

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  Климчук Нікіта Олегович  (*прізвище, ім’я, по батькові*)  варіант № 9 |  | 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.

### **Завдання**

Реалізувати на мові JavaScript (або TypeScript) консольну програму для керування двома типами сутностей із файлового сховища у форматі JSON: сутність "Користувач системи" та сутність за варіантом.

В якості сутностей за варіантом використати основну або додаткову сутність за темою вашої курсової роботи з "Основ програмування".

### Вказівки до виконання

Середовище розроблення:

* Встановити найновішу доступну LTS версію [**Node.js (node)**](https://nodejs.org/en/).
* Встановити глобально пакет **ESLint** за допомогою команди:  
  npm install -g eslint (із sudo на Linux).

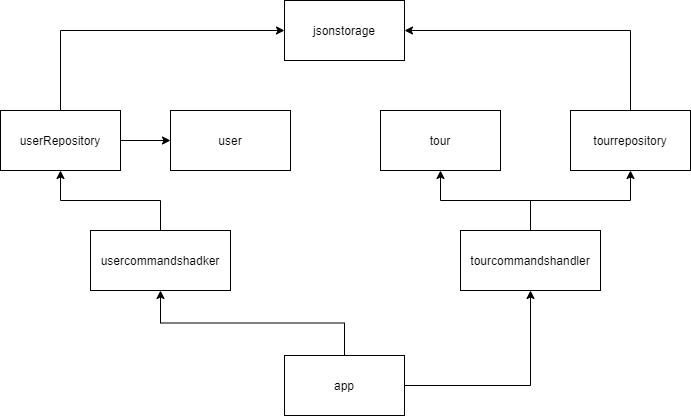
Проект завдання:

* Створити у навчальному репозиторії **webprogbase** директорію проекту: labs/lab1 і перейти до неї у редакторі (IDE).
* Створити у корені проекту файл app.js та .eslintrc.json (див. Додаток).
* Додати у корінь проекту файл .gitignore ([вміст файлу](https://github.com/github/gitignore/blob/master/Node.gitignore))
* Відкрити термінал у кореневій директорії завдання і виконати команду: npm init --yes, яка автоматично створить файл package.json.

Кроки виконання завдання:

1. Створити у корені проекту директорії data, models, repositories.
2. Реалізувати модуль jsonStorage.js JSON Сховища об'єктів із [цього завдання](http://progbase.herokuapp.com/modules/webprogbase/tasks/json_storage) для зчитування даних з data/users.json.
3. Модель користувача і репозиторій користувачів:
   1. Створити модуль models/user.js (див. [приклад](https://docs.google.com/document/d/1-Uz-Y-H6AtDygYpv_OhcOel6atb6clcGaTd04FD58WE/edit?usp=sharing)):  
      Кожен користувач повинен містити такі поля:
      * id - унікальний числовий (або uuid) ідентифікатор
      * login - унікальний рядок символів
      * fullname - рядок повного імені користувача
      * role - ціле число (0 - простий користувач, 1 - адміністратор).
      * registeredAt - рядок із датою у форматі [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601).
      * avaUrl - рядок з URL зображення.
      * isEnabled - відмітка чи користувача було активовано/деактивовано.
   2. Створити модуль repositories/userRepository.js (див. [приклад](https://docs.google.com/document/d/1-Uz-Y-H6AtDygYpv_OhcOel6atb6clcGaTd04FD58WE/edit?usp=sharing)) для отримання моделей всіх користувачів зі сховища та отримання моделі користувача за ідентифікатором.
4. Реалізувати у app.js командний інтерфейс користувача для виконання операцій з сутностями типу Користувач (формат команд можна змінити і придумати самостійно):
   1. На команду get/users завантажити через репозиторій і показати користувачу список (або таблицю) з короткою інформацією про всі сутності.
   2. На команду get/users/{id} (замість {id} вводиться ідентифікатор сутності) вивести у консоль детальну інформацію про обрану сутність.
5. Сутності за варіантом:
   1. Створити модуль models/{entity}.js (замість {entity} використати назву сутності англійською мовою) і підключити його у app.js. Використати у модулі затверджений тип сутностей за варіантом для створення модуля-репозиторія колекції об'єктів цього типу.  
      Кожен об'єкт сутності за варіантом повинен містити як мінімум:
      * унікальний ідентифікатор
      * 2 рядкових значеня
      * 2 числових значення
      * 1 рядок із датою у форматі [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601).
   2. Модуль репозиторія repository/{entity}Repository.js повинен містити такі синхронні функції:
      * add{Entity}(entityModel) - додати у сховище модель та повернути її новий ідентифікатор
      * get{Entities}() - отримати списком всі моделі зі сховища
      * get{Entity}ById(entityId) - отримати модель зі сховища за ідентифікатором
      * update{Entity}(entityModel) - оновити дані моделі у сховищі
      * delete{Entity}(entityId) - видалити модель зі сховища за ідентифікатором
   3. Реалізувати репозиторій сутностей таким чином, щоби всі описані функції синхронно працювали зі сховищем з JSON файлом (data/{entities}.json, замість {entities} використати назву сутності англійською мовою у множині), що буде містити масив із об'єктами.  
      Кожна зміна вмісту сховища повинна перезаписувати файл. **Текст JSON зберігати із відступами (indented)**.
6. Реалізувати оброблення консольних команд, за допомогою яких користувач буде мати можливість використовувати всі функцій із модуля-репозиторія сутностей за варіантом, наприклад (формат команд можна змінювати):
   1. get/{entities} - отримати список всіх сутностей за варіантом
   2. get/{entities}/{id} - отримати детальну інформацію про обрану сутність
   3. delete/{entities}/{id} - видалити з репозиторія обрану сутність
   4. update/{entities}/{id} - ввести нові дані для обраної сутності і оновити її у репозиторії
   5. post/{entities} - розпочати заповнення даних нової сутності, яку додати у репозиторій
7. При вводі ідентифікатора чи даних сутності обов'язково перевіряти коректність введених даних і сповіщати користувачу результати перевірки.
8. Для всіх випадків неможливості виконання операції виводити про це повідомлення.

**Діаграма залежності модулів**



**Тексти коду програм**

|  |
| --- |
| **Tour.js** |
| class Tour {  constructor(id = -1, name = null, country = null, price = null, maxTouristsCount = null, startDate = null) {  this.id = id;  this.name = name;  this.country = country;  this.price = price;  this.maxTouristsCount = maxTouristsCount;  this.startDate = startDate;  }  }  module.exports = Tour; |

|  |
| --- |
| **User.js** |
| class User {  constructor(id, login, fullname, role = 0, registeredAt = null, avaUrl = null, isEnabled = false) {  this.id = id; // number  this.login = login; // string  this.fullname = fullname; // string  this.role = role; //int  this.registeredAt = registeredAt; //date  this.avaUrl = avaUrl; //string  this.isEnabled = isEnabled; //bool  }  };  module.exports = User; |

|  |
| --- |
| **JsonStorage.js** |
| const fs = require('fs');  class JsonStorage {    constructor(filePath) {  this.filePath\_ = filePath;  }  get nextId() {  const id = (JSON.parse(fs.readFileSync(this.filePath\_))).nextId;  return id;  }  incrementNextId() {  const file = (JSON.parse(fs.readFileSync(this.filePath\_)));  file.nextId = file.nextId + 1;  fs.writeFileSync(this.filePath\_, JSON.stringify(file));  }  readItems() {  const items = (JSON.parse(fs.readFileSync(this.filePath\_))).items;  return items;  }  writeItems(items) {  const file = (JSON.parse(fs.readFileSync(this.filePath\_)));  file.items = items;  fs.writeFileSync(this.filePath\_, JSON.stringify(file, null, 4));  }  };  module.exports = JsonStorage; |

