb.group({
            email: ['', [Validators.required, Validators.email]],
            password: ['', [Validators.required, Validators.minLength(8)]],
            });
  }

  onSubmit(): void {
            if (this.loginForm.invalid) return;

            const { email, password } = this.loginForm.value;
            this.store.dispatch(AuthActions.login({ email, password }));
  }

  ngOnDestroy(): void {
            this.destroy$.next();
            this.destroy$.complete();
  }
}
```

Реалізуємо шаблон форми

```
<div class="login-container">
  <mat-card class="login-card">
            <mat-card-title>Вхід у систему</mat-card-title>

            <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
            <mat-form-field appearance="outline">
            <mat-label>Email</mat-label>
            <input matInput formControlName="email" type="email" />
            @if(loginForm.get('email')?.hasError('required')){
```

```
            <mat-error>
            Email обов’язковий
            </mat-error>
            }
            @if(loginForm.get('email')?.hasError('email') && loginForm.get('email')?.dirty){
            <mat-error>
            Некоректний email
            </mat-error>
            }
            </mat-form-field>

            <mat-form-field appearance="outline">
            <mat-label>Пароль</mat-label>
            <input matInput formControlName="password" type="password" />
            @if(loginForm.get('password')?.hasError('required') && loginForm.get('password')?.dirty){
            <mat-error>
            Пароль обов’язковий
            </mat-error>
            }
            @if(loginForm.get('password')?.hasError('minlength')){
            <mat-error>
            Пароль повинен містити мінімум {{loginForm.get('password')?.getError('minlength').requiredLength}}
символів
            </mat-error>
            }
            </mat-form-field>

            <button mat-raised-button color="primary" type="submit" [disabled]="loginForm.invalid || (loading$ | async)">
            Увійти
            </button>
            </form>
  </mat-card>
</div>
```

Та стилі

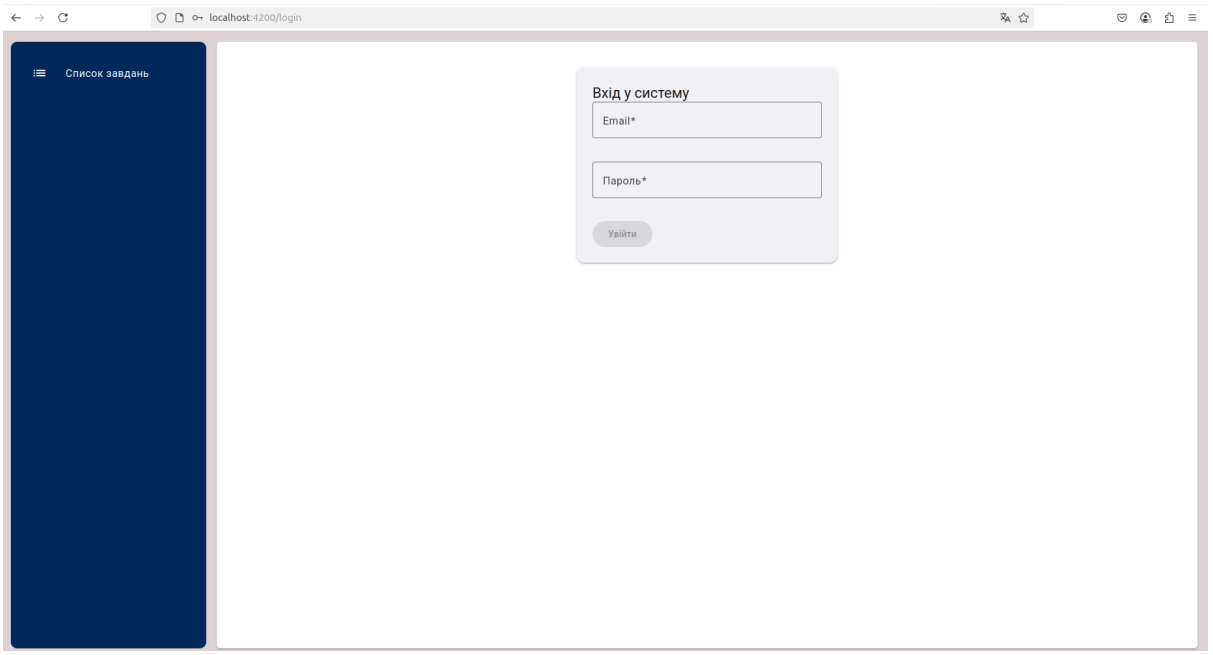
```
:host {
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}

