Node.js

Mongoose

Rest API

Завдання

Розробити RESTful API-додаток з обробкою таких запитів для даних users і tasks:

- GET (/users, /tasks)
- GET (/users/:id, /tasks/:id)
- POST (/users, /tasks)
- PATCH (/users/:id, /tasks/:id)
- DELETE (/users/:id, /tasks/:id)
- DELETE (/users, /tasks)

Встановити

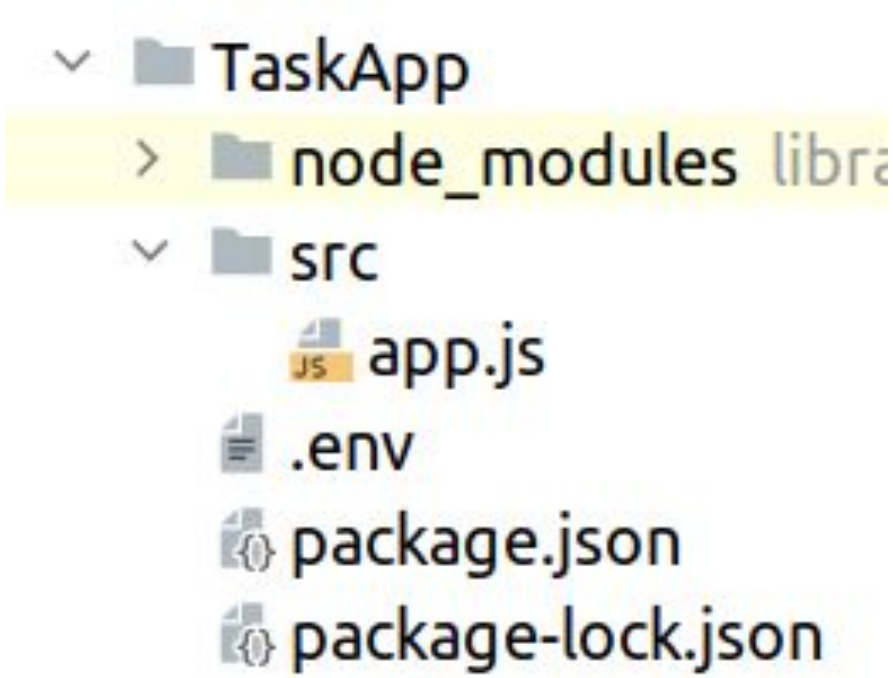
Сервер БД MongoDB або хмарний варіант

Postman – створення і виконання запитів

Для початку роботи використайте заготовку:
https://gist.github.com/alexanderkuzmenko/76fae3cbc2ab6b51b5cddc35c629b052

Завдання 1. Створіть проект TaskApp

- Встановіть модулі *mongoose*, *validator*, *express, dotenv*
- Створіть папку *src* та виконавчий файл *src/app.js*



Завдання 2. Підключимось до бази даних та створимо модель User

- Рядок з’єднання винесіть у файл .env в змінну MONGO_URL

```
8   //Створюємо модель
9   const User = mongoose.model('User', {
10      name: {type: String},
11      age: {type: Number}
12  });
```

Створимо екземпляр моделі User

```
14  //Створюємо екземпляр моделі
15  const user = new User({name: 'Alex', age: 34});
16
17  //Зберігаємо екземпляр в базу даних та виводимо повідомлення
18  user.save().then(() => {
19      console.log(user);
20  }).catch((error) => {
21      console.log(error);
22  });
```

Протестуйте додаток. Перевірте валідацію полів при введенні значень невірних типів

Валідація даних

- Для поля *name* моделі *User* встановіть обов’язкову наявність непорожнього значення:

```
name: {
    type: String,
    required: true
},
```

- Перевірте роботу валідатора даних при створенні екземпляра з порожнім полем *name*

Валідація даних:

- Для поля *age* моделі *User* створіть метод *validate(), в* якому забезпечте виключення від’ємних значень:

```
age: {
    type: Number,
    validate(value) {
        if (value < 0) {
            throw new Error( message: "Age must be a positive number");
        }
    }
}
```

Validator

- Для більш складніших перевірок (emails, passwords, phone numbers) можна використовувати модуль npm-validator
- Встановіть та підключіть даний модуль в проект
- Для моделі User створіть поле *email* та зробіть перевірку на його коректність
- При неправильному заданні email, повинно виводитись повідомлення:

```
Error: Email is invalid
```

