

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”

|  |  |  |
| --- | --- | --- |
| Виконала  студентка II курсу  групи КП-93  Сивоконь Юлія Владиславівна  (*прізвище, ім’я, по батькові*)  варіант №16 |  | Перевірив  16 жовтня 2020 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**

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".

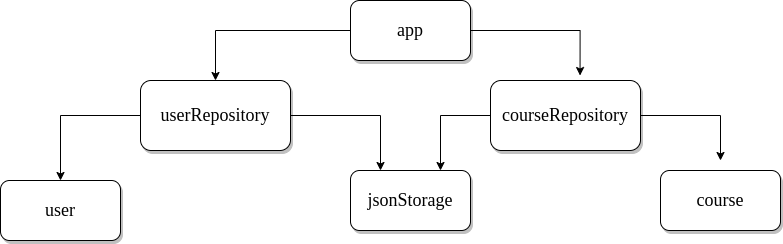
Проект завдання:

* 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
* 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 (**див. Додатки**) 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 залежностей**

****

**Тексти коду програми**

|  |
| --- |
| **course.js** |
| class Course {  constructor(id, name, teacher, hours, credits, beginningDate) {  this.id = id // number  this.name=name  this.teacher=teacher  this.hours=hours  this.credits=credits  this.beginingDate=beginningDate  }  };  module.exports = Course; |

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

|  |
| --- |
| **jsonStorage.js** |
| const fs = require('fs')  class JsonStorage {  constructor(filePath) {  this.filePath = filePath  this.obj = {}  }  get nextId() {  return this.readItems().nextId  }  incrementNextId() {  return ++this.obj.nextId  }  readItems() {  let jsonText = fs.readFileSync(this.filePath)  let module = JSON.parse(jsonText);  this.obj = module  return this.obj  }  writeItems(items) {  let newModel = {}  newModel.nextId = ++this.obj.nextId  newModel.items = items  const text = JSON.stringify(newModel, null, 4)  fs.writeFileSync(this.filePath, text)  }  }  module.exports = JsonStorage |

|  |
| --- |
| **courseRepository.js** |
| const Course = require('../models/course')  const JsonStorage = require('../jsonStorage')  class courseRepository {  constructor(filePath = '../data/courses.json') {  this.storage = new JsonStorage(filePath)  this.nextId = this.storage.nextId  }  getCourses() {  const obj = this.storage.readItems()  this.nextId = obj.nextId  this.items = obj.items  let coursesArr = obj.items  return coursesArr  }  getCourseById(courseId) {  const coursesArr = this.getCourses()  const theCourse=coursesArr.find(x => x.id === courseId)  return new Course(theCourse.id,theCourse.name, theCourse.teacher, theCourse.hours, theCourse.credits, theCourse.beginingDate)  }  addCourse(theCourse) {  theCourse.id = this.nextId  theCourse=new Course(theCourse.id,theCourse.name, theCourse.teacher, theCourse.hours, theCourse.credits, theCourse.beginingDate)  let coursesArr = this.getCourses()  coursesArr.push(theCourse)  this.storage.writeItems(coursesArr)  return this.nextId  }  updateCourse(theCourse) {  let index = this.items.findIndex(x => x.id === theCourse.id)  this.items[index] = new Course(theCourse.id,theCourse.name, theCourse.teacher, theCourse.hours, theCourse.credits, theCourse.beginingDate)  this.storage.writeItems(this.items)  }  deleteCourse(courseId) {  this.getCourses()  let newArr = this.items.filter(x => x.id !== courseId)  this.storage.writeItems(newArr)  }  }  module.exports = courseRepository; |

|  |
| --- |
| **UserRepository.js** |
| const User = require('../models/user')  const JsonStorage = require('../jsonStorage')  class UserRepository {  constructor(filePath = '../data/users.json') {  this.storage = new JsonStorage(filePath)  this.nextId = this.storage.nextId  }  getUsers() {  //let user=new User  const obj = this.storage.readItems()  this.nextId = obj.nextId  this.items = obj.items  let usersArr = obj.items  return usersArr  // console.log(usersArr);  }  getUserById(userId) {  const usersArr = this.getUsers()  const theUser = usersArr.find(x => x.id === userId)  return new User(theUser.id, theUser.login, theUser.fullname, theUser.role, theUser.registeredAt, theUser.avaUrl, theUser.isEnabled)  }  addUser(theUser) {  theUser.id = this.nextId  theUser=new User(theUser.id, theUser.login, theUser.fullname, theUser.role, theUser.registeredAt, theUser.avaUrl, theUser.isEnabled)  let usersArr = this.getUsers()  usersArr.push(theUser)  this.storage.writeItems(usersArr)  return this.nextId  }  updateUser(theUser) {  this.storage.readItems()  let index = this.items.findIndex(x => x.id === theUser.id)  this.items[index] = new User(theUser.id, theUser.login, theUser.fullname, theUser.avaUrl, theUser.role, theUser.registeredAt, theUser.isEnabled)  this.storage.writeItems(this.items)  }  deleteUser(userId) {  this.getUsers()  let newArr = this.items.filter(x => x.id !== userId)  this.storage.writeItems(newArr)  }  }  module.exports = UserRepository; |