.login-container {
  display: flex;
  justify-content: center;
  width: 40%;
}

.login-card {
  flex-grow: 1;
  gap: 40px;
  padding: 24px;
}

mat-form-field {
  display: block;
```

```
  width: 100%;
  margin-bottom: 16px;
}
```

Підключимо до роутів

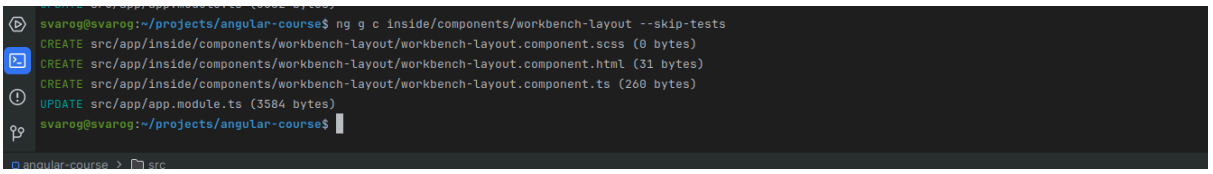
```
{path: 'login', component: LoginComponent},
```

Перевіримо взаємодію

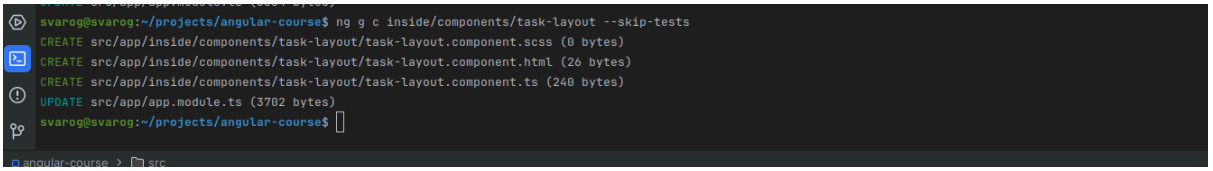


Відразу бачимо проблему, що авторизація відображається у полі контенту і користувачу вже одразу доступне навігаційне меню особистого кабінету. Це відбувається із-за того, що головний <router-outlet> у нас зараз у app.component.html, як і логіка відображення навігації та взаємодії з нею. Нам необхідно створити компонент, який буде відповідати за структуру для особистого кабінету і саме у ньому буде рендерится меню та взаємодія з завданнями.

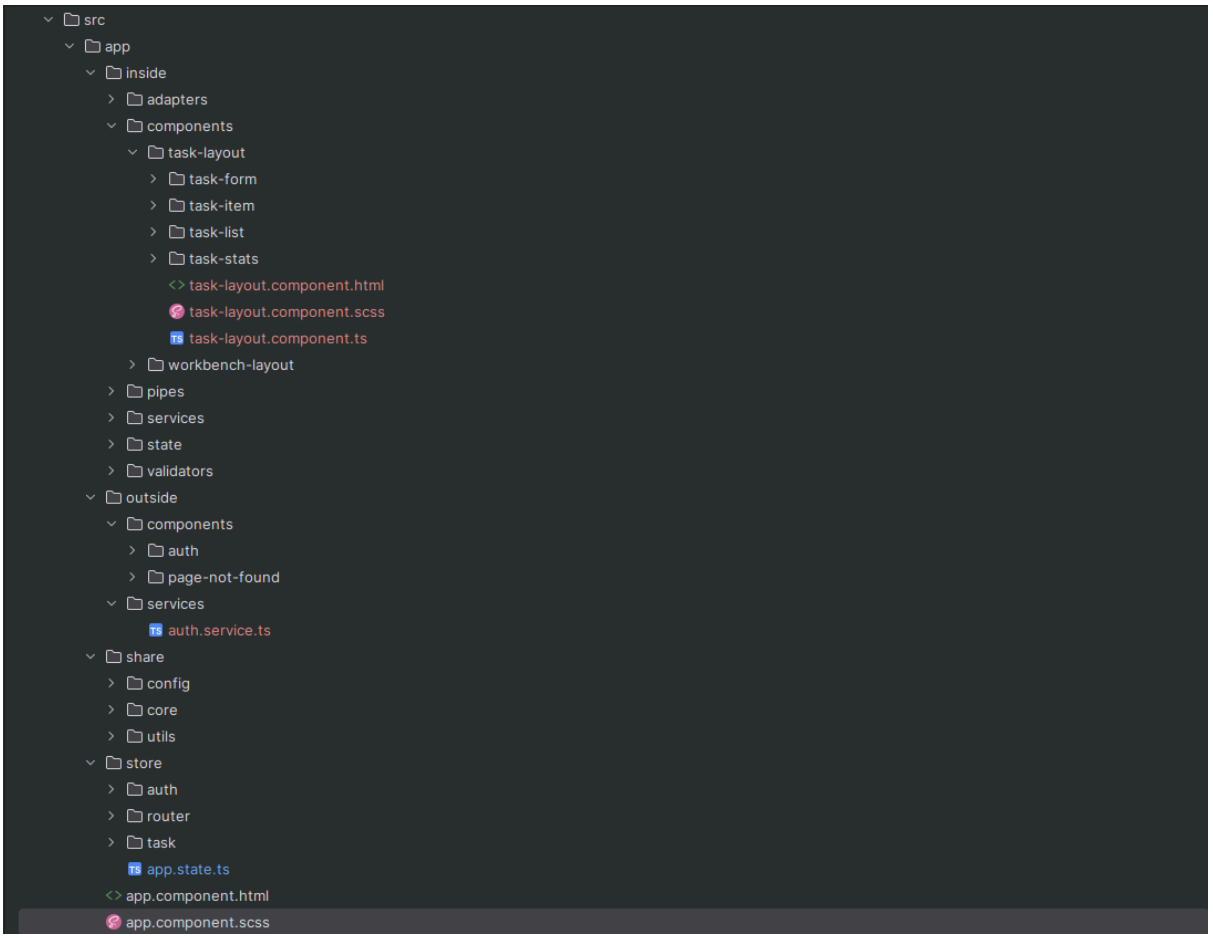
Для початку створимо компонент, який буде головним (батьківським) для компонентів, які відповідають за особистий кабінет. Він буде мати назву workbench-layout.



Також створимо task-layout, який буде батьківським для всього, що пов’язано з завданнями



Ми створили компоненти у новій теці app/inside - вона буде контейнером для всього, що пов’язано з внутрішньою логікою (опціонально, перенесіть всі компоненти, сервіси тощо, які мають відношення до внутрішньої логіки. Не забудьте перевірити імпорти). Зовнішню логіку переносимо до теку outside. Теку core переносимо у share, так як може бути необхідно використовувати наші інтерфейс з двох сторін (якщо ні, то розбиваємо за теками).



Тепер переносимо код із app.component.html/ts/scss у наші новостворені компоненти.

workbench-layout.component.html

```
<mat-sidenav-container class="workbench-container">
 <mat-sidenav mode="side" opened class="workbench-sidenav">
            <mat-list>
            <mat-list-item>
            <mat-icon matListItemIcon>list</mat-icon>
            <a matListItemTitle [routerLink]="[{ outlets: { primary: ['tasks', 'list'], aside: ['stats'] } }]"
routerLinkActive="active">
            <span>Список завдань</span>
            </a>
            </mat-list-item>
            </mat-list>
 </mat-sidenav>

 <mat-sidenav-content class="workbench-sidenav-content">
            <div class="workbench-content">
            @if (isNamedOutlet$ | async) {
            <aside class="app-aside">
            <router-outlet name="aside"></router-outlet>
```

```
            </aside>
            <main class="workbench-main">
            <router-outlet></router-outlet>
            </main>
            } @else {
            <main class="workbench-main">
            <router-outlet></router-outlet>
            </main>
            }
            </div>
  </mat-sidenav-content>
</mat-sidenav-container>
```

workbench-layout.component.ts

```
export class WorkbenchLayoutComponent implements OnInit{
  @ViewChild('drawer') drawer!: MatSidenav;

  isNamedOutlet$!: Observable<boolean>;

  constructor(private store: Store<AppState>) {}

  ngOnInit(): void {
            this.isNamedOutlet$ = this.store.select(selectUrl).pipe(
            filter(url => !!url),
            map(url => url.startsWith('/workbench/(tasks/list//aside:stats)')),
            );
  }
}
```

Стилі беремо з app.component.scss та змінюємо app- на workbench-. Також під нашу нову структуру перепишемо роутінг

```
const routes: Routes = [
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {path: 'workbench', component: WorkbenchLayoutComponent,
    children: [
      {path: 'tasks', component: TaskLayoutComponent,
        children: [
          {path: 'list', component: TaskListComponent},
          {path: 'view/:id', component: TaskItemComponent},
        ]
      },
      {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
    ]
  },
  {path: 'login', component: LoginComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

У app.component.html та task-layout.component.html прописуємо тільки <router-outlet>

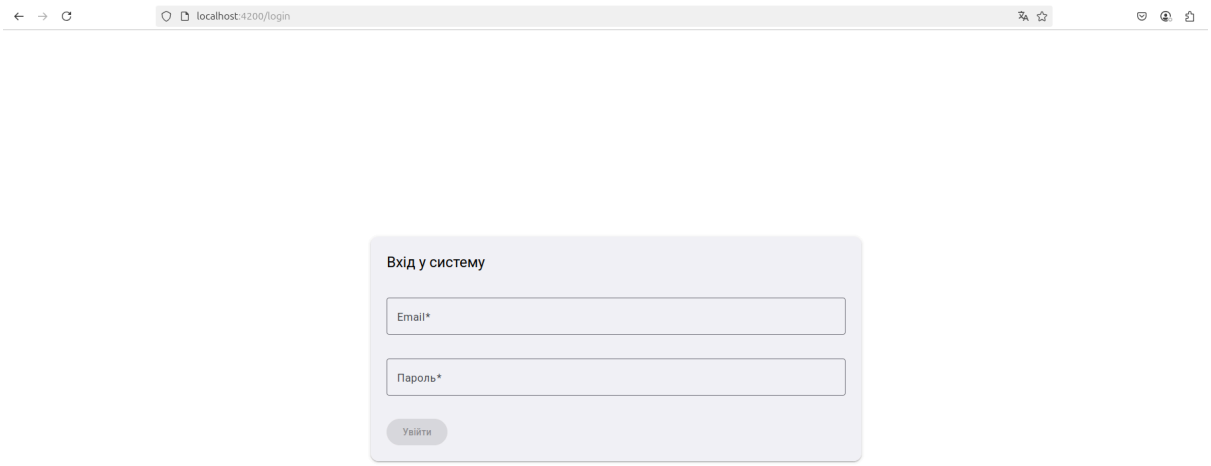
Стилі для app.component.scss

```scss
:host {
  height: 100%;
  display: block;
}
```

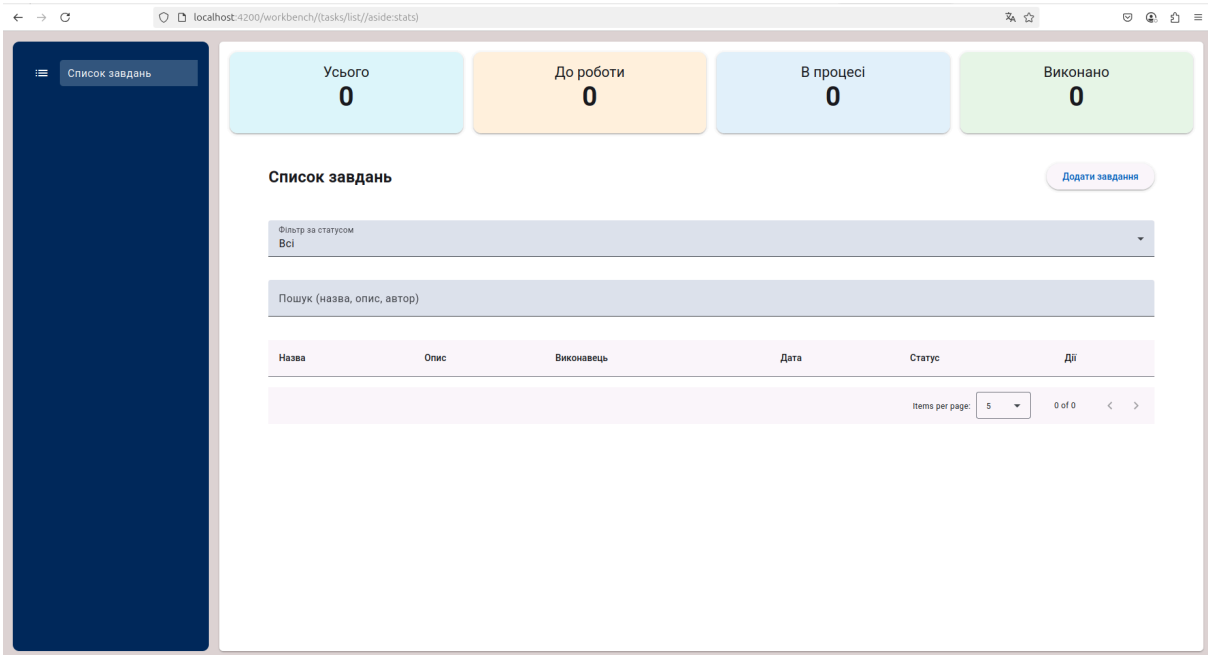
Також перепишемо редирект після входу

```
loginRedirect$ : Observable<{ user: AuthUser; token: strin... = createEffect(
  () : Observable<{ user: AuthUser; token: strin... =>
    this.actions$.pipe(
      ofType(AuthActions.loginSuccess),
      tap(() : Promise<boolean> => this.router.navigate([
        '/workbench',
        {
          outlets: {
            primary: ['tasks', 'list'],
            aside: ['stats']
          }
        }
      ]))
    ),
  { dispatch: false }
);
```

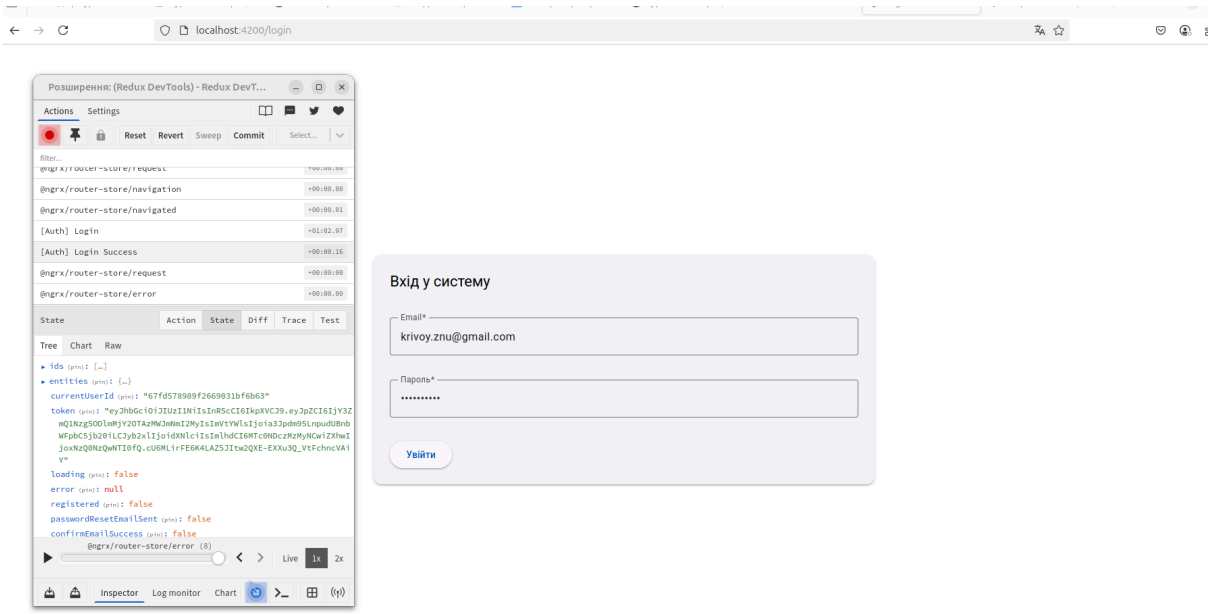
Перевіримо тепер відображення форми



Та відображення особистого кабінету (поки немає захисту)



Перевіримо вхід



Далі переходимо до реєстрації.
Створимо компонент register.component

```
svarog@svarog:~/projects/angular-course$ ng g c outside/components/auth/register --skip-tests
CREATE src/app/outside/components/auth/register/register.component.scss (0 bytes)
CREATE src/app/outside/components/auth/register/register.component.html (23 bytes)
CREATE src/app/outside/components/auth/register/register.component.ts (229 bytes)
UPDATE src/app/app.module.ts (3978 bytes)
svarog@svarog:~/projects/angular-course$
```

Реалізуємо логіку

```
export class RegisterComponent implements OnInit, OnDestroy {
  registerForm!: FormGroup;
  loading$!: Observable<boolean>;
  error$!: Observable<string | null>;
  registered$!: Observable<boolean>;

  private destroy$ = new Subject<void>();

  constructor(
              private fb: FormBuilder,
              private store: Store<AppState>,
              private snackBar: MatSnackBar,
  ) {}

  ngOnInit(): void {
              this.loading$ = this.store.select(AuthSelectors.selectAuthLoading);
              this.error$ = this.store.select(AuthSelectors.selectAuthError);
              this.registered$ = this.store.select(AuthSelectors.selectRegistered);
```

```
            this.error$.pipe(takeUntil(this.destroy$)).subscribe(error => {
            if (error) {
            this.snackBar.open(error, 'Закрити', { duration: 10000, panelClass: ['error-snackbar'] });
            }
            });

            this.registerForm = this.fb.group({
            email: ['', [Validators.required, Validators.email]],
            password: ['', [Validators.required, Validators.minLength(8)]],
            });
  }

  onSubmit(): void {
            if (this.registerForm.invalid) return;

            const { email, password } = this.registerForm.value;
            this.store.dispatch(AuthActions.register({ email, password }));
  }

  ngOnDestroy(): void {
            this.destroy$.next();
            this.destroy$.complete();
  }
}
```

Шаблон

```
<div class="register-container">
  <mat-card class="register-card">
            <mat-card-title>Реєстрація</mat-card-title>

            <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
            <mat-form-field appearance="outline">
            <mat-label>Email</mat-label>
            <input matInput formControlName="email" type="email" />
            @if(registerForm.get('email')?.hasError('required')){
            <mat-error>
            Email обов’язковий
            </mat-error>
            }
            @if(registerForm.get('email')?.hasError('email') && registerForm.get('email')?.dirty){
            <mat-error>
            Некоректний email
            </mat-error>
            }
            </mat-form-field>

            <mat-form-field appearance="outline">
            <mat-label>Пароль</mat-label>
            <input matInput formControlName="password" type="password" />
            @if(registerForm.get('password')?.hasError('required') && registerForm.get('password')?.dirty){
            <mat-error>
            Пароль обов’язковий
            </mat-error>
            }
            @if(registerForm.get('password')?.hasError('minlength')){
            <mat-error>
```

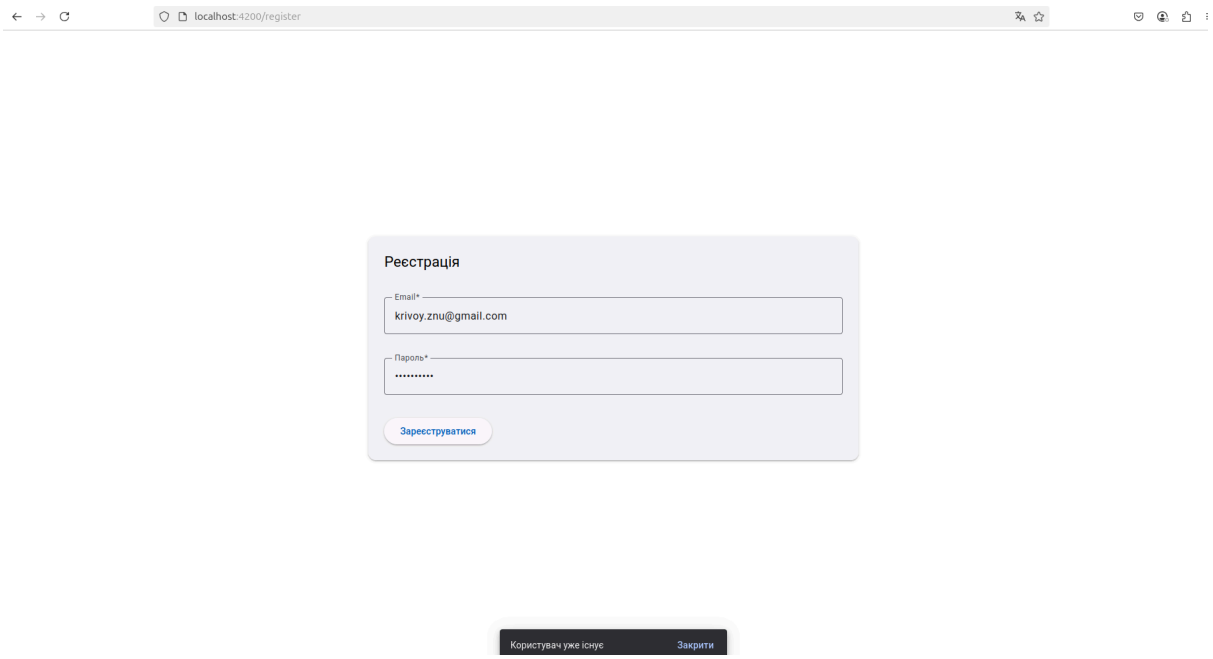
```
        Пароль повинен містити мінімум {{registerForm.get('password')?.getError('minlength').requiredLength}}
символів
        </mat-error>
        }
        </mat-form-field>

        <button mat-raised-button color="primary" type="submit" [disabled]="registerForm.invalid || (loading$ | async)">
        Зареєструватися
        </button>
        </form>
 </mat-card>
</div>
```

Стилі, як у login (змінюємо всі login на register).

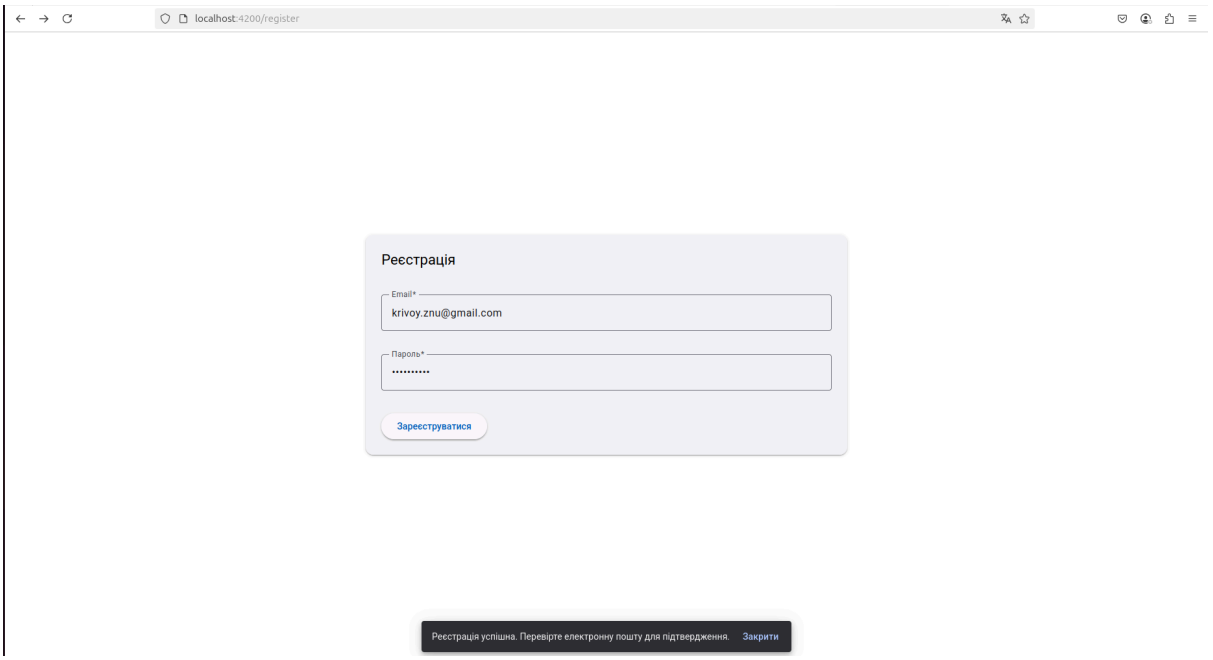
Реєструємо у роутах

```
const routes: Routes = [
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {path: 'workbench', component: WorkbenchLayoutComponent,
    children: [
      {path: 'tasks', component: TaskLayoutComponent,
        children: [
          {path: 'list', component: TaskListComponent},
          {path: 'view/:id', component: TaskItemComponent},
        ]
      },
      {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
    ]
  },
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

Перевіримо



Видалимо користувача та спробуємо знов



Тепер не будемо використовувати Postman, а відразу перейдемо до підтвердження з пошти.

Створимо компонент confirm-email.

```
svarog@svarog:~/projects/angular-course$ ng g c outside/components/auth/confirm-email --skip-tests
CREATE src/app/outside/components/auth/confirm-email/confirm-email.component.scss (0 bytes)
CREATE src/app/outside/components/auth/confirm-email/confirm-email.component.html (28 bytes)
CREATE src/app/outside/components/auth/confirm-email/confirm-email.component.ts (248 bytes)
UPDATE src/app/app.module.ts (4110 bytes)
svarog@svarog:~/projects/angular-course$
```

Реалізуємо логіку

```
export class ConfirmEmailComponent implements OnInit, OnDestroy {

  error$!: Observable<string | null>;
  message!: string;

  private destroy$ = new Subject<void>();

  constructor(private route: ActivatedRoute, private store: Store<AppState>,) {}

  ngOnInit(): void {
            this.error$ = this.store.select(AuthSelectors.selectAuthError);
            const token = this.route.snapshot.paramMap.get('token');

            if (token) {
            this.store.dispatch(AuthActions.confirmEmail({ token }));
            }

            this.store.select(AuthSelectors.selectConfirmEmailSuccess).pipe(takeUntil(this.destroy$))
            .subscribe(success => {
            this.message = success ? 'Пошта підтверджена' : 'Очікуємо підтвердження...';
            });

  }

  ngOnDestroy(): void {
            this.destroy$.next();
            this.destroy$.complete();
  }
}
```

Шаблон

```html
<div class="confirm-container">
  <mat-card>
    <mat-card-title>Підтвердження пошти</mat-card-title>
    @if (error$ | async; as error) {
      <mat-card-content>
        <p class="error-text">
          {{ error }}
        </p>
      </mat-card-content>
    } @else {
      <mat-card-content>
        <p>
          {{ message }}
        </p>
      </mat-card-content>
    }
  </mat-card>
</div>
```

Стилі

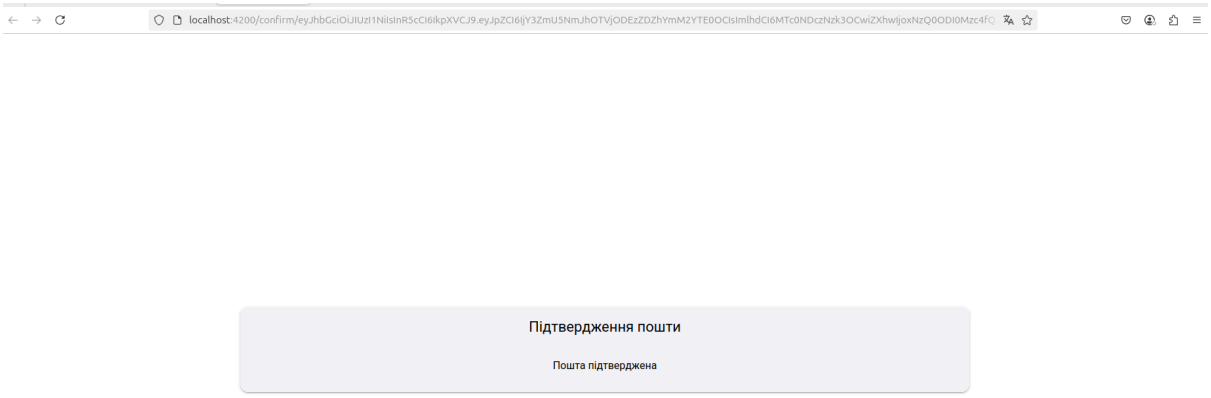
```css
:host {
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}

.confirm-container {
  display: flex;
  justify-content: center;
  width: 60%;
}

.confirm-card {
  flex-grow: 1;
  align-items: center;
  gap: 20px;
  padding: 15px;
}

.error-text {
  color: red;
}
```

Зареєструємо роут під компонент

```
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {path: 'workbench', component: WorkbenchLayoutComponent,
    children: [
      {path: 'tasks', component: TaskLayoutComponent,
        children: [
          {path: 'list', component: TaskListComponent},
          {path: 'view/:id', component: TaskItemComponent},
        ]
      },
      {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
    ]
  },
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
  {path: 'confirm/:token',component: ConfirmEmailComponent},
  {path: '**', component: PageNotFoundComponent}
];

@NgModule({  Show usages  Kryvyi Yaroslav
```

Змінимо посилання у email-сервісі на сервері та перезапустимо його (потрібно видалити користувача та пройти процес реєстрації знов)