Фільтрація даних (Data Sanitization)

- Додайте до поля name моделі User обрізку крайніх пробілів:

```
name: {
    type: String,
    required: true,
    trim: true
},
```

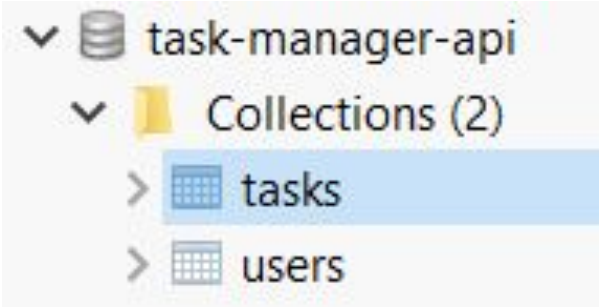
- Додайте до поля *email* моделі User конвертацію в нижній регістр
- Для поля *age* задайте значення по замовчуванню *0*

Завдання для моделі User

1. Створіть поле *password* і зробіть його обов’язковим
2. Довжина поля не повинна бути меншою ніж 7
3. Обріжте крайні пробіли в полі password
4. Пароль не повинен містити слово *“password”*
5. Для поля *email* встановіть його унікальність (відсутність однакових значень)
6. **Винесіть модель User в файл */models/user.js* та підключіть його в додаток як модуль**
7. Протестуйте роботу

Завдання для моделі Task

1. В файлі */models/task.js* створіть модель *Task* з полями *title (String)*, *description (String)* та *completed (Boolean)*
2. Зробіть поле *title, description* обов’язковим та з обрізкою крайніх пробілів
3. Make *completed* optional and default it to false
4. Test your work with and without errors
5. Створіть новий екземпляр моделі
6. Збережіть об’єкт в базі даних
7. Перевірте наявність БД *task-manager-api* та колекцій *tasks* та *users*



Структуруйте проект

- Винесіть моделі User і Task у відповідні файли
- Підключення до БД також винесіть в окремий файл
- Забезпечте підключення даних файлів як модулів в додатку index.js:

```
1  const express = require( id: "express");
2  require( id: './db/mongoose');
3  const User = require( id: './models/user');
4  const Task = require( id: './models/task');
```