|  |
| --- |
| **app.js** |
| const colors = require('colors')  const courseRepository = require('./repositories/courseRepository.js')  const userRepository = require('./repositories/userRepository.js')  const readLineSync = require('readline-sync')  let users = new userRepository('./data/users.json')  let courses = new courseRepository('./data/course.json')  let flag = 1  while (flag) {  console.log('\nEnter the command (entities: courses & users):\n'.blue.bold)  console.log('1. get/{entities}'.green)  console.log('2. get/{entities}/{id}'.green)  console.log('3. delete/{entities}/{id}'.green)  console.log('4. update/{entities}/{id}'.green)  console.log('5. post/{entities}'.green)  console.log('6. exit\n'.green)  const com = readLineSync.question('Your command: '.blue)  if (com === 'get/users') {  console.log('\nThe list of the users\n')  let usersArr = users.getUsers()  usersArr.forEach(x => {  for (let field in x) {  console.log(`${field}: ${x[field]}`)  }  console.log('\n')  })  }  else if (com === 'get/courses') {  console.log('\nThe list of the courses\n')  let coursesArr = courses.getCourses()  coursesArr.forEach(x => {  for (let field in x) {  console.log(`${field}: ${x[field]}`)  }  console.log('\n')  })  }  else if (com.includes('get/users/')) {  const id = Number(com.split('/')[2])  try {  const obj = users.getUserById(id)  for (let field in obj) {  console.log(`${field}: ${obj[field]}`)  }  }  catch (e) {  console.log('\nId is not correct'.red)  }  }  else if (com.includes('get/courses/')) {  const id = Number(com.split('/')[2])  try {  const obj = courses.getCourseById(id)  for (let field in obj) {  console.log(`${field}: ${obj[field]}`)  }  }  catch (e) {  console.log('\nId is not correct'.red)  }  }  else if (com.includes('delete/courses/')) {  try {  const id = Number(com.split('/')[2])  courses.deleteCourse(id)  console.log('\n Successfuly deleted'.green)  }  catch (e) {  console.log('\n Item is not deleted. Check the entered Id'.red)  }  }  else if (com.includes('delete/users/')) {  const id = Number(com.split('/')[2])  try {  users.deleteUser(id)  console.log('\n Successfuly deleted'.green)  }  catch (e) {  console.log('\n Item is deleted. Check the entered Id'.red)  }  }  else if (com.includes('update/users/')) {  const id = Number(com.split('/')[2])  let user = users.getUserById(id)  const login = readLineSync.question('Enter new login: ')  if (login.length === 0) {  console.log('\nLogin can not be empty\n'.red)  continue  }  user.login = login  const fullname = readLineSync.question('Enter new full name: ')  if (fullname.length === 0) {  console.log('\nFull name can not be empty\n'.red)  continue  }  user.fullname = fullname  const role = readLineSync.question('Enter new role (0 or 1): ')  if (role != 0 && role != 1) {  console.log('\nRole is not correct\n'.red)  continue  }  user.role=role  let time = new Date().getTime();  let date = new Date(time);  user.registeredAt = date.toString()  const avaUrl = readLineSync.question('Enter new URL : ')  user.avaUrl = avaUrl  const isEnabled = readLineSync.question('Enter isEnabled (0 or 1): ')  if (isEnabled != 0 && isEnabled != 1) {  console.log('\nisEnabled is not correct\n'.red)  continue  }  user.isEnabled=isEnabled  users.updateUser(user)  }  else if (com.includes('update/courses/')) {  const id = Number(com.split('/')[2])  let course = courses.getCourseById(id)  const name = readLineSync.question('Enter new course: ')  if (name.length === 0) {  console.log('\nField course can not be empty\n'.red)  continue  }  course.name = name  const teacher = readLineSync.question('Enter new teacher: ')  if (teacher.length === 0) {  console.log('\nField teacher can not be empty\n'.red)  continue  }  course.teacher = teacher  const hours = readLineSync.question('Enter new amount of hours: ')  if (isNaN(hours)) {  console.log('\nField hours should be a digit\n'.red)  continue  }  course.hours = hours  const credits = readLineSync.question('Enter new amount of credits: ')  if (isNaN(credits)) {  console.log('\nField credits should be a digit\n'.red)  continue  }  course.credits = credits  let time = new Date().getTime();  let date = new Date(time);  course.beginingDate = date.toString()  courses.updateCourse(course)  }  else if (com.includes('post/users')) {  let user = {}  const login = readLineSync.question('Enter new login: ')  if (login.length === 0) {  console.log('\nLogin can not be empty\n'.red)  continue  }  user.login = login  const fullname = readLineSync.question('Enter new full name: ')  if (fullname.length === 0) {  console.log('\nFull name can not be empty\n'.red)  continue  }  user.fullname = fullname  const role = readLineSync.question('Enter new role (0 or 1): ')  if (role != 0 && role != 1) {  console.log('\nRole is not correct\n'.red)  continue  }  user.role=role  let time = new Date().getTime();  let date = new Date(time);  user.registeredAt = date.toString()  const avaUrl = readLineSync.question('Enter new URL : ')  user.avaUrl = avaUrl  const isEnabled = readLineSync.question('Enter isEnabled (0 or 1): ')  if (isEnabled != 0 && isEnabled != 1) {  console.log('\nisEnabled is not correct\n'.red)  continue  }  user.isEnabled = isEnabled  users.addUser(user)  }  else if (com.includes('post/courses')) {  let course = {}  const name = readLineSync.question('Enter new course: ')  if (name.length === 0) {  console.log('\nField course can not be empty\n'.red)  continue  }  course.name = name  const teacher = readLineSync.question('Enter new teacher: ')  if (teacher.length === 0) {  console.log('\nField teacher can not be empty\n'.red)  continue  }  course.teacher = teacher  const hours = readLineSync.question('Enter new amount of hours: ')  if (isNaN(hours)) {  console.log('\nField hours should be a digit\n'.red)  continue  }  course.hours = hours  const credits = readLineSync.question('Enter new amount of credits: ')  if (isNaN(credits)) {  console.log('\nField credits should be a digit\n'.red)  continue  }  course.credits = credits  let time = new Date().getTime();  let date = new Date(time);  course.beginingDate = date.toString()  courses.addCourse(course)  }  else if (com === 'exit') flag = 0  else console.log('command is not correct'.red)  }  module.exports = users |

