



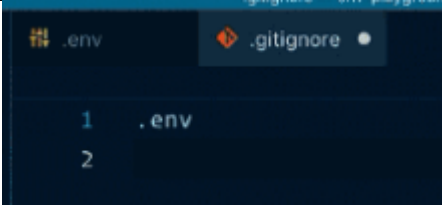


<div>Створюємо файл .env у кореневому каталозі</div> <div><div><div>> controllers</div><div>> db</div><div>> models</div><div>> node_modules</div><div>> public</div><div>> routes</div><div>> validators</div><div>> views</div><div> .env</div><div> app.mjs</div><div> package-lock.json</div><div> package.json</div></div></div>	<div><pre>PORT=3000 MONGODB_URL=mongodb://localhost:27017/ DATABASE_NAME=usersStoredb</pre></div>
<div>Додаємо правило ігнорування у .gitignore</div>	<div></div>
<div>Встановлюємо модуль dotenv</div>	<div><pre>npm install dotenv --save</pre></div>

Створюємо файл конфігурації

```
> bin
✓ config
JS default.mjs
> controllers
> db
> models
> node_modules
> public
> routes
> validators
> views
⚙ .env
JS app.mjs
{} package-lock.json
{} package.json
```

Зчитуємо значення з файлу .env

process.env. змінна

```
import dotenv from 'dotenv'
dotenv.config()

export default Object.freeze({
  databaseName: process.env.DATABASE_NAME,
  databaseUrl: process.env.MONGODB_URL,
  mongoURI: `${process.env.MONGODB_URL}${process.env.DATABASE_NAME}`,
  port: process.env.PORT,
})
```

Створюємо конфігураційний файл для Mongoose
Підключаємо і використовуємо значення

```
import config from
'../config/default.mjs'
```

```
> bin
> config
> controllers
✓ db
JS connectDB.mjs
> models
> node_modules
> public
```

```
import config from '../config/default.mjs'
// Імпортуємо необхідний модуль
import mongoose from 'mongoose'

// Встановлюємо глобальні проміси
mongoose.Promise = global.Promise
// Функція для підключення до MongoDB
export default async function () {
  try {
    await mongoose.connect(config.mongoURI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    console.log('Успішно підключено до MongoDB')
  } catch (err) {
    console.error('Помилка підключення до MongoDB:', err)
  }
}
```

Описуємо схему даних та створюємо модель даних

```
> bin
> config
> controllers
> db
▼ models\user
  JS User.mjs
  JS UsersDBService.mjs
> node_modules
> public
```

```
import mongoose from 'mongoose'
import config from '../../config/default.mjs'

const { Schema } = mongoose

const userSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Name is required'],
    minlength: [3, 'Name must be at least 3 characters long'],
    maxlength: [50, 'Name must be at most 50 characters long'],
    trim: true,
  },
  age: {
    type: Number,
    required: [true, 'Age is required'],
    min: [18, 'Age must be at least 18'],
    max: [120, 'Age must be at most 120'],
    toInt: true,
  },
  email: {
    type: String,
    required: [true, 'Email is required'],
    unique: true,
    trim: true,
    lowercase: true,
  },
  password: {
    type: String,
    required: [true, 'Password is required'],
    minlength: [6, 'Password must be at least 6 characters long'],
    maxlength: [8, 'Password must be at most 10 characters long'],
    validate: {
      validator: function (v) {
        return /^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/ .test(
          v
        )
      },
    },
  },
})
```

```

        message: (props) =>
          'Password must contain at least one letter, one number, and one special character',
        },
      },
    })

userSchema.static.checkDatabaseExists = async () => {
  const databases = await mongoose.connection.listDatabases()
  return databases.databases.some((db) => db.name === config.databaseName)
}

userSchema.static.checkCollectionExists = async function () {
  if (await this.checkDatabaseExists()) {
    const collections = await mongoose.connection.db
      .listCollections({ name: 'users' })
      .toArray()
    return collections.length > 0
  }
  return false
}

const User = mongoose.model('User', userSchema)
export default User

```

Створюємо допоміжний клас UsersDBService для операцій з базою даних

```

> bin
> config
> controllers
> db
▼ models\user
  JS User.mjs
  JS UsersDBService.mjs
> node_modules
> public

