

Лабораторна робота № 6. Вступ до Express.js та AsyncPipe Angular

Мета: ознайомитися та навчитися працювати з основними інструментами express щодо створення API-серверу. Навчитися обробляти потік даних за допомогою AsyncPipe в Angular.

Завдання

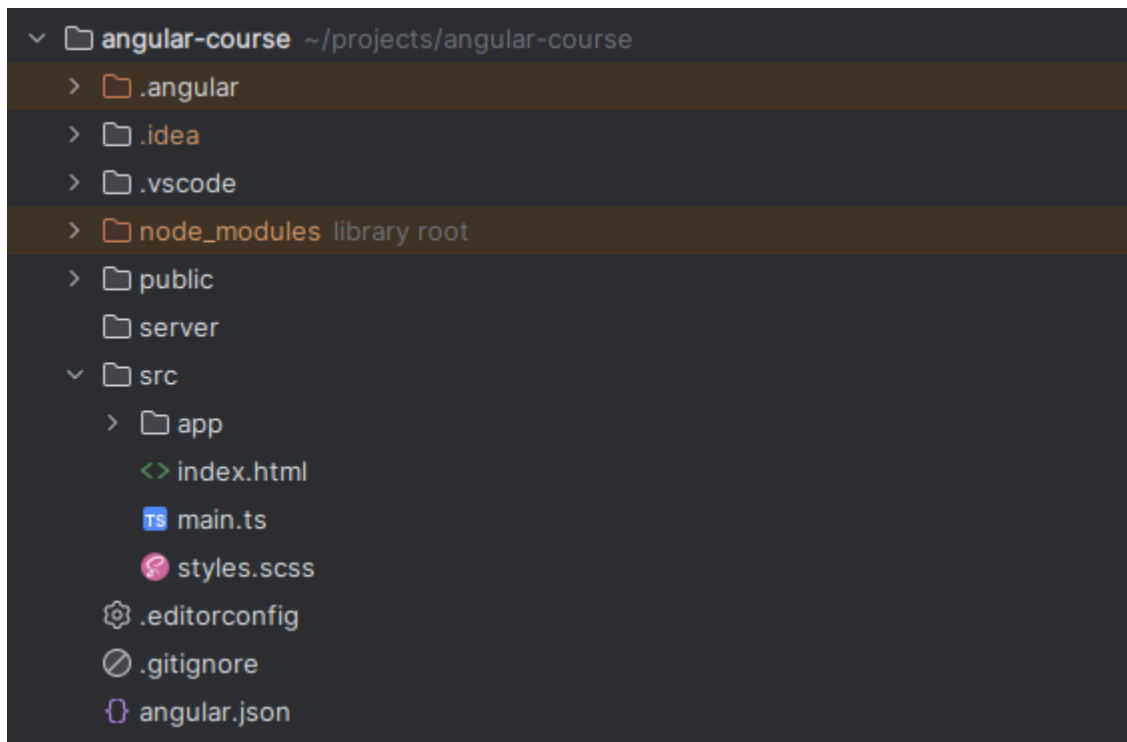
- 1) Ознайомитися з матеріалом лекції № 6;
- 2) Виконати пункти 1-7 ходу роботи;
- 3) У звіті відобразити кроки 1-7 ходу роботи;
- 4) Надати відповіді на контрольні питання.

Хід роботи

- 1) Для початку створимо сервер та ініціалізуємо package.json за допомогою команд у терміналі корня нашого Angular-проєкту

mkdir server && cd server/

```
svarog@svarog:~/projects/angular-course$ mkdir server && cd server/  
svarog@svarog:~/projects/angular-course/server$
```



Ініціалізуємо базовий `package.json` за допомогою команди

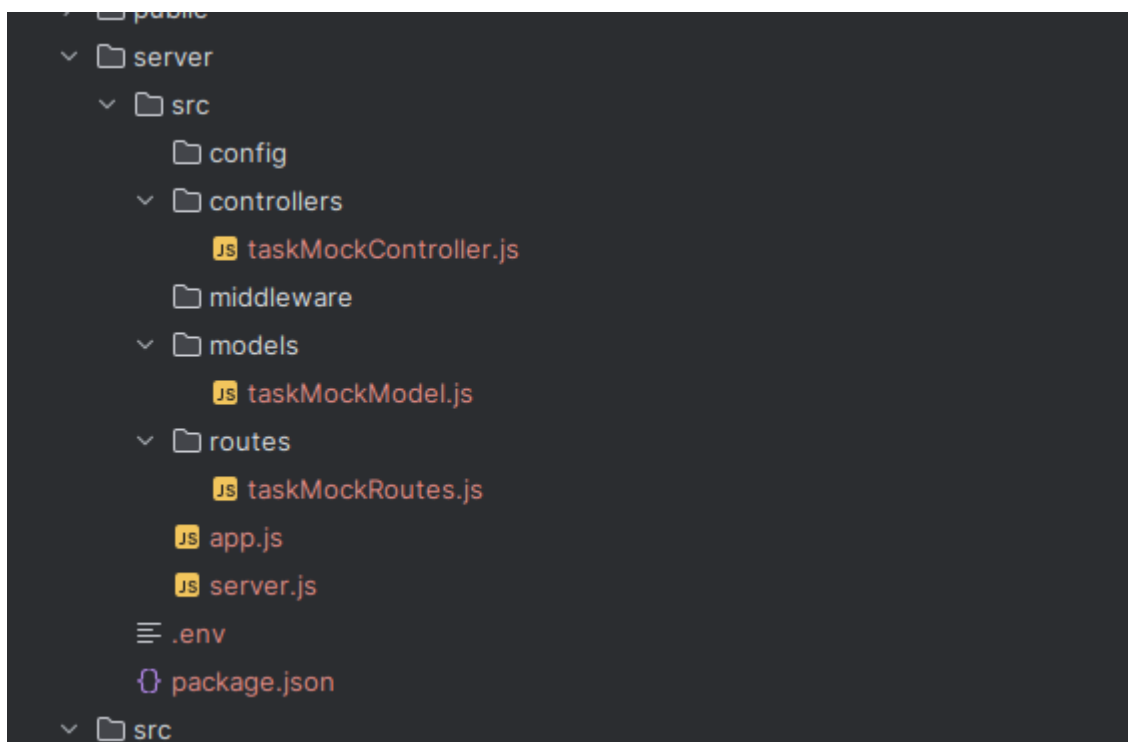
`npm init -y`

A screenshot of a terminal window with the title 'Terminal' and tabs 'Local', 'x', '+', and 'v'. The prompt is 'svarog@svarog:~/projects/angular-course/server\$'. The command 'npm init -y' has been executed, and the output shows the path to the newly created 'package.json' file. Below the path, the content of the 'package.json' file is displayed as a JSON object. At the bottom of the terminal, a breadcrumb path is shown: 'angular-course > src > app > services > task.service.ts > TaskService > m'.

Відразу додамо скрипт запуску до **scripts**

```
{
  "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": ""
}
```

2) Створимо базову структуру нашого REST API серверу



Розділяємо логіку наступним чином:

1. src/models/taskMockModel.js - аналог моделі, яка буде містити mock-дані;
2. src/controllers/taskMockController.js - контролер, який буде опрацьовувати звернення та повертати mock-дані;
3. src/routes/taskMockRoutes.js - прив'язка маршрутів до обробників HTTP-запитів у контролері;

4. `src/app.js` - файл, який містить необхідні налаштування для роботи сервера (підключення `middleware`, маршрутів тощо);
5. `src/server.js` - точка входу, файл який запускає сервер;
6. `.env` - містить конфіденційні змінні, які використовуються у проєкті.