|  |
| --- |
| **TourRepository.js** |
| const Tour = require('./../Models/Tour');  const JsonStorage = require('./JsonStorage')  class TourRepository{  constructor(filePath) {  this.tourStorage\_ = new JsonStorage(filePath);  }  AddTour(tour) {  const tours = this.GetTours();  tour.id = this.tourStorage\_.nextId;  this.tourStorage\_.incrementNextId();  tours.push(tour);  this.tourStorage\_.writeItems(tours);  }  DeleteTour(id) {  const tours = this.GetTours();  const tour = this.GetTourById(id);    if (tour != null) {  const index = tours.findIndex((x) => {  return x.id == id;  });  tours.splice(index, 1);  this.tourStorage\_.writeItems(tours);  }  else {  throw new Error('there is no tour like this');  }      }  GetTours() {  const items = this.tourStorage\_.readItems();  const tours = [];  for (const item of items) {  tours.push(new Tour(item.id, item.name, item.country, item.price, item.maxTouristsCount, item.startDate));  }  return tours;  }  GetTourById(id) {  const items = this.tourStorage\_.readItems();  for (const item of items) {  if (item.id == id) {  return new Tour(item.id, item.name, item.country, item.price, item.maxTouristsCount, item.startDate);  }  }  return null;  }  UpdateTour(tour) {  const tours = this.GetTours();  const tour\_ = this.GetTourById(tour.id);  if (tour\_ != null) {  const index = tours.findIndex((x) => {  return x.id == tour.id;  });  tours.splice(index, 1);  tours.push(tour);  this.tourStorage\_.writeItems(tours);  }    }  }  module.exports = TourRepository; |

|  |
| --- |
| **UserRepository.js** |
| const User = require('./../Models/User');  const JsonStorage = require('./JsonStorage')  class UserRepository {  constructor(filePath) {  this.userStorage\_ = new JsonStorage(filePath);  }  getUsers() {  const items = this.userStorage\_.readItems();  const users = [];  for (const item of items) {  users.push(new User(item.id, item.login, item.fullname));  }  return users;  }  getUserById(id) {  const items = this.userStorage\_.readItems();  for (const item of items) {  if (item.id == id) {  return new User(item.id, item.login, item.fullname);  }  }  return null;  }  };  module.exports = UserRepository; |

|  |
| --- |
| **UserCommandsHandler.js** |
| const UserStorage = require('./../Repositories/UserRepository')  const readlineSync = require('readline-sync')  const userRepo = new UserStorage('./Data/Users.json')  const commandsHandler = [];  commandsHandler['get\_users'] = () => {  const users = userRepo.getUsers();  for (const user of users) {  console.log(user.id, user.login, user.fullname);  }  }  commandsHandler['get\_userbyid'] = () => {  const input = readlineSync.question('Enter id: ');  const user = userRepo.getUserById(input);  if (user != undefined) {  console.log(user.id, user.login, user.fullname, user.avaUrl, user.role, user.registeredAt, user.isEnabled);  }  else {  console.log('incorrect index...');  }  }  commandsHandler['exit'] = () => {  process.exit();  }  module.exports = commandsHandler; |