```

```

import User from './User.mjs'
import mongoose from 'mongoose'

class UsersDBService {
  static async getList() {
    try {
      const exists = await User.checkCollectionExists
      if (exists) {
        const data = await mongoose.model(collectionName).find().exec()
        return data
      }
    }

    return (await User.find({})) ?? []
  }
}

```

```

    } catch (error) {
      return []
    }
  }

  static async create(data) {
    const user = new User(data)
    return await user.save()
  }

  static async getById(id) {
    return await User.findById(id)
  }

  static async update(id, data) {
    return await User.findByIdAndUpdate(id, data, {
      new: true,
      runValidators: true,
    })
  }

  static async deleteById(id) {
    return await User.findByIdAndDelete(id)
  }
}

export default UsersDBService

```

Створюємо файл валідатора

```

> bin
> config
> controllers
> db
> models \user
> node_modules
> public
> routes
v validators
JS userValidator.mjs
> views

```

```

import { body } from 'express-validator'

class UserValidator {
  static userValidationRules = [
    body('email').isEmail().withMessage('Invalid email address'),
    body('password')
      .isLength({ min: 6 })
      .withMessage('Password must be at least 6 characters long'),
    body('name').not().isEmpty().withMessage('Name is required'),
  ]
  static userSchema = {

```

```

email: {
  isEmail: {
    errorMessage: 'Invalid email address',
  },
  normalizeEmail: true, // Нормалізує email
},
password: {
  isLength: {
    options: { min: 6 },
    errorMessage: 'Password must be at least 6 characters long',
  },
},
age: {
  isInt: {
    options: { min: 18, max: 120 },
    errorMessage: 'Age must be between 18 and 120',
  },
  toInt: true,
},
name: {
  notEmpty: {
    errorMessage: 'Name is required',
  },
  isLength: {
    options: { min: 3 },
    errorMessage: 'Username must be at least 3 characters long',
  },
  trim: true, // Видаляє пробіли на початку і в кінці
  escape: true, // Екранує HTML символи
},
}
}

export default UserValidator

```

Створюємо файл контролера

```

import UsersDBService from '../models/user/UsersDBService.mjs'
import { validationResult } from 'express-validator'

```

> bin
> config
v controllers
JS userController.mjs
> db
v models\user

```
class UserController {
  static async usersList(req, res) {
    try {
      const dataList = await UsersDBService.getList()
      console.log('=====dataList')
      console.log(dataList)

      res.render('usersList', {
        users: dataList,
      })
    } catch (err) {
      res.status(500).json({ error: err.message })
    }
  }

  static async registerForm(req, res) {
    try {
      const id = req.params.id
      let user = null
      if (id) {
        //отримати об'єкт за id
        user = await UsersDBService.getById(id)
      }
      //відредерити сторінку з формою
      res.render('register', {
        errors: [],
        data: user,
      })
    } catch (err) {
      res.status(500).json({ error: err.message })
    }
  }

  static async registerUser(req, res) {
    // Якщо валідація пройшла успішно, виконуємо логіку реєстрації
    const errors = validationResult(req)
    const data = req.body
```

```
if (!errors.isEmpty()) {
  if (req.params.id) data.id = req.params.id
  return res.status(400).render('register', {
    errors: errors.array(),
    data,
  })
}

try {
  const { email, age, password, name } = req.body
  console.log('====>>> req.body')
  console.log(req.body)

  if (req.params.id) {
    // Оновлюємо дані про користувача в базі даних
    await UsersDBService.update(req.params.id, {
      email,
      age,
      password,
      name,
    })
  } else {
    // Додаємо користувача в базу даних
    await UsersDBService.create({ email, age, password, name })
  }

  res.redirect('/users')
} catch (err) {
  res.status(500).render('register', {
    errors: [{ msg: err.message }],
    data,
  })
}

static async deleteUser(req, res) {
  try {
    await UsersDBService.deleteById(req.body.id)
```



```

    res.json({ success: true })
  } catch (error) {
    res.status(500).json({ success: false, message: 'Failed to delete user' })
  }
}
}

export default UserController

```

Створюємо головний файл роутера

```

> bin
> config
> controllers
> db
> models\user
> node_modules
> public
v routes
JS index.mjs
JS users.mjs

```

```

import { Router } from 'express'
const router = Router()

router.get('/', (req, res) => {
  res.render('index', { title: 'Express' })
})

export default router

```

Створюємо файл роутера для користувачів

```

> bin
> config
> controllers
> db
> models
> node_modules
> public
v routes
JS index.mjs
JS users.mjs
v validators