**Віст файлу course.json**

|  |
| --- |
| {  "nextId": 17,  "items": [  {  "id": 1,  "name": "Science",  "teacher": "Ivanova",  "hours": 90,  "credits": 3,  "beginingDate": "Wed Oct 14 2020 04:08:30 GMT+0300 (EEST)"  },  {  "id": 2,  "name": "Mathimatics",  "teacher": "Legeza",  "hours": "150",  "credits": "5",  "beginingDate": "Thu Oct 15 2020 00:05:29 GMT+0300 (EEST)"  },  {  "id": 3,  "name": "Linux",  "teacher": "Rybachok",  "hours": "120",  "credits": "4",  "beginingDate": "Wed Oct 14 2020 04:08:30 GMT+0300 (EEST)"  },  {  "id": 12,  "name": "linal",  "teacher": "Sushuk-Slusarenko",  "hours": "60",  "credits": "2",  "beginingDate": "Wed Oct 14 2020 23:42:07 GMT+0300 (EEST)"  },  {  "id": 13,  "name": "drawing",  "teacher": "Petrash",  "hours": "120",  "credits": "4",  "beginingDate": "Wed Oct 14 2020 23:59:17 GMT+0300 (EEST)"  }  ]  } |

**Віст файлу user.json**

|  |
| --- |
| {  "nextId": 20,  "items": [  {  "id": 1,  "login": "admin",  "fullname": "Admin Admin",  "role": 1,  "registeredAt": "Wed Oct 14 2020 01:15:32 GMT+0300 (EEST)",  "avaUrl": "https://developer.mozilla.org/ru/pct",  "isEnabled": 1  },  {  "id": 2,  "login": "j",  "fullname": "julia sivokon",  "role": 0,  "registeredAt": "Wed Oct 14 2020 04:45:12 GMT+0300 (EEST)",  "avaUrl": "URL",  "isEnabled": 0  },  {  "id": 3,  "login": "ann",  "fullname": "ann ivanova",  "role": 0,  "registeredAt": "Wed Oct 14 2020 04:02:56 GMT+0300 (EEST)",  "avaUrl": "http",  "isEnabled": 0  },  {  "id": 4,  "login": "user",  "fullname": "My name",  "role": 0,  "registeredAt": "Wed Oct 14 2020 07:40:21 GMT+0300 (EEST)",  "avaUrl": "https://google.com/pct",  "isEnabled": 1  },  {  "id": 17,  "login": "sunshine",  "fullname": "igor grigoruk",  "role": "theurl",  "registeredAt": "1",  "avaUrl": "Wed Oct 14 2020 23:57:27 GMT+0300 (EEST)",  "isEnabled": "1"  }  ]  } |

**Приклади результатів**

|  |
| --- |
| **get/courses ; get/users** |
| **get/courses/id; get/users/id** |
|  |

**delete/courses/id ; deleted/users/id**

|  |
| --- |
| До / після |

**update/users/id; update/courses/id**

|  |
| --- |
|  |

**post/users; post/courses**

|  |
| --- |
|  |

|  |
| --- |
| **Приклад консольного інтерфейсу** |

|  |
| --- |
| **Приклад обробки помилок** |

**Висновки**

У лабораторній роботі було розроблено сховище для основної сутності та для сутносты користувача, яке за допомогою консольного інтерферсу може взаємодіяти з файлами формату JSON. Робота була виконана мовою програмування JavaScript у середовищі Note.js.

Розроблено модулі, які працюють з синхронно з файловими потоками, і обробляють данні, які є моделями. Користувач має змогу зчитати данні з файлу, вибрати модель з подрібним індексом, редагувати, видаляти та створювати модель.

Як додаткові модуль використано ‘readline-sync’ та ‘colors’ .