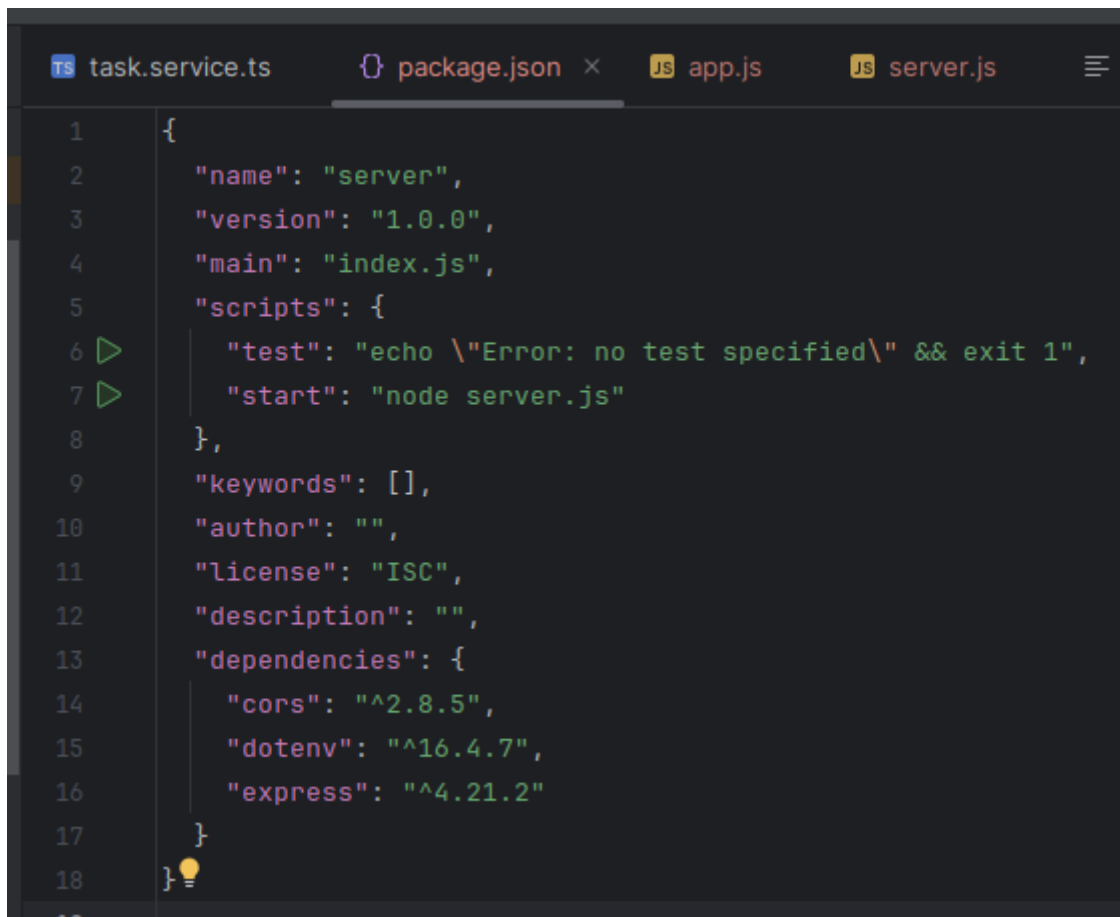
Також для роботи з деякими наступними елементами (да і для серверу в цілому) потрібно встановити **express**, **dotenv** та відразу **cors**.
Робимо це за допомогою команди

`npm install express cors dotenv`

```
svarog@svarog:~/projects/angular-course/server$ npm install express cors dotenv
added 72 packages, and audited 73 packages in 4s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```



```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1",
7     "start": "node server.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "description": "",
13  "dependencies": {
14    "cors": "^2.8.5",
15    "dotenv": "^16.4.7",
16    "express": "^4.21.2"
17  }
18 }
```

- 3) Реалізуємо покроково модель, контролер та роутер:
- а) Почнемо з моделі

У файлі **taskMockModel.js** пишемо наступний код:

```

1  const TaskStatus : {...} = {
2      TODO: 'TODO',
3      IN_PROGRESS: 'IN_PROGRESS',
4      DONE: 'DONE'
5  };
6
7  const tasks : [{id: number, title: string, a...} = [
8      {
9          id: 0,
10         title: 'Встановити Angular',
11         assignee: 'Ярослав',
12         dueDate: new Date( value: '2025-02-14'),
13         status: TaskStatus.DONE
14     },
15 ];
16
17 module.exports = { tasks };
18

```

При тесті з Angular змінити id на 1

Так як TaskStatus представляє з себе набір констант і може бути необхідність використовувати його у інших частинах серверу, то краще винести у окремий файл (наприклад, src/constants/taskStatus.js)

```

1  const TaskStatus : {...} = {
2      TODO: 'TODO',
3      IN_PROGRESS: 'IN_PROGRESS',
4      DONE: 'DONE'
5  };
6
7  module.exports = TaskStatus;

```

Підключаємо TaskStatus у файл **taskMockModel.ts**

```

1  const TaskStatus : {...} | {...} = require("../constants/taskStatus");
2
3  const tasks : [{id: number, title: string, a...} = [
4    {
5      id: 0,
6      title: 'Встановити Angular',
7      assignee: 'Ярослав',
8      dueDate: new Date( value: '2025-02-14'),
9      status: TaskStatus.DONE
10   },
11 ];
12
13 module.exports = { tasks };

```

b) Працюємо з контролером **taskMockController.js**

Для початку потрібно підключити до нього модель

```

JS app.js  JS server.js  .env  JS taskMockController.js  JS taskMockModel.js
const { tasks : [{id: number, title: string, a...} } = require('../models/taskMockModel');

```

Реалізовуємо метод **GET-запиту**

```

const { tasks : [{id: number, title: string, a...} } = require('../models/taskMockModel');

const getTasks = (req, res) : void => { no usages
  res.json(tasks);
}

```

Базовий метод, який повертає всі наявні завдання. При невеликій кількості записів (наприклад у БД) така реалізація має право на існування. Якщо записів багато, то краще додати можливість фільтрації та пагінації.

Реалізовуємо метод **POST**-запиту

```
const createTask = (req, res) :void => { Show usages
  const taskId :number = tasks.length ? Math.max(...tasks.map((task :{...}) :number => task.id)) + 1 : 1;
  const newTask :any & {...} = {
    id: taskId,
    ...req.body,
    dueDate: new Date(req.body.dueDate),
  }

  tasks.push(newTask);

  res.status(201).json(newTask);
}
```

Базовий метод, який приймає нове завдання, назначає йому id та записує до mock-моделі. Має певні проблеми з безпекою, так як не перевіряє чи всі обов'язкові властивості були передані та чи не містить об'єкт неправильних властивостей. У подальшому реалізуємо ці перевірки.

Реалізовуємо метод **PUT**-запиту

```
const updateTask = (req, res) :void => { Show usages
  const taskId :number = parseInt(req.params.id);
  const updatedTask :any & {...} = {...req.body, dueDate: new Date(req.body.dueDate),};

  const taskIndex :number = tasks.findIndex(task :{...} => task.id === taskId);
  if (taskIndex !== -1) {
    tasks[taskIndex] = { id: taskId, ...updatedTask };
    res.json(tasks[taskIndex]);
  } else {
    res.status(404).json({ message: 'Завдання не знайдено' });
  }
}
```

Базовий метод, що приймає id завдання через params та нове значення. Так як значення приходить, як string, то його потрібно конвертувати у int через функцію **parseInt**. Далі за допомогою **findIndex** шукаємо у нашій mock-моделі індекс завдання з вказаним id. Якщо знайдено, то повністю оновлюємо його, якщо ні, то надсилаємо статус 404. Має ті самі проблеми, що і POST.

Реалізовуємо метод PATCH-запиту