```

```

import express from 'express'
import UserController from '../controllers/userController.mjs'
import UserValidator from '../validators/userValidator.mjs'
import { checkSchema } from 'express-validator'

const router = express.Router()

router.get('/', UserController.usersList)

router.get('/register/:id?', UserController.registerForm)

router.post(
  '/register/:id?',
  checkSchema(UserValidator.userSchema),
  UserController.registerUser
)

router.delete('/', UserController.deleteUser)

```

	<pre>export default router</pre>
<div>Файли шаблонів відображення</div> <div><div><div>> public</div><div>> routes</div><div>> validators</div><div>> views</div><div><div>> sections</div><div>> footer.ejs</div><div>> header.ejs</div><div>> error.ejs</div><div>> index.ejs</div><div>> register.ejs</div></div></div></div>	<pre><header class="main-menu"> <nav> Home Users list </nav> <h1><%= title%></h1> </header></pre>
<div><div>> public</div><div>> routes</div><div>> validators</div><div>> views</div><div><div>> sections</div><div>> footer.ejs</div><div>> header.ejs</div><div>> error.ejs</div><div>> index.ejs</div><div>> register.ejs</div></div></div>	<pre><footer> <p>MyCorp - Copyright © 2024</p> </footer></pre>

- > public
- > routes
- > validators
- ▼ views
 - ▼ sections
 - <> footer.ejs
 - <> header.ejs
 - <> error.ejs
 - <> index.ejs
 - <> register.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet" href="/stylesheets/style.css" />
    <link rel="stylesheet" href="/stylesheets/main.css" />
  </head>
  <body>
    <%- include('sections/header', { title: 'User form' }) %>

    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

- > routes
- > validators
- ▼ views
 - ▼ sections
 - <> footer.ejs
 - <> header.ejs
 - <> error.ejs
 - <> index.ejs
 - <> register.ejs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Register page</title>
    <link rel="stylesheet" href="/stylesheets/main.css" />
  </head>
  <body>
    <%- include('sections/header', { title: 'User form' }) %>
    <hr />
    <% if (errors?.length > 0) { %>
      <ul>
        <% errors.forEach(function(error) { %>
          <li><%= error.msg %></li>
        <% }) %>
      </ul>
    <hr />
    <% } %>
    <form
      action="/users/register<%= data?.id ? '/' + data.id : '' %>"
      method="post"
```

```
>
  <input
    type="text"
    name="name"
    placeholder="Name"
    value="<%= data ? data.name : ' ' %>"
  />
  <input
    type="number"
    name="age"
    placeholder="age"
    value="<%= data ? data.age : ' ' %>"
  />
  <input
    type="email"
    name="email"
    placeholder="Email"
    value="<%= data ? data.email : ' ' %>"
  />
  <input
    type="password"
    name="password"
    placeholder="Password"
    value="<%= data ? data.password : ' ' %>"
  />

  <button type="submit">Register</button>
</form>
</body>
</html>
```

Головний файл

```
import express from 'express'
import path from 'path'
import cookieParser from 'cookie-parser'
import logger from 'morgan'
import { fileURLToPath } from 'url'
// --- routers ---
```

- > public
- > routes
- > validators
- > views
- ⚙ .env
- JS app.mjs**
- { } package-lock.json
- { } package.json

```
import indexRouter from './routes/index.mjs'
import usersRouter from './routes/users.mjs'

import connectDB from './db/connectDB.mjs'

const app = express()
const __filename = fileURLToPath(import.meta.url) // get the resolved path to the file
const __dirname = path.dirname(__filename) // get the name of the directory

connectDB()
app.set('views', path.join(__dirname, 'views'))
app.set('view engine', 'ejs')

app.use(logger('dev'))
app.use(express.json())
app.use(express.urlencoded({ extended: false }))
app.use(cookieParser())
app.use(express.static(path.join(__dirname, 'public')))

app.use('/', indexRouter)
app.use('/users', usersRouter)

// catch 404 and forward to error handler
app.use((req, res, next) => {
  const err = new Error('Not Found')
  err.status = 404
  next(err)
})

// error handler
app.use((err, req, res, next) => {
  // set locals, only providing error in development
  res.locals.message = err.message
  res.locals.error = req.app.get('env') === 'development' ? err : {}

  // render the error page
  res.status(err.status || 500)
  res.render('error')
```