```
exports.sendConfirmationEmail = async (email, token) : Promise<void>  => {
  const url : string  = `${process.env.FRONTEND_URL}/confirm/${token}`;

  try {
    await transporter.sendMail({
      from: `"Task App" <${process.env.SMTP_EMAIL}>`,
      to: email,
      subject: 'Підтвердження реєстрації',
      html: `<p>Натисніть <a href="${url}">тут</a>, щоб підтвердити свій акаунт.</p>`
    });
    console.log('Email sent successfully');
  } catch (error) {
    console.error('Error sending email:', error);
  }
};

exports.sendResetPasswordEmail = async (email, token) : Promise<void>  => {
  const url : string  = `${process.env.FRONTEND_URL}/${token}`;
```



За аналогією робить компонент запиту на відновлення та відновлення пароля.

Переходимо до відправки токена у headers та захисту маршрутів в Angular, бо зараз ми можемо перейти за захищеним на сервері маршрутом, але не можемо отримати дані. Але це все ще поганий досвід для користувача.

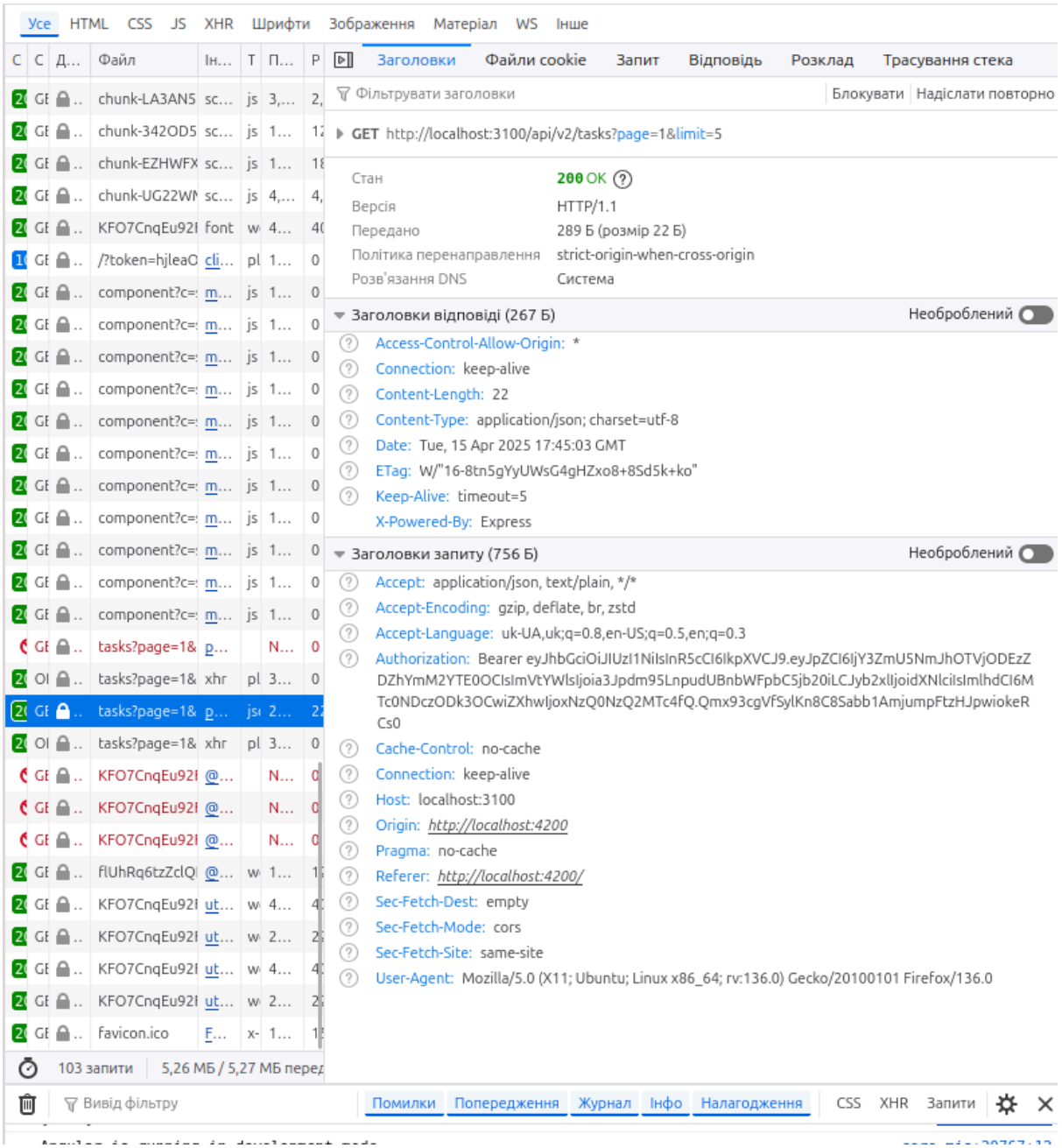
Почнемо зі створення Interceptor. Створимо файл outside/interceptors/auth.interceptor.ts

```
@Injectable()  Show usages
export class AuthInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}  no usages

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {  no usages
    const token : string | null  = this.authService.getToken();

    if (token) {
      const cloned : HttpRequest<any>  = req.clone({
        setHeaders: {
          Authorization: `Bearer ${token}`
        }
      });
      return next.handle(cloned);
    }

    return next.handle(req);
  }
}
```

Підключимо його у app.module.ts

```
providers: [
  provideHttpClient(withInterceptorsFromDi()),
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
],
```

Тепер кожен наш запит передає токен



Далі реалізуємо Guard. Для цього створимо файл outside/guards/auth.guard.ts

```
svarog@svarog:~/projects/angular-course$ ng g g outside/guards/auth --skip-tests
✔ Which type of guard would you like to create? CanActivate
CREATE src/app/outside/guards/auth.guard.ts (128 bytes)
svarog@svarog:~/projects/angular-course$
```

Реалізуємо

```
import {CanActivate, Router} from '@angular/router';
import {Injectable} from '@angular/core';
import {AuthService} from '../services/auth.service';

@Injectable({ providedIn: 'root' }) no usages
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {} no usages

  canActivate(): boolean { no usages
    const isLoggedIn :boolean = this.authService.isLoggedIn();
    if (!isLoggedIn) {
      this.router.navigate(['/login']);
      return false;
    }
    return true;
  }
}
```

Підключаємо до маршрутів

```
const routes: Routes = [
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {path: 'workbench', component: WorkbenchLayoutComponent, canActivate: [AuthGuard],
    children: [
      {path: 'tasks', component: TaskLayoutComponent,
        children: [
          {path: 'list', component: TaskListComponent},
          {path: 'view/:id', component: TaskItemComponent},
        ]
      },
      {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
    ]
  },
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
  {path: 'confirm/:token',component: ConfirmEmailComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

У нашому випадку, достатньо захистити тільки батьківський і всі дочірні успадкують захист. Також може виникнути питання: “Чому не використали Store?”. Справа у тому, що ми отримуємо AuthUser тільки після входу до системи, поки ми не оновимо сторінку, ми зможемо звертатися до цієї частини Store. Але при оновлені сторінки Store повернеться до початкового стану. Цю проблему можна вирішити, якщо у кореневому компоненті, який відповідає за внутрішню сторону (outside) прописати отримання даних користувача. Звісно для цього на сервері необхідно створити функцію, яка ці дані буде повертати. Також це буде корисно, якщо ми виводимо інформацію про користувача, наприклад особистий кабінет.

3) Залишився останній пункт, пов’язаний з авторизацією. А саме - прив’язка дії із завданнями до користувача.