```
const patchTask = (req, res) :void => { no usages
  const taskId :number = parseInt(req.params.id);
  const updates = req.body;

  const task :{...} = tasks.find(task :{...} => task.id === taskId);
  if (task) {
    Object.assign(task, updates);
    res.json(task);
  } else {
    res.status(404).json({ message: 'Завдання не знайдено' });
  }
}
```

Базовий метод, що приймає id завдання через params та значення, які необхідно оновити. Так як значення приходить, як string, то його потрібно конвертувати у int через функцію **parseInt**. Далі за допомогою **find** шукаємо у нашій mock-моделі завдання з вказаним id. Якщо знайдено, то оновлюємо передані властивості за допомогою **Object.assign(target, source)**, який копіює всі поля (значення) з **source** у **target**, а так як **find** повертає по суті посилання на знайдене завдання, тому **Object.assign** оновлює значення напряму у mock-моделі **tasks**. Якщо завдання не знайдено, то надсилаємо статус 404. Має ті самі проблеми, що і POST за виключенням обов'язкових параметрів.

Реалізовуємо метод DELETE-запиту

```
const deleteTask = (req, res) :void => { no usages
  const taskId :number = parseInt(req.params.id);
  const taskIndex :number = tasks.findIndex(task :{...} => task.id === taskId);

  if (taskIndex !== -1) {
    tasks.splice(taskIndex, deleteCount: 1);
    res.json({ message: `Завдання ${taskId} видалене` });
  } else {
    res.status(404).json({ message: 'Завдання не знайдено' });
  }
}
```

Базовий метод, що приймає id завдання через params. Так як значення приходить, як string, то його потрібно конвертувати у int через функцію **parseInt**. Далі за допомогою **findIndex** шукаємо у нашій

mock-моделі індекс завдання з вказаним id. Якщо знайдено, то видаляємо його за допомогою функції **splice**, яка видаляє дані починаючи з вказаного індекса та необов'язкового параметра кількості елементів, які необхідно видалити, якщо ні, то надсилаємо статус 404.

У кінці файлу експортуємо наші обробники

```
task.splice(taskIndex, 1);  
res.json({ message: `Завдання ${taskId} видалене` });  
} else {  
  res.status(404).json({ message: 'Завдання не знайдено' });  
}  
}  
  
module.exports = { getTasks, createTask, updateTask, patchTask, deleteTask };
```

с) Переходимо до реалізації роутера **taskMockRouter.js**

Для початку підключаємо express.Router та файл контролера

```
const express :e|() => core.Express = require('express');  
const router :Router = express.Router();  
const taskMockController :{...}|{...} = require('../controllers/taskMockController');
```

Далі пов'язуємо маршрути та обробники

```
const express :e|() => core.Express = require('express');  
const router :Router = express.Router();  
const taskMockController :{...}|{...} = require('../controllers/taskMockController');  
  
router.get( path: '/', taskMockController.getTasks);  
router.post( path: '/', taskMockController.createTask);  
router.put( path:('/:id', taskMockController.updateTask);  
router.patch( path:('/:id', taskMockController.patchTask);  
router.delete( path:('/:id', taskMockController.deleteTask);
```

Експортуємо наші маршрути

```

router.post(path: '/', taskMockController.createTask);
router.put(path:('/:id', taskMockController.updateTask);
router.patch(path:('/:id', taskMockController.patchTask);
router.delete(path:('/:id', taskMockController.deleteTask);

module.exports = router;

```

- 4) Далі необхідно підключити наші маршрути у **app.js**, одразу підключимо вбудований middleware для роботи з json.

```

const express : e | () => core.Express = require('express');
const taskMockRoutes : Router | {...} = require('./routes/taskMockRoutes');

const app : any | Express = express();

app.use(express.json());
app.use('/api/v1/tasks', taskMockRoutes);

module.exports = app;

```

Тепер наші маршрути будуть доступні за **api/v1/tasks**

- 5) Налаштовуємо файл запуску сервера (**server.js**)

```

const app : ... = require('./app');
require("dotenv").config();

const PORT : string | number = process.env.PORT || 3000;

app.listen(PORT, hostname: () : void => {
  console.log(`Server running on port ${PORT}`);
})

```

Та прописуємо порт у **.env**

```
PORT = 3100
```

6) Запустимо наш сервер за допомогою **npm run start**

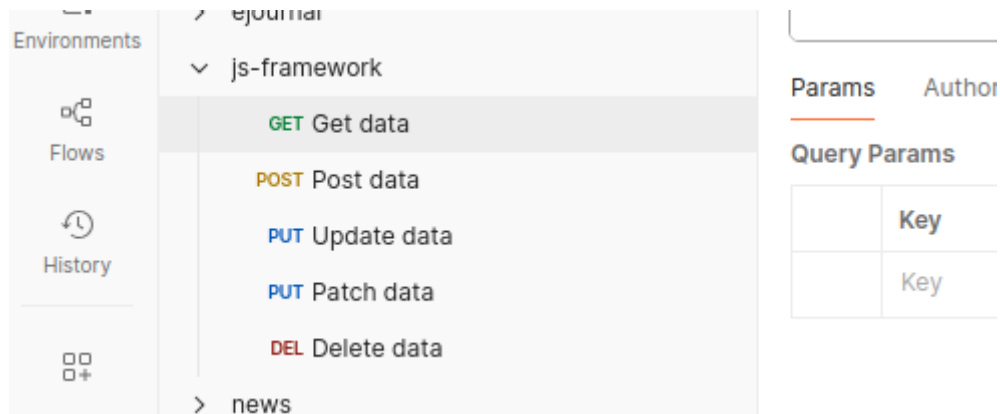
```
Terminal  Local x + v
svarog@svarog:~/projects/angular-course/server$ npm run start

> server@1.0.0 start
> node src/server.js

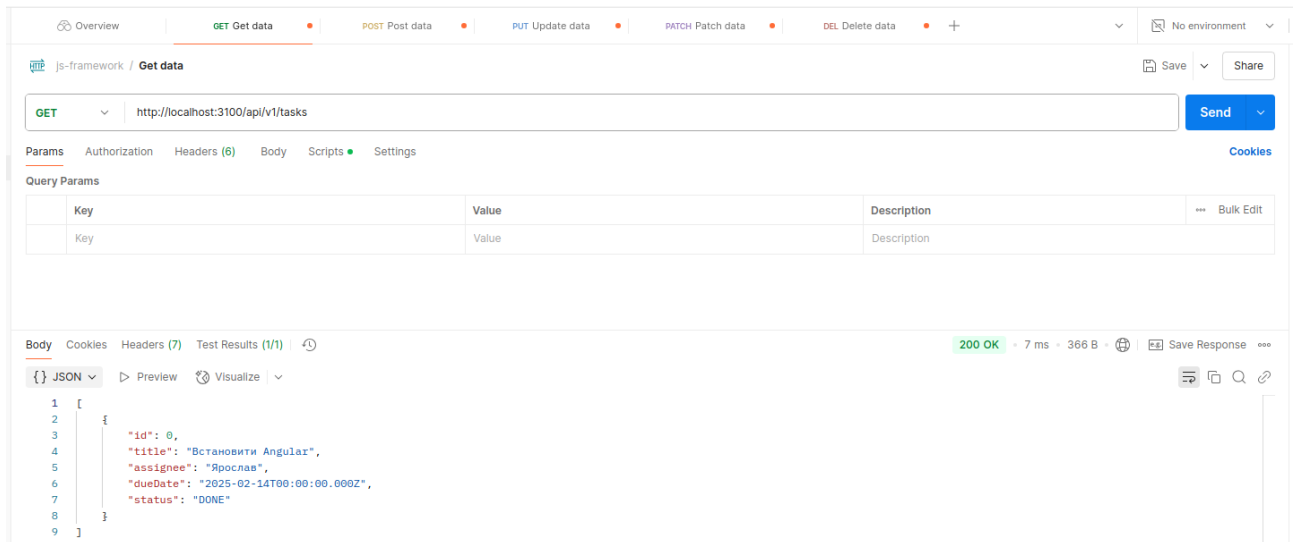
Server running on port 3100
```

Перед тим, як перейти до зв'язку з Angular необхідно протестувати наші маршрути. Для цього використаємо Postman-desktop.

Створимо колекцію для REST API



Тестуємо GET-запит за шляхом **http://localhost:3100/api/v1/tasks**



Тестуємо POST-запит за шляхом **http://localhost:3100/api/v1/tasks**

Якщо ми надсилаємо дані через **raw(JSON)**, як здебільше надсилає Angular, то ніякої додаткової обробки зі сторони express не потрібно, крім підключення **app.use(express.json())** у файлі **app.js**.

Overview GET Get data POST Post data PUT Update data PATCH Patch data

js-framework / Post data

POST http://localhost:3100/api/v1/tasks

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "title": "Ознайомитися з компонентами",
3   "description": "Ознайомитися з компонентами та оглянути взаємодію між ними",
4   "assignee": "Ярослав",
5   "dueDate": "2025-02-25",
6   "status": "IN_PROGRESS"
7 }
```