|  |
| --- |
| **TourCommandsHandler.js** |
| const TourRepository = require('./../Repositories/TourRepository');  const Tour = require('./../Models/Tour')  const readlineSync = require('readline-sync')  const tourRepo = new TourRepository('./Data/Tours.json');  const commandsHandler = [];  commandsHandler['get\_tours'] = () => {  const tours = tourRepo.GetTours();  for (const tour of tours) {  console.log(tour.id, tour.name, tour.country);  }  };  commandsHandler['get\_tourbyid'] = () => {  const input = readlineSync.question('Enter id: ');  const tour = tourRepo.GetTourById(input);  if (tour != null) {  console.log(tour.id, tour.name, tour.country, tour.price, tour.maxTouristsCount, tour.startDate)  }  else {  console.log('incorrect index...');  }  }  commandsHandler['delete\_tour'] = () => {  const input = readlineSync.question('Enter id: ');  try {  tourRepo.DeleteTour(input);  console.log('Deleted');  }  catch {  console.log('Nothing changed...');  }    }  commandsHandler['update\_tour'] = () => {  const input = readlineSync.question('Enter id: ');  const tour = tourRepo.GetTourById(input);  if (tour != null) {  console.log('Tour: ', tour.id, tour.name, tour.country, tour.price, tour.maxTouristsCount, tour.startDate);  const name = readlineSync.question('Enter tour name: ');  tour.name = name;  const country = readlineSync.question('Enter tour country: ');  tour.country = country;  const price = readlineSync.question('Enter tour price: ');  const numPrice = Number(price);  if (numPrice != NaN) {  tour.price = numPrice;  const maxTouristsCount = readlineSync.question('Enter tour maxTouristsCount: ');  const numMaxTouristsCount = Number(maxTouristsCount);  if (numMaxTouristsCount != NaN) {  tour.maxTouristsCount = numMaxTouristsCount;  const startDate = readlineSync.question('Enter tour startDate: ');  try {  const dateStartDate = new Date(startDate);  const strStartDate = dateStartDate.toISOString();  tour.startDate = strStartDate;  tourRepo.UpdateTour(tour);  console.log('the value is changed');  }  catch (err){  console.log('incorrect startDate format');  console.log('nothing changed...');  }  }  else {  console.log('incorrect maxTouristsCount format');  console.log('nothing changed...');  }  }  else {  console.log('incorrect price format');  console.log('nothing changed...');  }  }  else {  console.log('incorrect index...');  }  }  commandsHandler['post\_tour'] = () => {  tour = new Tour();  const name = readlineSync.question('Enter tour name: ');  tour.name = name;  const country = readlineSync.question('Enter tour country: ');  tour.country = country;  const price = readlineSync.question('Enter tour price: ');  const numPrice = Number(price);  if (numPrice != NaN) {  tour.price = numPrice;  const maxTouristsCount = readlineSync.question('Enter tour maxTouristsCount: ');  const numMaxTouristsCount = Number(maxTouristsCount);  if (numMaxTouristsCount != NaN) {  tour.maxTouristsCount = numMaxTouristsCount;  const startDate = readlineSync.question('Enter tour startDate: ');  try {  const dateStartDate = new Date(startDate);  const strStartDate = dateStartDate.toISOString();  tour.startDate = strStartDate;  tourRepo.AddTour(tour);  console.log('the value is added');  }  catch (err) {  console.log('incorrect startDate format');  console.log('nothing changed...');  }  }  else {  console.log('incorrect maxTouristsCount format');  console.log('nothing changed...');  }  }  else {  console.log('incorrect price format');  console.log('nothing changed...');  }  }  module.exports = commandsHandler; |

|  |
| --- |
| **app.js** |
| 'start point'  const readlineSync = require('readline-sync')  const UserCommandsHandler = require('./CommandHandlers/UserCommandsHandler')  const TourCommandsHandler = require('./CommandHandlers/TourCommandsHandler')  while (true) {  const input = readlineSync.question('Enter command: ').trim().toLowerCase();  try {  UserCommandsHandler[input]();  }  catch (ex) {  try {  TourCommandsHandler[input]();  }  catch (ex) {  console.log('Wrong command');  }  }  } |

**Вміст json файлів**

|  |
| --- |
| **Users.json** |
| {  "nextId": 3,  "items": [  {  "id": 1,  "login": "admin",  "fullname": "Admin Admin"  },  {  "id": 2,  "login": "testUser",  "fullname": "Test User"  }  ]  } |

|  |
| --- |
| **Tours.json** |
| {  "nextId": 7,  "items": [  {  "id": 6,  "name": "dasasdfas",  "country": "awswq",  "price": 123123,  "maxTouristsCount": 3425,  "startDate": "1995-12-17T01:24:00.000Z"  }  ]  } |

**Висновок**

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