

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №1**

з дисципліни “Основи web-програмування”

тема “**Основи JavaScript**”

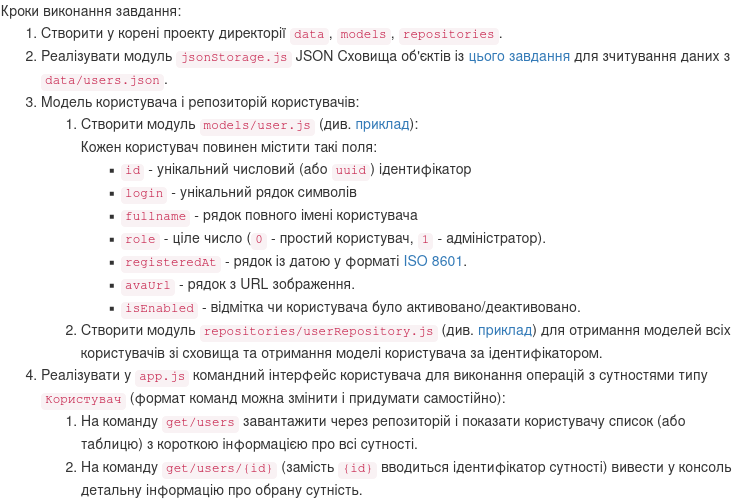
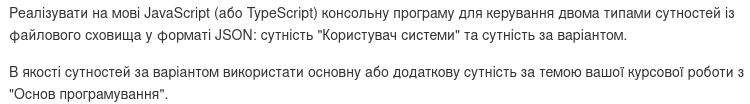
|  |  |  |
| --- | --- | --- |
| Виконав(ла)  студент(ка) I курсу  групи КП-91  Маховой Олександр Вікторович  (*прізвище, ім’я, по батькові*)  варіант № |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

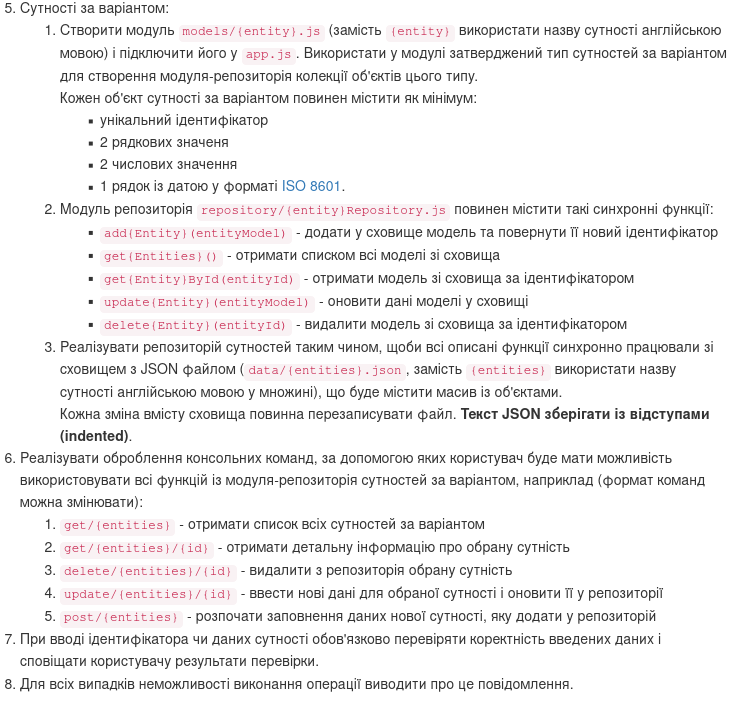
Київ 2020

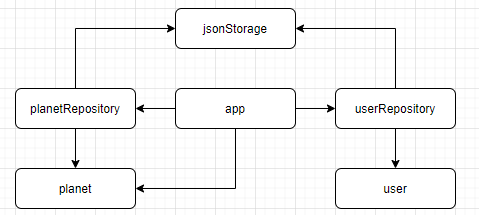
**Мета роботи**

Познайомитись з мовою програмування JavaScript та середовищем Node.js.  
Навчитись створювати модулі, синхронно працювати із файловими потоками та обробляти JSON-текст.

**Постановка завдання**

****

**Діаграма залежності модулів**

****

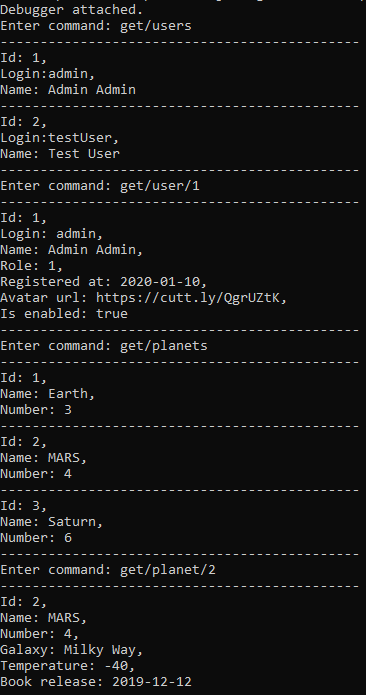
**Тексти коду програми**

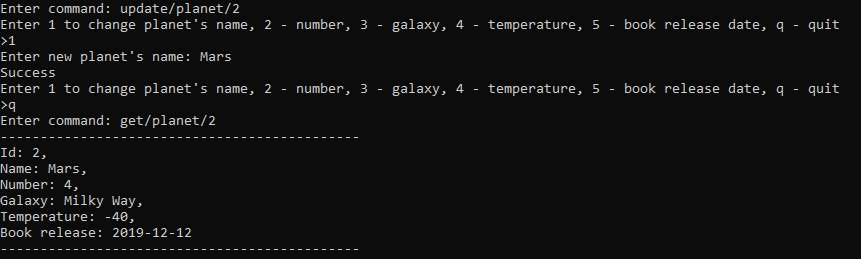
|  |
| --- |
| **app.js** |
| const readlineSync = require('readline-sync');  const UserRepository = require('./repositories/userRepository');  const PlanetRepository = require('./repositories/planetRepository');  const Planet = require('./models/planet');  const parse = require('date-fns/parse');  const userRepository = new UserRepository('./data/users.json');  const planetRepository = new PlanetRepository('./data/planets.json');  while (1)  {  const input = readlineSync.question('Enter command: ').trim().toLowerCase();  const parts = input.split("/");  const command = parts[0] + "/" + parts[1];  switch (command) {  case "get/users":  const users = userRepository.getUsers();  separator();  for (const user of users) {  console.log(`Id: ${user.id}, \nLogin:${user.login}, \nName: ${user.fullname}`);  separator();  }  break;  case "get/user":  if (!isNaN(parts[2]) && Number.isInteger(parseFloat(parts[2]))) {  const userId = parseInt(parts[2]);  const user = userRepository.getUserById(userId);  if (!user) {  console.log(`Error: user with id ${userId} not found.`);  }  else {  separator();  console.log(`Id: ${user.id}, \nLogin: ${user.login}, \nName: ${user.fullname}, \nRole: ${user.role}, \nRegistered at: ${user.registeredAt}, \nAvatar url: ${user.avaUrl}, \nIs enabled: ${user.isEnabled}`);  separator();  }  }  else console.log(`Error: id must be an integer`);  break;  case "get/planets":  const planets = planetRepository.getPlanets();  separator();  for (const planet of planets) {  console.log(`Id: ${planet.id}, \nName: ${planet.name}, \nNumber: ${planet.number}`);  separator();  }  break;  case "get/planet":  if (!isNaN(parts[2]) && Number.isInteger(parseFloat(parts[2]))) {  const planetId = parseInt(parts[2]);  const planet = planetRepository.getPlanetById(planetId);  if (!planet) {  console.log(`Error: planet with id ${planetId} not found.`);  }  else {  separator();  console.log(`Id: ${planet.id}, \nName: ${planet.name}, \nNumber: ${planet.number}, \nGalaxy: ${planet.galaxy}, \nTemperature: ${planet.temperature}, \nBook release: ${planet.book\_release}`);  separator();  }  }  else console.log(`Error: id must be an integer`);  break;  case "delete/planet":  if (!isNaN(parts[2]) && Number.isInteger(parseFloat(parts[2]))) {  const planetId = parseInt(parts[2]);  if (planetRepository.deletePlanet(planetId))  console.log("Deleted succesfully");  else console.log(`Error: planet with id ${planetId} not found.`);  }  else console.log(`Error: id must be an integer`);  break;  case "update/planet":  if (!isNaN(parts[2]) && Number.isInteger(parseFloat(parts[2]))) {  const planetId = parseInt(parts[2]);  const planet = planetRepository.getPlanetById(planetId);  if (!planet) {  console.log(`Error: planet with id ${planetId} not found.`);  break;  }  while (1) {  const input = readlineSync.question("Enter 1 to change planet's name, 2 - number, 3 - galaxy, 4 - temperature, 5 - book release date, q - quit\n>");  if (input === "q") break;  switch (input) {  case "1":  const newName = readlineSync.question("Enter new planet's name: ");  if (newName.length !== 0)  planet.name = newName;  else {  console.log("Error: invalid input. Name field can't be empty");  break;  }  if (planetRepository.updatePlanet(planet))  console.log("Success");  else  console.log("Error");  break;  case "2":  const newNumber = readlineSync.question("Enter new planet's number: ");  if (newNumber > 0)  planet.number = newNumber;  else {  console.log("Error: invalid input. Number field must be numeric and can't be less than zero");  break;  }  if (planetRepository.updatePlanet(planet))  console.log("Success");  else  console.log("Error");  break;  case "3":  const newGalaxy = readlineSync.question("Enter new planet's galaxy name: ");  if (newGalaxy.length !== 0)  planet.galaxy = newGalaxy;  else {  console.log("Error: invalid input. Galaxy field can't be empty");  break;  }  if (planetRepository.updatePlanet(planet))  console.log("Success");  else  console.log("Error");  break;  case "4":  const newTemp = readlineSync.question("Enter new planet's temperature (in °C): ");  if (!isNaN(newTemp))  planet.temperature = newTemp;  else {  console.log("Error: invalid input. Temperature field must be numeric");  break;  }  if (planetRepository.updatePlanet(planet))  console.log("Success");  else  console.log("Error");  break;  case "5":  const input = readlineSync.question("Enter new planet's book release date: ");  let release = parse(input, 'yyyy-MM-dd', new Date());  if (release)  planet.book\_release = release.toISOString();  else {  console.log("Error: invalid input.");  break;  }  if (planetRepository.updatePlanet(planet))  console.log("Success");  else  console.log("Error");  break;  default:  console.log("Not supported command");  }  }  }  else console.log(`Error: id must be an integer`);  break;  case "post/planet":  const planet = new Planet();  while (1) {  const newName = readlineSync.question("Enter new planet's name: ");  if (newName.length !== 0) {  planet.name = newName;  break;  }  else  console.log("Error: invalid input. Name field can't be empty");  }  while (1) {  const newNumber = readlineSync.question("Enter new planet's number: ");  if (newNumber > 0) {  planet.number = newNumber;  break;  }  else  console.log("Error: invalid input. Number field must be numeric and can't be less than zero");  }  while (1) {  const newGalaxy = readlineSync.question("Enter new planet's galaxy name: ");  if (newGalaxy.length !== 0) {  planet.galaxy = newGalaxy;  break;  }  else  console.log("Error: invalid input. Galaxy field can't be empty");  }  while (1) {  const newTemp = readlineSync.question("Enter new planet's temperature: ");  if (!isNaN(newTemp)) {  planet.temperature = newTemp;  break;  }  else {  console.log("Error: invalid input. Temperature field must be numeric");  }  }  while (1) {  const input = readlineSync.question("Enter new planet's book release date (yyyy-MM-dd): ");  let newDate = parse(input, 'yyyy-MM-dd', new Date());  if (newDate) {  planet.book\_release = newDate.toISOString();  break;  }  else {  console.log("Error: invalid input.");  }  }  let id = planetRepository.addPlanet(planet);  console.log(`Success. New planet's id: ${id}`);  break;  default:  console.log("Not supported command");  }  }  function separator() {  console.log("---------------------------------------------");  } |
| **./models/planet** |
| class Planet {  constructor(id, name, number, galaxy, temperature, book\_release) {  this.id = id; // number  this.name = name; // string  this.number = number; // number  this.galaxy = galaxy; // string  this.temperature = temperature; // number  this.book\_release = book\_release; // date  }  };  module.exports = Planet; |
| **./models/user** |
| class User {  constructor(id, login, fullname, role = 0, registeredAt = null, avaUrl = null, isEnabled = true) {  this.id = id; // number  this.login = login; // string  this.fullname = fullname; // string  this.role = role; // int (0/1)  this.registeredAt = registeredAt; // date  this.avaUrl = avaUrl; // string  this.isEnabled = isEnabled; // boolean  }  };  module.exports = User; |
| **./repositories/userRepository** |
| const User = require('../models/user');  const JsonStorage = require('./jsonStorage');  class UserRepository {  constructor(filePath) {  this.storage = new JsonStorage(filePath);  }  getUsers() {  const items = this.storage.readItems();  const users = [];  for (const item of items) {  users.push(new User(item.id, item.login, item.fullname , item.role, item.registeredAt, item.avaUrl, item.isEnabled));  }  return users;  }  getUserById(userId) {  const items = this.storage.readItems();  for (const item of items) {  if (item.id === userId) {  return new User(item.id, item.login, item.fullname , item.role, item.registeredAt, item.avaUrl, item.isEnabled);  }  }  return null;  }  };  module.exports = UserRepository; |
| **./repositories/planetRepository** |
| const Planet = require('./../models/planet');  const JsonStorage = require('./jsonStorage');  class PlanetRepository {  constructor(filePath) {  this.storage = new JsonStorage(filePath);  }  addPlanet(planet) {  const planets = this.getPlanets();  planet.id = this.storage.nextId();  planets.push(planet);  this.storage.incrementNextId();  this.storage.writeItems(planets);  return planet.id;  }  getPlanets() {  const items = this.storage.readItems();  const planets = [];  for (const item of items) {  planets.push(new Planet(item.id, item.name, item.number, item.galaxy, item.temperature, item.book\_release));  }  return planets;  }  getPlanetById(planetId) {  const items = this.storage.readItems();  for (const item of items) {  if (item.id === planetId) {  return new Planet(item.id, item.name, item.number, item.galaxy, item.temperature, item.book\_release);  }  }  return null;  }  updatePlanet(updPlanet) {  const planets = this.getPlanets();  const planet = this.getPlanetById(updPlanet.id);  if (planet !== null) {  const index = planets.findIndex((plnt) => {  return plnt.id === planet.id;  });  planets.splice(index, 1, updPlanet);  this.storage.writeItems(planets);  return true;  }  else return null;  }  deletePlanet(id) {  const planets = this.getPlanets();  const planet = this.getPlanetById(id);  if (planet !== null) {  const index = planets.findIndex((plnt) => {  return plnt.id === planet.id;  });  planets.splice(index, 1);  this.storage.writeItems(planets);  return true;  }  else return null;  }  };  module.exports = PlanetRepository; |
| **jsonStorage.js** |
| const fs = require('fs');  class JsonStorage{  constructor(filePath) {  this.filePath = filePath;  }  nextId() {  return (JSON.parse(fs.readFileSync(this.filePath))).nextId;  }  incrementNextId() {  const file = JSON.parse(fs.readFileSync(this.filePath));  file.nextId++;  fs.writeFileSync(this.filePath, JSON.stringify(file, null, 4, (err) => {  if (err) throw err;  }));  }  readItems() {  return (JSON.parse(fs.readFileSync(this.filePath))).items;  }  writeItems(items) {  const file = (JSON.parse(fs.readFileSync(this.filePath)));  file.items = items;  fs.writeFileSync(this.filePath, JSON.stringify(file, null, 4), (err) => {  if(err) throw err;  });  }  };  module.exports = JsonStorage; |

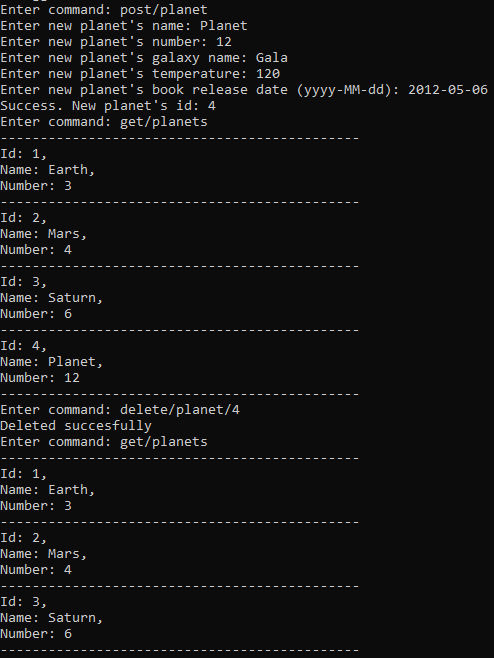
**Вміст json файлів**

|  |
| --- |
| **planets.json** |
| {  "nextId": 4,  "items": [  {  "id": 1,  "name": "Earth",  "number": 3,  "galaxy": "Milky Way",  "temperature": 45,  "book\_release": "1986-10-13"  },  {  "id": 2,  "name": "Mars",  "number": 5,  "galaxy": "Milky Way",  "temperature": 150,  "book\_release": "2019-12-12"  },  {  "id": 3,  "name": "Saturn",  "number": 6,  "galaxy": "Milky Way",  "temperature": -120,  "book\_release": "1999-01-27"  }  ]  } |
| **users.json** |
| {  "nextId": 3,  "items": [  {  "id": 1,  "login": "admin",  "fullname": "Admin Admin",  "role": 1,  "registeredAt": "2020-01-10",  "avaUrl": "https://cutt.ly/QgrUZtK",  "isEnabled": true  },  {  "id": 2,  "login": "testUser",  "fullname": "Test User",  "role": 1,  "registeredAt": "2020-29-09",  "avaUrl": "-",  "isEnabled": false  }  ]  } |

**Приклад результата**

****

****

****

**Висновки**

Виконавши цю лабораторну роботу я познайомився з мовою програмування JavaScript та середовищем Node.js. Навчився створювати модулі, синхронно працювати із файловими потоками, обробляти JSON-текст, та встановлювати сторонні бібліотеки за допомогою утиліти npm.