Body Cookies Headers (7) Test Results (1/1)

{ } JSON Preview Visualize

```
1 {
2   "id": 1,
3   "title": "Ознайомитися з компонентами",
4   "description": "Ознайомитися з компонентами та оглянути взаємодію між ними",
5   "assignee": "Ярослав",
6   "dueDate": "2025-02-25T00:00:00.000Z",
7   "status": "IN_PROGRESS"
8 }
```

Перевіряємо список всіх завдань

Overview GET Get data POST Post data PUT Update data PATCH Patch data

js-framework / Get data

GET http://localhost:3100/api/v1/tasks

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (7) Test Results (1/1)

JSON Preview Visualize

```
1 [
2   {
3     "id": 0,
4     "title": "Встановити Angular",
5     "assignee": "Ярослав",
6     "dueDate": "2025-02-14T00:00:00.000Z",
7     "status": "DONE"
8   },
9   {
10    "id": 1,
11    "title": "Ознайомитися з компонентами",
12    "description": "Ознайомитися з компонентами та оглянути взаємодію між ними",
13    "assignee": "Ярослав",
14    "dueDate": "2025-02-25T00:00:00.000Z",
15    "status": "IN_PROGRESS"
16  }
17 ]
```

Якщо використовувати `x-www-form-urlencoded`, то необхідно у `app.js` додати `app.use(express.urlencoded({ extended: true }))`

```
1 const express : e | () => core.Express = require('express');
2 const taskMockRoutes : Router | {...} = require('./routes/taskMockRoutes');
3
4 const app : any | Express = express();
5
6 app.use(express.json());
7 app.use(express.urlencoded({ options: { extended: true } }));
8
9 app.use('/api/v1/tasks', taskMockRoutes);
10
11 module.exports = app;
12
```

Тоді запит через **x-www-form-urlencoded** потрапить до **req.body**

The screenshot shows a REST client interface with the following components:

- Method and URL:** POST http://localhost:3100/api/v1/tasks
- Body Type:** x-www-form-urlencoded (selected)
- Body Data:** A table with 5 rows, each containing a checked checkbox, a key, and a value.

	Key	Value
☒	title	Ознайомитися з Control Flow
☒	assignee	Ярослав
☒	dueDate	2025-03-08
☒	status	DONE
	Key	Value

Below the table, the **Body** tab is active, showing the JSON representation of the request body:

```
1 {
2   "id": 2,
3   "title": "Ознайомитися з Control Flow",
4   "assignee": "Ярослав",
5   "dueDate": "2025-03-08T00:00:00.000Z",
6   "status": "DONE"
7 }
```

Для обробки form-data необхідно встановити пакет **multer** та підключити у **app.js**

```
const multer = require('multer');
```

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(upload.none());
```

Тестуємо PUT-запит за шляхом **http://localhost:3100/api/v1/tasks/1**

The screenshot shows a REST client interface with the following components:

- Method and URL:** PUT http://localhost:3100/api/v1/tasks/1
- Body Type:** raw (selected), JSON (available)
- Request Body:**

```
1 {  
2   "title": "Ознайомитися з компонентами v2",  
3   "assignee": "Ярослав",  
4   "dueDate": "2025-02-26",  
5   "status": "TODO"  
6 }
```
- Response Body:**

```
1 {  
2   "id": 1,  
3   "title": "Ознайомитися з компонентами v2",  
4   "assignee": "Ярослав",  
5   "dueDate": "2025-02-26T00:00:00.000Z",  
6   "status": "TODO"  
7 }
```

Перевіряємо відображення завдань

Overview

GET Get data

POST Post data

PUT Update data

PATCH Patch data

js-framework / Get data

GET

http://localhost:3100/api/v1/tasks

Params

Authorization

Headers (6)

Body

Scripts

Settings

Query Params

Key	Value
Key	Value

Body

Cookies

Headers (7)

Test Results (1/1)

🔄

{ } JSON

▶ Preview

📄 Visualize

▼

```
1  [
2    {
3      "id": 0,
4      "title": "Встановити Angular",
5      "assignee": "Ярослав",
6      "dueDate": "2025-02-14T00:00:00.000Z",
7      "status": "DONE"
8    },
9    {
10     "id": 1,
11     "title": "Ознайомитися з компонентами v2",
12     "assignee": "Ярослав",
13     "dueDate": "2025-02-26T00:00:00.000Z",
14     "status": "TODO"
15   },
16   {
17     "id": 2,
18     "title": "Ознайомитися з Control Flow",
19     "assignee": "Ярослав",
20     "dueDate": "2025-03-08T00:00:00.000Z",
21     "status": "DONE"
22   }
23 ]
```

І дійсно бачимо, що ми повністю перезаписали об'єкт

Тестуємо PATCH-запит за шляхом **http://localhost:3100/api/v1/tasks/1**

Overview

GET Get data

POST Post data

PUT Update data

PATCH Patch data

js-framework / Patch data

PATCH

http://localhost:3100/api/v1/tasks/1

Params

Authorization

Headers (8)

Body

Scripts

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

```
1 {
2   "status": "DONE"
3 }
```

Body

Cookies

Headers (7)

Test Results (1/1)

{ } JSON

Preview

Visualize

```
1 {
2   "id": 1,
3   "title": "Ознайомитися з компонентами v2",
4   "assignee": "Ярослав",
5   "dueDate": "2025-02-26T00:00:00.000Z",
6   "status": "DONE"
7 }
```

Тестуємо DELETE-запит за шляхом **http://localhost:3100/api/v1/tasks/1**

Overview

GET Get data

POST Post data

PUT Update data

PATCH Patch data

DEL Delete data

js-framework / Delete data

DELETE

http://localhost:3100/api/v1/tasks/1

Params

Authorization

Headers (6)

Body

Scripts

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body

Cookies

Headers (7)

Test Results (1/1)

{ } JSON

Preview

Visualize

```
1 {
2   |   "message": "Завдання 1 видалено"
3 }
```

Перевіряємо і бачимо, що завдання видалено

Overview

GET Get data

POST Post data

PUT Update data

PATCH Patch data

js-framework / Get data

GEThttp://localhost:3100/api/v1/tasks

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Query Params

Key	Value
Key	Value

BodyCookiesHeaders (7)Test Results (1/1)

{ } JSONPreviewVisualize

```
1  [
2    {
3      "id": 0,
4      "title": "Встановити Angular",
5      "assignee": "Ярослав",
6      "dueDate": "2025-02-14T00:00:00.000Z",
7      "status": "DONE"
8    },
9    {
10     "id": 2,
11     "title": "Ознайомитися з Control Flow",
12     "assignee": "Ярослав",
13     "dueDate": "2025-03-08T00:00:00.000Z",
14     "status": "DONE"
15   }
16 ]
```

А що поверне сервер, якщо ми спробуємо оновити або видалити неіснуючий запис?

Overview

GET Get data

POST Post data

PUT Update data

PATCH Patch data

DEL Delete data

js-framework / Update data

PUThttp://localhost:3100/api/v1/tasks/3

ParamsAuthorizationHeaders (8)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1  {
2    "title": "Ознайомитися з компонентами v2",
3    "assignee": "Ярослав",
4    "dueDate": "2025-02-26",
5    "status": "TODO"
6  }
```

BodyCookiesHeaders (7)Test Results (0/1)

{ } JSONPreviewVisualize

```
1  {
2    "message": "Завдання не знайдено"
3  }
```

404 Not Found

Overview GET Get data POST Post data PUT Update data PATCH Patch data DEL Delete data +

js-framework / Patch data

PATCH http://localhost:3100/api/v1/tasks/3

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "status": "DONE"
3 }
```

Body Cookies Headers (7) Test Results (0/1) 404 Not Found

{ } JSON Preview Visualize

```
1 {
2   "message": "Завдання не знайдено"
3 }
```

Overview GET Get data POST Post data PUT Update data PATCH Patch data DEL Delete data +

js-framework / Delete data

DELETE http://localhost:3100/api/v1/tasks/3

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results (0/1) 404 Not Found

{ } JSON Preview Visualize

```
1 {
2   "message": "Завдання не знайдено"
3 }
```

Як бачимо всі 3 запити відпрацювали коректно та повернули помилку 404, яку вже може обробити клієнт, а як себе поведе сервер у випадку передачі неіснуючих значень або якщо не передати всі обов'язкові поля? Перевіримо це за допомогою POST

The image displays two screenshots of a REST client interface, likely Postman, showing a POST request to `http://localhost:3100/api/v1/tasks`.

Top Screenshot (Request):

- Method: **POST**
- URL: `http://localhost:3100/api/v1/tasks`
- Body Type: **x-www-form-urlencoded**
- Body Data:

Key	Value	Description
☒ title	Ознайомитися з Control Flow	
☐ assignee	Ярослав	
☒ dueDate	2025-03-08	
☐ status	DONE	

Response: **201 Created** (3 ms)

Body (JSON):

```
1 {
2   "id": 3,
3   "title": "Ознайомитися з Control Flow",
4   "dueDate": "2025-03-08T00:00:00.000Z"
5 }
```

Bottom Screenshot (Response):

- Method: **POST**
- URL: `http://localhost:3100/api/v1/tasks`
- Body Type: **x-www-form-urlencoded**
- Body Data:

Key	Value	Description
☒ titleV1	Ознайомитися з Control Flow	
☐ assignee	Ярослав	
☒ dueDate	2025-03-08	
☐ status	DONE	

Response: **201 Created** (4 ms)

Body (JSON):

```
1 {
2   "id": 4,
3   "titleV1": "Ознайомитися з Control Flow",
4   "dueDate": "2025-03-08T00:00:00.000Z"
5 }
```

Можемо побачити, що сервер некоректно обробляє ці запити, що призводить до запису у `mock`-модель неправильних даних, а при використанні БД викличе помилку. Для того щоб уникнути таких ситуацій можна використовувати готові рішення щодо валідації, а саме пакет **express-validator**.

Інсталюємо його за допомогою команди ***npm install express-validator***

Далі створюємо файл за шляхом `src/validators/taskValidator.js` та пишемо наступний код

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

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

```
// ♦ Валідація `POST` та `PUT`
const validateTask = [
  check('title').notEmpty().withMessage('Поле title є обов'язковим'),
  check('assignee').notEmpty().withMessage('Поле assignee 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();
  }
];
```

```
// ♦ Валідація `PATCH` (перевіряємо тільки статус)
const validateTaskPatch = [
  check('status').optional().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();
  }
];
```

```
// ♦ Функція для перевірки помилок `express-validator`
const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
  next();
};
```

```
module.exports = { validateTask, validateTaskPatch, handleValidationErrors };
```

Основні методи перевірки значень

Метод	Опис	Приклад
.notEmpty()	Перевіряє, що значення не порожнє	check('title').notEmpty().withMessage('Поле title обов'язкове')
.isString()	Перевіряє, що значення рядком	check('title').isString().withMessage('Має бути текстом')
.isNumeric()	Перевіряє, що значення числом	check('price').isNumeric().withMessage('Має бути числом')
.isInt()	Перевіряє, що значення ціле число	check('age').isInt({ min: 18 }).withMessage('Має бути 18+')
.isBoolean()	Перевіряє, що значення булевым (true або false)	check('isActive').isBoolean().withMessage('Має бути true або false')
.isEmail()	Перевіряє, що значення є коректним email	check('email').isEmail().withMessage('Некоректний email')
.isISO8601()	Перевіряє, що значення є датою у форматі ISO	check('dueDate').isISO8601().toDate().withMessage('Некоректна дата')
.isLength({ min, max })	Перевіряє довжину рядка	check('password').isLength({ min: 6 }).withMessage('Пароль має бути 6+ символів')

<code>.matches(regex)</code>	Перевіряє, чи значення відповідає регулярному виразу	<code>check('username').matches(/^[a-zA-Z0-9]+\$/).withMessage('Тільки літери та цифри')</code>
------------------------------	---	---

Методи перевірки списків та специфічних значень

Метод	Опис	Приклад
<code>.isIn([...])</code>	Перевіряє, що значення є у списку дозволених	<code>check('status').isIn(['TODO', 'IN_PROGRESS', 'DONE']).withMessage('Недозволений статус')</code>
<code>.custom(value => ...)</code>	Використовує кастомну функцію для перевірки	<code>check('password').custom(value => value.includes('@')).withMessage('Має містити @')</code>
<code>.optional()</code>	Робить поле необов'язковим	<code>check('description').optional().isString()</code>

Обробка помилок

Завжди потрібно використовувати `validationResult()`, щоб отримати список помилок та передати їх у відповідь

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

const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
  next();
};
```

Після створення правил валідації підключаємо їх до файлу роутера та до потрібних маршрутів

```
pp.js  server.js  .env  taskMockController.js  taskMockModel.js  status.enum.ts  tasks.ts  taskMockRoutes.js x
1  const express :e| => core.Express  = require('express');
2  const router :Router  = express.Router();
3
4  const taskMockController :({})[{}]  = require('../controllers/taskMockController');
5  const {
6    validateTask :[ValidationChain, ValidationCh...,
7    validateTaskPatch :[ValidationChain, function(any...,
8    handleValidationErrors
9  } = require('../validators/taskValidator');
10
11  router.get( path: '/', taskMockController.getTasks);
12  router.post( path: '/', validateTask, handleValidationErrors, taskMockController.createTask);
13  router.put( path: '/:id', validateTask, handleValidationErrors, taskMockController.updateTask);
14  router.patch( path: '/:id', validateTaskPatch, handleValidationErrors, taskMockController.patchTask);
15  router.delete( path: '/:id', taskMockController.deleteTask);
16
17  module.exports = router;
18
```

Перевіряємо роботу

Search Postman

Overview GET Get data POST Post data PUT Update data PATCH Patch data DEL Delete data +

js-framework / Post data

POST http://localhost:3100/api/v1/tasks

Params Authorization Headers (8) Body Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description
☒ titleV1	Ознайомитися з Control Flow	
☐ assignee	Ярослав	
☒ dueDate	2025-03-08	
☐ status	DONE	

Body Cookies Headers (7) Test Results (0/1) 400 Bad Request 19 ms

{ } JSON Preview Visualize

```
1  {
2    "error": "Зайві поля: titleV1"
3  }
```

js-framework / **Post data**

POST `http://localhost:3100/api/v1/tasks`

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description
☒ title	Ознайомитися з Control Flow	
☒ assignee	Ярослав	
☒ dueDate	2025-03-08	
☐ status	DONE	

Body Cookies Headers (7) Test Results (0/1) 400 Bad Request • 6 ms • 38

```

1 {
2   "errors": [
3     {
4       "type": "field",
5       "msg": "Статус має бути одним із: TODO, IN_PROGRESS, DONE",
6       "path": "status",
7       "location": "body"
8     }
9   ]
10 }

```

js-framework / **Post data**

POST `http://localhost:3100/api/v1/tasks`

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description
☒ title	Ознайомитися з Control Flow	
☐ assignee	Ярослав	
☒ dueDate	2025-03-08	
☒ status	DONE	

Body Cookies Headers (7) Test Results (0/1) 400 Bad Request • 4 ms • 36

```

1 {
2   "errors": [
3     {
4       "type": "field",
5       "msg": "None assignee є обов'язковим",
6       "path": "assignee",
7       "location": "body"
8     }
9   ]
10 }

```

Тепер переходимо до вирішення наступної задачі. У попередніх роботах та у поточній ми конвертували дату між типами **string** та **Date**, як на стороні Angular, так і на стороні серверу. Для автоматизації цього процесу напишемо кастомний middleware.

Він буде мати назву **dateMiddleware.js**. Створюємо його у теці **middleware** та пишемо наступний код

```

1  const formatDate = (date) :null | string => { Show usages
2    if (!date) return null;
3    const d :Date = new Date(date);
4    return d.toISOString().split('T')[0]; // Отримуємо 'YYYY-MM-DD'
5  };
6
7  const parseDateMiddleware = (req, res, next) :void => { Show usages
8    if (req.body.dueDate) {
9      req.body.dueDate = new Date(req.body.dueDate); // Зберігаємо як `Date`
10   }
11   next();
12 };
13
14 const formatResponseDateMiddleware = (req, res, next) :void => { Show usages
15   const oldJson = res.json; // Зберігаємо оригінальний `res.json`
16   res.json = function (data) :void {
17     if (Array.isArray(data)) { // Якщо відповідь - масив об'єктів (наприклад, список завдань)
18       data = data.map(task => ({
19         ...task,
20         dueDate: task.dueDate ? formatDate(task.dueDate) : null,
21       }));
22     } else if (typeof data === 'object' && data !== null) { // Якщо відповідь - один об'єкт (одне завдання)
23       data.dueDate = data.dueDate ? formatDate(data.dueDate) : null;
24     }
25     oldJson.call(this, data); // Викликаємо оригінальний `res.json`
26   };
27   next();
28 };

```

Давайте трохи розберемо, що за що відповідає:

- 1) **formatDate** - не є **middleware**, використовуємо як допоміжну функцію для **formatResponseDateMiddleware**. По суті призначена для того щоб конвертувати **Date** у формат **'YYYY-MM-DD'**;
- 2) **parseDateMiddleware** - це вже **middleware**, по суті перетворює отриману від Angular дату (**YYYY-MM-DD**) у об'єкт **Date** (наприклад, перед збереженням у **mock**-модель або БД);
- 3) **formatResponseDateMiddleware** - це також **middleware**, що конвертує дату у відповідях сервера (**Date** -> **YYYY-MM-DD**) перед відправкою Angular.

Тепер підключаємо ці **middleware** у файлі **app.js**

```

1  const express :e | () => core.Express = require('express');
2
3  const taskMockRoutes :Router | {...} = require('./routes/taskMockRoutes');
4
5  const {
6    parseDateMiddleware :parseDateMiddleware | function(any, any, any): void ,
7    formatResponseDateMiddleware :formatResponseDateMiddleware | function(any, any, any): void
8  } = require('./middleware/dateMiddleware');
9
10 const app :any | Express = express();
11
12 app.use(express.json());
13 app.use(express.urlencoded( options: { extended: true }));
14
15 app.use(parseDateMiddleware);
16 app.use(formatResponseDateMiddleware);
17
18 app.use('/api/v1/tasks', taskMockRoutes);
19
20 module.exports = app;

```

Також можна прибрати взаємодію з датою у **createTask** та **updateTask**.

Наш невеликий сервер майже готов, єдине що залишилося, це підключити **cors** у файлі **app.js**

```

1  const express :e | () => core.Express = require('express');
2  const cors :<T=e.CorsRequest extends e.Cor... | e = require('cors');
3
4  const taskMockRoutes :Router | {...} = require('./routes/taskMockRoutes');
5
6  const {
7    parseDateMiddleware :parseDateMiddleware | function(any, any, any): void ,
8    formatResponseDateMiddleware :formatResponseDateMiddleware | function(any, any, any): void
9  } = require('./middleware/dateMiddleware');
10
11 const app :any | Express = express();
12
13 app.use(cors());
14
15 app.use(express.json());
16 app.use(express.urlencoded( options: { extended: true }));
17
18 app.use(parseDateMiddleware);
19 app.use(formatResponseDateMiddleware);
20
21 app.use('/api/v1/tasks', taskMockRoutes);
22
23 module.exports = app;

```

7) Переходимо до підключення нашого API до Angular

Для початку необхідно внести певні зміни у існуючі файли, враховуючи логіку сервера, а саме:

- 1) Змінимо значення кирилицею у файлі **status.enum.ts** на їх аналоги латиницею (зробимо, як на сервері)

```
1 export enum TaskStatus { Show usages  📄 Kryvyi Yaroslav *
2   TODO = 'TODO',
3   IN_PROGRESS = 'IN_PROGRESS',
4   DONE = 'DONE'
5 }
```

- 2) Приберемо з Angular логіку, що пов'язана з форматуванням дати

Для цього у файлі **task.model.ts** змінюємо тип властивості `dueDate` на `string` та образу робимо `id` необов'язковим параметром (не найкраща практика, приміняємо тільки у контексті навчання, можливі інші варіанти, коли `id` обов'язковий: окремий інтерфейс для POST, використання `Partial` (максимально незручно, якщо об'єкт містить багато властивостей), використання `id = null` (врахуйте зміни в інтерфейсі, а також додаткові перевірки))

```
1 import { TaskStatus } from './status.enum';
2
3 export interface Task { // експортуємо інтерфейс для використання у інших частинах коду
4   id?: number;
5   title: string; // заголовок завдання
6   description?: string; // опис завдання (? - необов'язковий параметр)
7   assignee: string; // виконавець
8   dueDate: string; // дата дедлайну
9   status: TaskStatus; // статус завдання
10 }
11
```

Модернізуємо **task-form.component.ts**

```

ngOnChanges(changes: SimpleChanges): void { no usages Kryvyi Yaroslav *
  if(changes['editTask'] && this.editTask) {
    this.taskForm.patchValue(this.editTask);
  }
}

addTask(): void { Show usages Kryvyi Yaroslav *
  if (this.taskForm.valid) {
    let taskData :{id: number | undefined; title?: string ...} = {
      ...this.taskForm.value,
      id: this.editTask ? this.editTask.id : undefined,
    };
    this.taskAdd.emit(taskData as Task);
    this.taskForm.reset();
  }
}

```

Так як id став необов'язковим параметром, то трохи модернізуємо функцію deleteTask у **task-item.component.ts**

```

@Output() taskEdited: EventEmitter<Task> = new EventEmitter<Task>(); // подія для редагування

deleteTask(id: number | undefined): void { Show usages new *
  if (!id) return; // Якщо id відсутній - виходимо
  this.taskDeleted.emit(id); // Викликаємо подію при натисканні кнопки "Видалити"
}

```

Тепер переходимо до адаптації нашого сервісу. Для початку запишемо url нашого REST API-сервера до змінної у config.ts

```

1 import {InjectionToken} from '@angular/core';
2
3 export interface AppConfig { Show usages
4   apiUrl: string;
5 }
6
7 export const APP_CONFIG = { Show usages
8   apiUrl: 'http://localhost:3100/api'
9 }
10
11 export const CONFIG_TOKEN = new InjectionToken<AppConfig>('CONFIG_TOKEN', { no usages
12   providedIn: 'root',
13   factory: () :{apiUrl: string} => APP_CONFIG
14 });

```

Підключимо HttpClient у **app.module.ts**

```
8 import { TaskListComponent } from '../components/task-list/task-list.component';
9 import { TaskItemComponent } from '../components/task-item/task-item.component';
10 import { StatusFilterPipe } from '../share/pipes/status-filter.pipe';
11 import { TaskFormComponent } from '../components/task-form/task-form.component';
12 import {provideHttpClient, withInterceptorsFromDi} from '@angular/common/http';
13
14 @NgModule({ Show usages  Kryvyi Yaroslav
15   declarations: [
16     AppComponent,
17     TaskListComponent,
18     TaskItemComponent,
19     StatusFilterPipe,
20     TaskFormComponent
21   ],
22   imports: [
23     BrowserModule,
24     AppRoutingModule,
25     FormsModule,
26     ReactiveFormsModule
27   ],
28   providers: [
29     provideHttpClient(withInterceptorsFromDi()),
30   ],
31   bootstrap: [AppComponent]
32 })
33 export class AppModule { }
```

Модифікуємо TaskService, для початку інжектимо необхідні залежності (HttpClient, CONFIG_TOKEN)

```
1 import {Inject, Injectable} from '@angular/core';
2 import {Task} from '../core/models/task.model';
3 import {HttpClient} from '@angular/common/http';
4 import {AppConfig, CONFIG_TOKEN} from '../share/config/config';
5
6 @Injectable({ Show usages  Kryvyi Yaroslav *
7   providedIn: 'root'
8 })
9 export class TaskService {
10
11   constructor( no usages new *
12     private http: HttpClient,
13     @Inject(CONFIG_TOKEN) private config: AppConfig
14   ) { }
15 }
```

Реалізовуємо методи

```

16
17 getTasks(): Observable<Task[]> { Show usages new *
18   return this.http.get<Task[]>(`${this.config.apiUrl}/v1/tasks`);
19 }
20
21 createTask(newTask: Task): Observable<Task> { Show usages new *
22   return this.http.post<Task>(`${this.config.apiUrl}/v1/tasks`, newTask);
23 }
24
25 updateTask(id: number, updateTask: Task): Observable<Task> { Show usages new *
26   const { id: _, ...update } = updateTask;
27   return this.http.put<Task>(`${this.config.apiUrl}/v1/tasks/${id}`, update);
28 }
29
30 patchTask(id: number, updateTask: Partial<Task>): Observable<Task> { Show usages new *
31   return this.http.patch<Task>(`${this.config.apiUrl}/v1/tasks/${id}`, updateTask);
32 }
33
34 deleteTask(id: number): Observable<void> { Show usages new *
35   return this.http.delete<void>(`${this.config.apiUrl}/v1/tasks/${id}`);
36 }
37
38 }
39

```

Модифікуємо **task-list.component.ts**

```
export class TaskListComponent implements OnInit {
```

```
  myTasks$: Observable<Task[]>;
```

```
  selectedStatus!: TaskStatus | 'all';
```

```
  editingTask: Task | null = null;
```

```
  constructor(private taskService: TaskService) {
  }

```

```
  ngOnInit(): void {
    this.myTasks$ = this.taskService.getTasks();
  }

```

```
  loadTasks(): void {
    this.myTasks$ = this.taskService.getTasks();
  }

```

```
  addTask(task: Task): void {
    if (this.editingTask) {
      if (!task.id) return;
      this.taskService.updateTask(task.id, task).subscribe({

```

```

        next: () => this.loadTasks(),
        error: error => console.log(error),
    })
    this.editingTask = null;
  } else {
    this.taskService.createTask(task).subscribe( {
      next: () => this.loadTasks(),
      error: error => console.log(error),
    });
  }
}

editTask(task: Task): void {
  this.editingTask = {...task};
}

deleteTask(id: number): void {
  this.taskService.deleteTask(id).subscribe( {
    next: () => this.loadTasks(),
    error: error => console.log(error),
  });
}

onSelected(event: Event): void {
  const status = (event.target as HTMLSelectElement).value;
  this.selectedStatus = status as TaskStatus | 'all';
}

protected readonly TaskStatus = TaskStatus;
}

```

Також оновимо метод **updateStatus** для демонстрації роботи patch у **task-item.component.ts**

```

updateStatus(event: Event) : void { Show usages  Kryvyi Yaroslav *
  const selectedValue : string = (event.target as HTMLSelectElement).value;
  if (!this.task.id) return;

  this.taskService.patchTask(this.task.id, { status: selectedValue as TaskStatus }).subscribe({
    next: updatedTask : Task => this.task.status = updatedTask.status,
    error: error : any => console.error('Помилка оновлення статусу', error),
  });
}

```

Оновлюємо **task-list.component.ts** додаючи до циклу `@for` async pipe

```

<app-task-form
  [editTask]="editingTask"
  (taskAdd)="addTask($event)"
></app-task-form>
<h2>Список завдань</h2>
<div class="task-filter">
  <p>Фільтр за статусом</p>
  <select (change)="onSelected($event)">
    <option [value]="all">Всі</option>
    <option [value]="TaskStatus.TODO">До роботи</option>
    <option [value]="TaskStatus.IN_PROGRESS">В процесі</option>
    <option [value]="TaskStatus.DONE">Виконано</option>
  </select>
</div>
@for(task of myTasks$ | async; track task.id; let index = $index) {
  <app-task-item
    [task]="task"
    (taskDeleted)="deleteTask($event)"
    (taskEdited)="editTask($event)"
  >
  </app-task-item>
} @empty {
  <p>Завдання відсутні</p>
}

```

Перевіряємо роботу додатка

← → ↺

localhost:4200

🔍 ☆

🏠 ⬇️ 🗑️ ⚙️

Назва

Опис

Дата дедлайну

дд . мм . рrrr

📅

Виконавець

Статус

До роботи

▼

Додати нове завдання

Список завдань

Фільтр за статусом

Всі ▼

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

🚩 До роботи ▼

Редагувати

Видалити

Додаємо нове завдання

← → ↺

localhost:4200

🔍 ☆

🏠 ⬇️ 🗑️ ⚙️

Назва

Моє перше завдання з express

Опис

Дата дедлайну

14 . 03 . 2025

📅

Виконавець

Ярослав

Статус

До роботи

▼

Додати нове завдання

Список завдань

Фільтр за статусом

Всі ▼

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

🚩 До роботи ▼

Редагувати

Видалити

localhost:4200

Дата дедлайну

дд . мм . рrrr

Виконавець

Статус

Додати нове завдання

Список завдань

Фільтр за статусом

Всі

- Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити
- Моє перше завдання з express

Виконавець: Ярослав

Дата виконання: 14.03.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити

Оновлюємо завдання

localhost:4200

Дата дедлайну

дд . мм . рrrr

Виконавець

Статус

Додати нове завдання

Список завдань

Фільтр за статусом

Всі

- Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити
- Моє перше завдання з express

Виконавець: Ярослав

Дата виконання: 14.03.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити

localhost:4200

Назва

Моє перше завдання з express

Опис

Тепер у нас є опис

Дата дедлайну

14 . 03 . 2025

Виконавець

Ярослав

Статус

До роботи

Оновити завдання

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

До роботи

Редагувати

Видалити

Моє перше завдання з express

Виконавець: Ярослав

Дата виконання: 14.03.2025

Додати нове завдання

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

До роботи

Редагувати

Видалити

Моє перше завдання з express

Опис: Тепер у нас є опис

Виконавець: Ярослав

Дата виконання: 14.03.2025

Статус: TODO

До роботи

Редагувати

Видалити

Видалення

Додати нове завдання

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити

Моє перше завдання з express

Опис: Тепер у нас є опис

Виконавець: Ярослав

Дата виконання: 14.03.2025

Статус: TODO

🚩 До роботи

Редагувати

Видалити

← → ↺

localhost:4200

🔍 ☆

📄 📌 📧 📁 ☰

Назва

Опис

Дата дедлайну

дд . мм . рrrr

Виконавець

Статус

До роботи

Додати нове завдання

Список завдань

Фільтр за статусом

Всі

Моє перше завдання з express

Опис: Тепер у нас є опис

Виконавець: Ярослав

Дата виконання: 14.03.2025

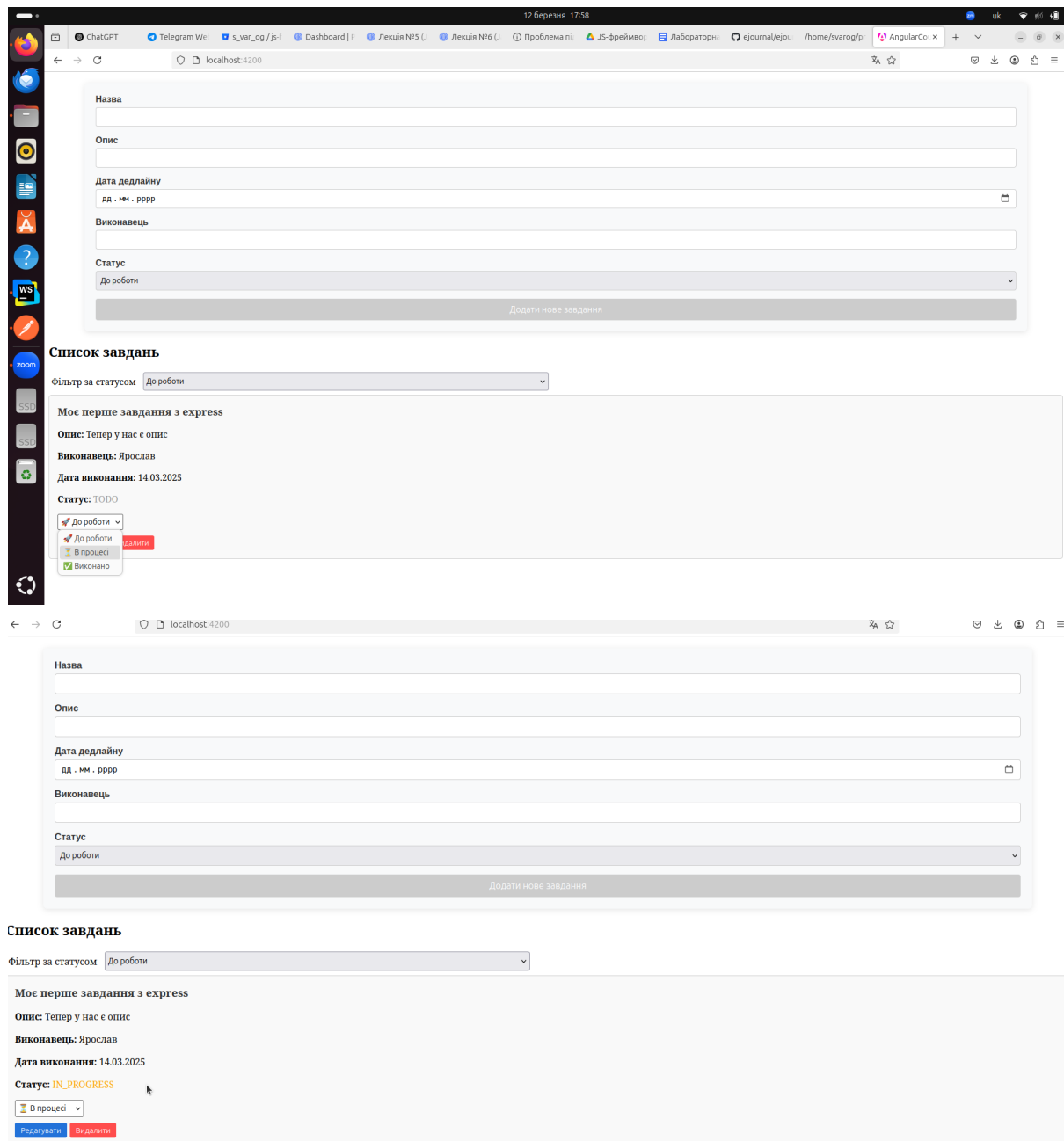
Статус: TODO

🚩 До роботи

Редагувати

Видалити

Часткове оновлення



Вирішимо ще дві проблеми. Перша заключається у тому, що статуси тепер виводяться на латиниці та більше не працює фільтр за статусом (бо прибрати з `@for` наш кастомний `pipe`). Переклад вирішимо за допомогою іншого кастомного пайпу, можна звісно використати `@switch`, але якщо потрібно буде використовувати переклад у різних частинах коду, то це буде надлишково. Створюємо наш кастомний пайп у тій же теці, що і попередній:

```
svarog@svarog:~/projects/angular-course$ ng g p share/pipes/task-status
CREATE src/app/share/pipes/task-status.pipe.spec.ts (204 bytes)
CREATE src/app/share/pipes/task-status.pipe.ts (246 bytes)
UPDATE src/app/app.module.ts (1118 bytes)
```

Реалізуємо пайп, він повинен приймати TaskStatus

```
@Pipe({ Show usages
  name: 'taskStatus',
  standalone: false
})
export class TaskStatusPipe implements PipeTransform {

  transform(status: TaskStatus): string { Show usages
    const statusMap: { [key in TaskStatus]: string } = {
      [TaskStatus.TODO]: 'До роботи',
      [TaskStatus.IN_PROGRESS]: 'В процесі',
      [TaskStatus.DONE]: 'Виконано'
    };
    return statusMap[status] || 'Невідомий статус';
  }
}
```

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

☒ Виконано

Редагувати

Видалити

Далі реалізуємо фільтрацію, для цього спочатку модернізуємо get-запити на стороні сервера та Angular

```

getTasks(status?: string): Observable<Task[]> { Show usages new *
  let params :HttpParams = new HttpParams();

  if (status) params = params.set('status', status);

  return this.http.get<Task[]>(`${this.config.apiUrl}/v1/tasks`, {params: params});
}

```

Тепер метод приймає необов'язковий параметр статусу та перевіряє, якщо він є, то записує його як параметр та передаємо серверу.

```

const getTasks = (req, res) :void => { Show usages
  let filteredTasks :({})[] = [...tasks];

  if (req.query.status) {
    filteredTasks = filteredTasks.filter(task :({}) => task.status === req.query.status);
  }

  res.json(filteredTasks);
}

```

На сервері також робимо перевірку, якщо статус наявний у запиті, то фільтруємо та повертаємо тільки завдання з вказаним статусом, якщо ні, то всі завдання.

Далі необхідно внести деякі зміни у **task-list.component.ts**

```

loadTasks(status?: string): void { Show usages new *
  this.myTasks$ = this.taskService.getTasks(status);
}

```

Для **loadTasks** також вказуємо необов'язковий параметр статусу

```

selectedStatus: TaskStatus | '' = '';

```

```
onSelected(event: Event): void { Show usages  ⤴ Kryvyi Yaroslav *  
    const status :string = (event.target as HTMLSelectElement).value;  
    this.selectedStatus = status as TaskStatus | '';  
    this.loadTasks(this.selectedStatus);  
}
```

Тепер при зміні статусу буде автоматично викликатися функція **loadTasks**, якій автоматично буде передаватися поточний статус.

Список завдань

Фільтр за статусом

Виконано

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

☒ Виконано

Редагувати

Видалити

Список завдань

Фільтр за статусом

В процесі

Завдання відсутні

Якщо є потреба при додаванні, оновленні або видаленні залишати поточну фільтрацію, то необхідно передавати змінну **selectedStatus** при виклику методу **loadTasks** відповідних методів додавання, оновлення або видалення.

Контрольні питання

- 1) Яка роль та зона застосування middleware у Express.js? Який метод дозволяє їх підключати?
- 2) Яке призначення req у Express.js?
- 3) Яке призначення res у Express.js?
- 4) Який метод групує маршрути за одним шляхом?
- 5) Для чого використовується AsyncPipe в Angular?