Почнемо з серверної частини, необхідно трохи змінити методи, щоб вони працювали під користувача і хоч ми тільки емулюємо рольову модель, для методу отримання завдань пропишемо логіку перевірки ролі.

```
const TaskModel = require('../models/task.model');

const getTasks = async (req, res) => {
 try {
        const page = parseInt(req.query.page) || 1;
        const limit = parseInt(req.query.limit) || 5;
        const skip = (page - 1) * limit;

        const filter = req.query.filter?.trim().toLowerCase() || '';
        const status = req.query.status;

        const isAdmin = req.user.role === 'admin';

        console.log(req.user.role);

        const searchQuery = filter
        ? {
        $or: [
        { title: { $regex: filter, $options: 'i' } },
        { description: { $regex: filter, $options: 'i' } },
        ]
        }
        : {};

        const statusQuery = status ? { status } : {};

        const query = {...searchQuery, ...statusQuery, ...(isAdmin ? {} : { assignee: req.user.id })};
```

```
            const [tasks, total] = await Promise.all([
            TaskModel.find(query).populate('assignee', 'email role').skip(skip).limit(limit),
            TaskModel.countDocuments(query)
            ]);

            console.log({ tasks, total });

            res.json({ tasks, total });
  } catch (error) {
            res.status(500).json({ message: "Error retrieving tasks", errors: error });
  }
};

const createTask = async (req, res) => {
  try {
            const newTask = new TaskModel({...req.body, assignee: req.user.id});
            await newTask.save();
            const populated = await newTask.populate('assignee', 'email role');
            res.status(201).json(populated);
  } catch (error) {
            if (error.name === "ValidationError") {
            const errors = Object.values(error.errors).map(err => err.message);
            return res.status(400).json({ message: "Validation error", errors });
            }
            if (error.code === 11000) {
            return res.status(400).json({ message: "Task already exists" });
            }
            res.status(400).json({ message: "Error creating task", errors: error });
  }
};

const updateTask = async (req, res) => {
  try {
            console.log(req.body);
            const updatedTask = await TaskModel.findOneAndReplace(
            { _id: req.params.id, assignee: req.user.id },
            {...req.body, assignee: req.user.id},
            { new: true, runValidators: true }
            ).populate('assignee', 'email role');

            if (!updatedTask) {
            return res.status(404).json({ message: "Task not found" });
            }

            res.json(updatedTask);
  } catch (error) {
            if (error.name === "ValidationError") {
            const errors = Object.values(error.errors).map(err => err.message);
            return res.status(400).json({ message: "Validation error", errors });
            }
            if (error.code === 11000) {
            return res.status(400).json({ message: "Task already exists" });
            }
            res.status(400).json({ message: "Error updating task", errors: error });
  }
};
```

```
const patchTask = async (req, res) => {
  try {
        const updatedTask = await TaskModel.findByIdAndUpdate(
        { _id: req.params.id, assignee: req.user.id },
        req.body,
        { new: true, runValidators: true }
        ).populate('assignee', 'email role');

        if (!updatedTask) return res.status(404).json({ message: "Task not found" });

        res.json(updatedTask);
  } catch (error) {
        if (error.name === "ValidationError") {
        const errors = Object.values(error.errors).map(err => err.message);
        return res.status(400).json({ message: "Validation error", errors });
        }
        if (error.code === 11000) {
        return res.status(400).json({ message: "Task already exists" });
        }
        res.status(400).json({ message: "Error partially updating task", errors: error });
  }
};

const deleteTask = async (req, res) => {
  try {

        const deletedTask = await TaskModel.findOneAndDelete({ _id: req.params.id, assignee: req.user.id });

        if (!deletedTask) {
        return res.status(404).json({ message: "Task not found or permission denied" });
        }

        const total = await TaskModel.countDocuments();

        res.json({ message: "Task deleted", total });
  } catch (error) {
        res.status(500).json({ message: "Error deleting task", errors: error });
  }
};

module.exports = { getTasks, createTask, updateTask, patchTask, deleteTask };
```

Що змінилося:

1) Додали перевірку прав для запиту отримання всіх завдань. Тепер якщо у користувача є права адміністратора, то він отримує всі задання, якщо ні - тільки свої;
2) Додали populate('assignee', 'email role') до функцій, які шукають/оновлюють задання, а також для логіки створення завдання;

3) Змінили метод у функції видалення на findOneAndDelete, щоб можна було шукати не тільки за ID задання, а і користувача. також трохи змінили опис повертаємої помилки;
4) Додали два критерії пошуку до всіх методів, які шукають/шукають та оновлюють/шукають та видаляють запис. За ID задання і користувача.

Ці дії дозволили створити функції, які “працюють під користувача”, а також створили універсальну функцію для отримання всіх завдань.

Далі необхідно створити нові інтерфейси зі сторони Angular та оновити існуючі. Створимо нові інтерфейси у user.model.ts.

```
export interface UserAPI {  no usages
  _id: string;
  email: string;
  role: string;
}

export interface User {  no usages
  id: string;
  email: string;
  role: string;
}
```

Оновимо task.model.ts

```
import { TaskStatus} from './status.enum';
import {User} from './user.model';

export interface Task { // експортуємо інтерфейс для використання у інших частинах коду  Show usages  Kryvyi Yaroslav *
  id: string;
  title: string; // заголовок завдання
  description?: string; // опис завдання (? - необов'язковий параметр)
  assignee: User; // виконавець
  dueDate: string; // дата дедлайну
  status: TaskStatus; // статус завдання
}

export interface TaskLoad {  Show usages  Kryvyi Yaroslav
  tasks: Task[];
  total: number;
}

```

Та task-api.model.ts

```typescript
import {UserAPI} from './user.model';

export interface TaskApi {  Show usages  Kryvyi Yaroslav *
  _id: string;
  title: string;
  description?: string;
  assignee: UserAPI;
  dueDate: string;
  status: string;
}

export type TaskMutationApi = Omit<TaskApi, '_id' | 'assignee'>;

export interface TaskLoadApi {  Show usages  Kryvyi Yaroslav
  tasks: TaskApi[];
  total: number;
}
```

Далі створимо новий адаптер user.adapter.ts

```typescript
import {User, UserAPI} from '../../share/core/models/user.model';

export class UserAdapter {  no usages
  static fromUserAPI(response: UserAPI): User {  no usages
    return {
      id: response._id,
      email: response.email,
      role: response.role
    }
  }

  static toUserAPI(user: User): string {  no usages
    return user.id;
  }
}
```

Та оновимо task.adapter.ts

```
import ...

export class TaskAdapter {  Show usages  Kryvyi Yaroslav *
  static fromAPI(response: TaskApi): Task {  Show usages  Kryvyi Yaroslav *
    return {
      id: response._id,
      title: response.title,
      description: response.description || '',
      assignee: UserAdapter.fromUserAPI(response.assignee),
      dueDate: response.dueDate?.split('T')[0],
      status: response.status as TaskStatus
    }
  }

  static fromLoadAPI(response: TaskLoadApi): TaskLoad {  Show usages  Kryvyi Yaroslav
    return {
      tasks: response.tasks.map(task :TaskApi => TaskAdapter.fromAPI(task)),
      total: response.total
    };
  }

  static toAPI(task: Task): TaskMutationApi {  Show usages  Kryvyi Yaroslav *
    return {
      title: task.title,
      description: task.description || '',
      // assignee: UserAdapter.toUserAPI(task.assignee), це поле нам не потрібно, якщо не використовуємо рольову модель
      dueDate: new Date(task.dueDate).toISOString(),
      status: task.status
    }
  }
```

Так як ми не використовуємо рольову модель, то поле assignee нам не потрібно при відправці даних до API.

Завдяки тому, що ми використовуємо інтерфейси та адаптери, нам не потрібно змінювати логіку у task.service.ts.

Далі відкорегуємо компоненти

task-list.component.ts

```
viewTask(id: string): void {  Show usages  Kryvyi Yaroslav *
  this.router.navigate([{ outlets: { primary: ['workbench', 'tasks', 'view', id], aside: null } }]);
}
```

У нас ускладнилася структура роутінгу, тому необхідно оновити навігацію до деталей про завдання.

task-list.component.html

```
<!-- Виконавець -->
<ng-container matColumnDef="assignee">
  <th mat-header-cell *matHeaderCellDef> Виконавець </th>
  <td mat-cell *matCellDef="let task"> {{ task.assignee.email }} </td>
</ng-container>
```

task.assignee тепер об’єкт зі своїми властивостями. Можна напряму звернутися до його властивостей або написати кастомний селектор, щоб витягувати користувача з завдання, що дасть більше керування.

task-item.component.html

```
<div class="task-field">
  <span class="label">Виконавець:</span>
  <span class="value">{{ task.assignee.email }}</span>
</div>
```

```
<div class="actions">
  <button mat-stroked-button color="primary" [routerLink]="['/workbench', { outlets: { primary: ['tasks', 'list'], aside: ['stats'] } }]"
  >
    <mat-icon>arrow_back</mat-icon>
    Назад
  </button>
</div>
```

Оновили звернення до assignee, а також навігацію кнопки “Назад”.

task-form.component.ts

```
this.taskForm = this.fb.group({
  id: [''],
  title: ['', Validators.required],
  description: ['', TaskFormValidator.forbiddenWordsValidator(['React', 'Vue'])],
  dueDate: ['',  [Validators.required, TaskFormValidator.dateValidator]],
  assignee: [''],
  status: [TaskStatus.TODO, Validators.required]
});
```

Зробили assignee необов’язковим (залишаємо у випадку використання рольової моделі, адаптер і так його видаляє перед відправкою запиту).

У **task-form.component.html** прибираємо поле assignee.

І останнє що залишається - кнопка виходу з системи

Оновимо структуру workbench-layout.component.html (саме mat-sidenav)

```
<mat-sidenav mode="side" opened class="workbench-sidenav">
  <div class="sidenav-content">
    <mat-list>
      <mat-list-item>
        <mat-icon matListItemIcon>list</mat-icon>
        <a matListItemTitle [routerLink]="[{ outlets: { primary: ['tasks', 'list'], aside: ['stats'] } }]"
          routerLinkActive="active">
          <span>Список завдань</span>
        </a>
      </mat-list-item>
    </mat-list>

    <div class="sidenav-action">
      <button mat-button (click)="logout()">Вихід</button>
    </div>
  </div>
```

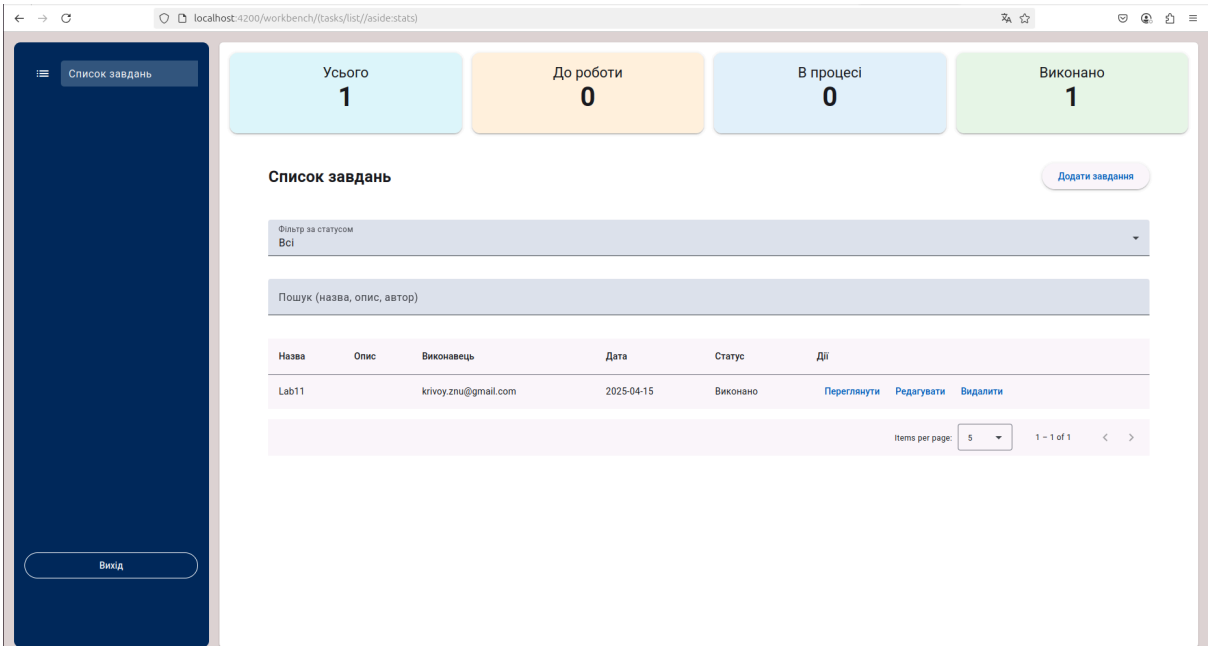
Стилізуємо

```
.workbench-sidenav  {
  margin: 16px;
  width: 300px;
  background: #002b5c;
  color: $color_1;
  border-radius: 10px;
  padding: 16px;
  text-align: center;

  .sidenav-content {
    display: flex;
    flex-direction: column;
    height: 80%;
    justify-content: space-between;
  }

    flex-direction: column;
    gap: 10px;
  }
  .sidenav-action {
    display: flex;
    justify-content: center;

    button {
      border: 1px solid $color_1;
      width: 100%;
      color: $color_1;

      &:hover {
        background-color: $color_1;
        color: $color_2;
      }
    }
  }
  mat-list-item {
```

Пропишемо логіку виходу

```
  styleUrl: './workbench-layout.component.scss'
})
export class WorkbenchLayoutComponent implements OnInit{
  @ViewChild('drawer') drawer!: MatSidenav;

  isNamedOutlet$!: Observable<boolean>;

  constructor(private store: Store<AppState>, private authService: AuthService, private router: Router) {}  no usages

  ngOnInit(): void {  no usages
    this.isNamedOutlet$ = this.store.select(selectUrl).pipe(
      filter(url : string  => !!url),
      map(url : string  => url.startsWith('/workbench/(tasks/list//aside:stats)')),
    );
  }

  logout(): void {  Show usages
    this.authService.logout();
    this.router.navigate(['/login']);
  }
}

```

Отримаємо



Контрольні питання

1) Що таке JWT? З яких частин складається?
2) Що може містити Payload?
3) Коли JWT-токен може не спрацювати?
4) Який метод дозволяє безпечно перевірити токен?
5) Чому важливо захищати маршрути? Які механізми для цього використовує Angular та Express.js?