Частина 2. REST API

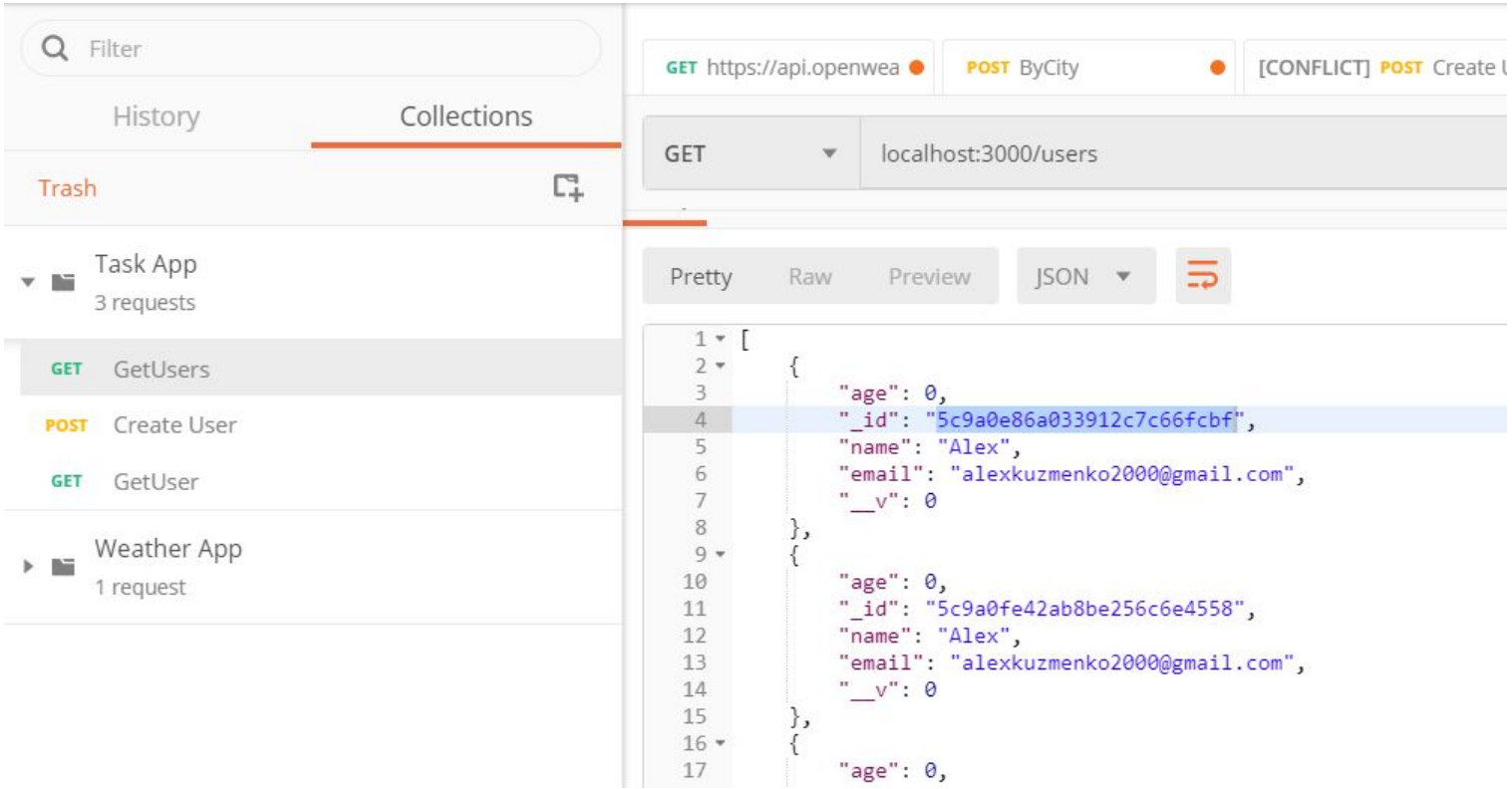
Обробка GET-запиту

- В додатку створіть обробник запиту на отримання всіх користувачів з БД:

```
20  app.get("/users", (req, res) => {
21      User.find({}).then((users) => {
22          res.status(200).send(users);
23      }).catch((error) => {
24          res.status(500).send();
25      });
26  });
```

З допомогою програми POSTMAN протестуйте виконання даного запиту:

- Створіть категорію Task App
- В цій категорії створіть та протестуйте запит GetUsers



Завдання. Створіть обробку запитів

- отримання користувача з БД по id
- додавання нового користувача в БД
- Створіть відповідний набір операцій REST API для моделі Task

Здійсніть рефакторинг коду в додатку Task Application з використанням *async/await*

Приклад

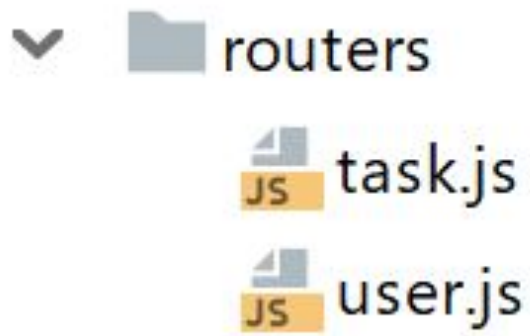
```
app.post("/places", async(req, res) => {
    let place = new Place(req.body);
    try {
        await place.save();
        res.status(201).send(place);
    } catch (e) {
        res.status(500).send();
    }
})
```

Створіть обробку запиту видалення по id для моделей User і Task

- Шлях для видалення */user/:id*
- Операція повинна включати блоки *try/catch*
- Обробник повинен включати *async/await*
- Результат видалення повинен бути одним із:
  - Success (200)
  - Not Found (404)
  - Error (400)

Separate Route Files (маршрутизатори)

- Розділіть всі обробники маршрутів для користувачів і задач на два окремі файли, що міститимуться src/routers:



- Дані файли називаються маршрутизаторами. В кожному маршрутизаторі повинні міститись по 6 обробників маршрутів

Приклад створення роутера

- В файлі-маршрутизаторі routers/user.js створимо роутер:

```
const express = require("express");
const router = new express.Router();

router.get('/test', (req, res) => {
    res.send("From a new File");
})

module.exports = router;
```

- В index.js підключимо роутер:

```
const userRouter = require('./routers/user');
```

- Додамо його в middleware додатку:

```
app.use(userRouter);
```

- Перевіримо виконання: