Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

|  |  |
| --- | --- |
| Факультет | Копьютерна інженерія |
|  | (повна назва) |

|  |  |
| --- | --- |
| Кафедра | електронних обчислювальних машин |
|  | (повна назва) |

**КУРСОВИЙ ПРОЕКТ**

|  |  |
| --- | --- |
| Рівень вищої освіти |  |

|  |
| --- |
| Системи захисту програмного забезпечення в операційних |
| системах сімейства Unix |
| (Linux) та аналіз засобів зламу цих захистів. |
| (тема) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Виконав: | | | | | |
| студент |  | | курсу, групи | | 21-1 |
| Сєров А.С. | | | | | |
| (прізвище, ініціали) | | | | | |
| Спеціальність | | | КІУКІ | | |
|  | | | | | |
| (код і повна назва спеціальності) | | | | | |
| Освітня програма | | | | Комп’ютерна інженерія | |
|  | | | | | |
| (повна назва освітньої програми) | | | | | |
| Керівник: | |  | | | |
|  | | (посада, прізвище, ініціали) | | | |

2023 р.

**РЕФЕРАТ**

Курсова робота: 22 ст., 3рис., 16 додатків ,3 джерел

КОМП’ЮТЕРНА МЕРЕЖА, ІНТЕРНЕТ, UNIX, ACL, MAC, DAC, DTE, RBAC, API, NESTJS

Метою курсової роботи є вивчення систем захисту програмного забепечення в операційних системах сімейства Unix та впевненість в безпечності системи Linux.

У ході виконання курсової роботи було вивчено способи захисту програмного забезпечення та операційної системи linux. Такі методи захисту як ACL, MAC, DAC, DTE та RBAC. На прикладі показано безпечність та надійність використання api-сервісів замість локального варіанту. Також робота над цим проектом допомогла в виченні роботи з git та github. У розділі висновку є посилання на цей проект у повному розмірі, так як для пояснення не було додано до структури проекту частину файлів без яких проєкт не запустити.

**ПЛАН**

Contents

[ВСТУП 3](#_Toc153963567)

[Модель доступу Unix 3](#_Toc153963568)

[Стандартні моделі доступу Unix 3](#_Toc153963569)

[Дискретне управління доступом (Discretionary Access Control, DAC) 3](#_Toc153963570)

[Модель тип-домен (Domain Type Enforcement, DTE) 4](#_Toc153963571)

[Списки управління доступом (Access Control Lists, ACL) 4](#_Toc153963572)

[Управління доступом на основі ролей (Role-based Access Control, RBAC): 4](#_Toc153963573)

[Мандатне управління доступом (Mandatory Access Control, MAC): 4](#_Toc153963574)

[Використання списків контролю доступу (ACL) 5](#_Toc153963575)

[Гранулярність 5](#_Toc153963576)

[Використання в різних областях 5](#_Toc153963577)

[Файлові системи з ACL 5](#_Toc153963578)

[Використання кілець захисту 5](#_Toc153963579)

[Засоби зламу захисту ПЗ в системах Unix 6](#_Toc153963580)

[Підбір паролю користувача 6](#_Toc153963581)

[Уразливості неоновленого ПЗ 7](#_Toc153963582)

[Неналежне використання прав доступу 7](#_Toc153963583)

[Атаки на сервіси 7](#_Toc153963584)

[Приклад захисту ПЗ 8](#_Toc153963585)

[Структура програми: 8](#_Toc153963586)

[Клієнтська частина 8](#_Toc153963587)

[Серверна частина 10](#_Toc153963588)

[ВИСНОВОК 11](#_Toc153963589)

ВСТУП

Системи захисту в операційних системах сімейства Unix базуються на декількох ключових принципах:

1. Модель доступу Unix
2. Використання списків контролю доступу (ACL)
3. Використання кілець захисту

Однак, важливо зазначити, що Unix та Linux не є фундаментально безпечними операційними системами. Це означає, що для забезпечення безпеки програмного забезпечення в Unix та Linux потрібно використовувати додаткові заходи безпеки.

Існують загальні практики для підвищення безпеки пристроїв Unix та Linux:

1. Застосування принципу найменшого привілею - це означає, що користувачам та програмам надаються лише ті привілеї, які їм дійсно потрібні для виконання їхніх завдань.
2. Регулярне оновлення системи - це включає встановлення останніх патчів безпеки та оновлень.
3. Обмеження доступу до системи - це може включати використання файрволів, VPN та інших технологій для обмеження доступу до системи.
4. Моніторинг системи - це включає використання систем моніторингу для виявлення незвичайної або підозрілої активності.
5. Використання інструментів безпеки Linux - Linux має вбудовані інструменти безпеки, такі як файрволи, які використовують пакетні фільтри в ядрі, механізм перевірки прошивки UEFI Secure Boot, опцію конфігурації Linux Kernel Lockdown та системи покращення безпеки SELinux або AppArmor Mandatory Access Control (MAC).

Модель доступу Unix

Стандартні моделі доступу Unix

У останні роки було проведено багато досліджень у сфері гнучкого управління доступом. Варто відзначити, що всі ці методи призначені для кваліфікованих користувачів і вимагають спеціального налаштування. Проте в результаті можна отримати захищену систему, здатну витримати навіть при постійних атаках. Почнемо з огляду стандартних моделей доступу Unix.

Дискретне управління доступом (Discretionary Access Control, DAC)

DAC - це формальна назва стандартної системи користувачів і прав доступу, яка визнана всіма Unix-користувачами. У даній системі доступ до об’єкта (наприклад до файлу) контролюється власником даного об’єкта за допомогою прав доступу. Власник може контролювати не тільки тих користувачів, яким дозволений доступ до об’єкта, але й контролювати режим доступу (наприклад, читання, запис, виконання). Дискретне управління доступом представляє собою розмежування доступу між іменованими суб’єктами та іменованими об’єктами. Суб’єкт з певним правом доступу може передати це право будь-якому іншому суб’єкту. Даний вид організується на базі методів розмежування за списками або за допомогою матриці.

Модель тип-домен (Domain Type Enforcement, DTE)

DTE - це одна з реалізацій MAC, заснована на концепції мінімальних привілеїв: процесу повинні бути надані мінімально необхідні привілеї (тобто тільки ті, які реально обов’язкові процесу). У DTE об’єкти (файли) формують типи, а суб’єкти (процеси) - домени. Таблиця DDT (Domain Definition Table - таблиця визначення домену) описує, як домени та типи можуть взаємодіяти один з одним.

Списки управління доступом (Access Control Lists, ACL)

Unix-система прав доступу файлів визначені режимами файлів:

8 бітів задають, хто може отримати доступ до цього файлу, і визначають режими доступу. Оскільки для прав доступу застосовуються 9 бітів, то це дозволяє створити три класи користувачів (власник, група, інші) і три класи доступу (читання, запис, виконання).

Файлові системи ext2 та ext3 підтримують розширені атрибути файлів (Extended Attributes, EA), які дозволяють встановлювати додаткові параметри для кожного файлу. EA можуть використовуватися для надання прав доступу додатковим групам користувачів, що дозволяє створювати ACL.

Управління доступом на основі ролей (Role-based Access Control, RBAC):

В цій моделі кожному користувачеві призначається одна або кілька певних ролей, які він може реалізовувати в системі.

Мандатне управління доступом (Mandatory Access Control, MAC):

В MAC доступ контролюється не власником, а адміністратором системи. В MAC всі користувачі та файли мають призначені рівні доступу, наприклад, “таємно”, “цілком таємно”.

Рівні доступу впорядковуються за принципом домінування одного рівня над іншим. Потім доступ до захищених файлів здійснюється за двома простими правилами:

* Користувач має право читати лише ті документи, рівень безпеки яких не перевищує його власний рівень безпеки. Це правило забезпечує захист інформації, яку обробляють користувачі з вищим рівнем безпеки, від доступу з боку користувачів з нижчим рівнем безпеки.
* Користувач має право вносити інформацію лише в ті документи, рівень безпеки яких не нижчий за його власний рівень безпеки. Це правило запобігає порушенню режиму доступу з боку учасників процесу обробки інформації з вищим рівнем безпеки до користувачів з нижчим рівнем безпеки.

Використання списків контролю доступу (ACL)

ACL - це один з основних способів керування безпекою в комп’ютерних мережах та системах. Вони дозволяють адміністраторам контролювати доступ до конфіденційної інформації та захищати системи від несанкціонованого доступу та зловживань тими правами доступу, які дозволені.

Ключові аспекти використання ACL:

Гранулярність

ACL дозволяє встановлювати деталізовані права доступу до ресурсів для окремих користувачів та груп. Наприклад, ACL з високою гранулярністю має користувачів та груп, в той час як низька гранулярність означає загальні правила для всіх.

Використання в різних областях

ACL можуть застосовувати у корпоративних мережах, системі керування базами даних, мережевому обладнанні (наприклад, підключаючи інтернет провайдер заводить у дім 10Гбіт кабель, а користувачі отримують лише ту швидкість за яку заплатили), хмарних послугах (Google Drive може зберігати терабайти даних, але залежно від плану який купив користувач, йому доступна лише виділена частина, а компанія Google може як читати так і редагувати дані користувача, так як має найвищі права), операційних системах.

Файлові системи з ACL

У файлових системах для реалізації використовується ідентифікатор користувача процесу (UID в термінах POSIX). Список доступу являє собою структуру даних (зазвичай таблицю), що містить записи, які визначають права індивідуального користувача або групи на спеціальні системні об’єкти, такі як програми, процеси або файли.

Використання кілець захисту

Кільця захисту - це архітектура інформаційної безпеки та функціональної відмовостійкості, що реалізує апаратний поділ системного й користувацького рівнів привілеїв. Структуру привілеїв можна зобразити у вигляді кількох концентричних кіл.

У цьому випадку системний режим (режим супервізора або нульове кільце, так зване «кільце 0»), що забезпечує максимальний доступ до ресурсів, є внутрішнім колом, тоді як режим користувача з обмеженим доступом — зовнішнім.

Традиційно сімейство мікропроцесорів x86 забезпечує чотири кільця захисту. Архітектурі кілець захисту зазвичай протиставляють системи, засновані на мандатній адресації, що забезпечує доступ до об’єкта за його описом.

Механізм кілець суворо обмежує шляхи, за допомогою яких керування можна передати від одного кільця до іншого, а також передбачає обмеження на операції доступу до пам’яті, які можуть бути зроблені всередині кільця. Зазвичай існує деяка інструкція (шлюз), що передає контроль від менш захищеного до більш захищеного (з меншим номером) кільця.

Одним з прикладів інструкції для передачі контролю меншому кільцю в архітектурі кілець захисту є системний виклик або інструкція привілейованого рівня, яка дозволяє виконувати операції в кільці з більшими привілеями.

Наприклад, в операційних системах, що базуються на архітектурі x86, інструкція «call gate» використовується для переходу між різними рівнями привілеїв, дозволяючи коду з меншим рівнем привілеїв (наприклад, кільце 3) викликати процедури, які знаходяться в більш привілейованому кільці (наприклад, кільце 0).

Засоби зламу захисту ПЗ в системах Unix

Насамперед потрібно розуміти, що системи на базі Unix мають дуже міцну і універсальну систему безпеки. Всі злами та хакерські атаки відбуваються через неправильно налаштовану систему доступу або людський фактор.

Основна відмінність від системи Windows у тому, що для роботи Linux не потрібна більшість сервісів, а також при встановленні можна налаштувати ядро системи заздалегідь вимкнувши потенційно можливі шляхи зламу (наприклад USB). Антивірусна програма також не має сенсу так як підозрілі дії не зможуть виконатись без відповідних прав користувача.

Наприклад якщо працівник компанії користуючись Unix сервером завантажить шкідливе ПЗ під загрозою лише та частина файлів, які доступні цьому працівнику. Також в кожній складній і важливій системі існують можливісті робити резервні копії, що мають найвищий рівень доступу.

Іноді можливе лиже відновлення фізичним шляхом, тобто копія може зберігатись на носії, що можна фізично під’єднати до серверу для синхронізації і відкласти для зберігання.

Серед основних способів зламу можна виділити такі:

Підбір паролю користувача

Якщо система не має обмеженої кількості перевірки паролів то лише запустивши скрипт підбору паролю можна отримати повний контроль над пристроєм. І якщо обеження все ж існує, потрібно ретельно вибирати або генерувати пароль.

Зазвичай пароль має бути довгим, використовувати різноманітні символи і також не мати сенсу. Паролі накшталт 12345678 або дата народження - це перше, що приходить на думку і так само може думати зловмисник.

Уразливості неоновленого ПЗ

При розробці ПЗ можуть з’явитись різноманітні уразливості, які можуть виправити в майбутніх оновленнях. В приклад можна навести вразливість Windows кілька років тому, що призвела до хвилі хакерських атак. Звісно Unix подібні системи це не зачепило, але ніколи не можна виключати такої можливості, особливо коли Linux стає все популярніше.

Неналежне використання прав доступу

Як написано на початку розділу вся небезпека залежить лише від неправильних налаштувань прав доступу. Процеси, що потребують права мають мати мінімальну їх кількість. Наприклад калькулятор не повинен мати доступ до будь-яких файлів окрім тих, що він сам створив.

Атаки на сервіси

Такий спосіб зламу має ту ж ідею що неоновлене ПЗ. Якщо сервіс було оновлено або модифіковано користувачем то існує шанс, що сервіс має вразливість.

Наприклад, в компанії NVIDIA існує сервіс для стімінгу, що надає графічний контроль над пристроєм, тобто можна користуватись мишею і клавіатурою, в тому числі можливо відкрити темінал і виконати команди.

Для трансляції у локальній мережі навіть пароль не потрібен, а якщо ввімкнути роботу з глобальною мережею то сервіс має доступ до переадресації портів на роутері локальної мережі. Ця система надає зловмисникам набагато більше можливостей для ураження.

Звісно без паролю супервізора багато нашкодити не вийде, але загроза існує.

Наразі в операційних системах, що потребують захисту, використовуються комбінації різних методів захисту. Все залежить від багатьох факторів, від збоїв в роботі системи до вмотивованості зловмисника.

Приклад захисту ПЗ

Останні роки компанії, що надають різноманітні послуги активно переносять важливу частину коду, що не повинна потрапити до рук зловмисників або халявщиків, до хмари. Тим самим створюючи хмарні сервіси. Деякі переносять весь функціонал в браузер як це зробили Microsoft з програмами пакету office, інші навпроти викоритсовують API.

API – це функція, що виконується на сервері за запитом користувача і повертає йому результат виконання. Наприклад компанія Adobe додаючи у свою програму Photoshop AI-помічника, що домальовує зображення або просто допомагає в редактуванні. Він працює лише при наявному доступі в інтернет і активній підписці. Так всі ті хто завантажив цю програму з торрентів не зможуть скористатись новою функцією.

В моєму прикладі буде використано часткову заміну функціоналу програми.

Структура програми:

* Api.java (Додаток А)
* Main.java (Додаток Б)
* Login.java (Додаток В)
* ProfileFrame.java (Додаток Г)
* server-side-project
  + src
    - Api
      * api.controller.ts (Додаток Д)
      * api.module.ts (Додаток Є)
      * api.service.ts (Додаток Ж)
      * user.dto.ts (Додаток З)
    - Login
      * login.controller.ts (Додаток І)
      * login.module.ts (Додаток К)
      * login.service.ts (Додаток Л)
    - app.controller.ts (Додаток М)
    - app.module.ts (Додаток Н)
    - app.service.ts (Додаток О)
    - main.ts (Додаток П)
  + database.json (Додаток Р)

Програма поділена на дві основні частини – клієнтську та серверну.

За клієнтську відповідають \*.java файли, а за серверну директорія server-side-project.

Клієнтська частина

Програма написана на мові Java.

Main.java – викликає вікно авторизаціїї Login.java.

Login.java – створює вікно «login window» в якому існують поле логіну «username», поле паролю «password» та кнопка відправки даних «login» (Рисунок 1).

A screenshot of a computer screen

Description automatically generated

*Рисунок 1 – зовнішній вигляд вікна «login window»*

Для перевірки користувача до серверу надсилається запит.

Якщо такого користувача не існує або невірний пароль то запит повертає пустий рядок і у клієнта відображається інформація про це.(Рисунок 2)

A screen shot of a computer

Description automatically generated

*Рисунок 2 – зовнішній вигляд вікна «error»*

Якщо дані вірні то створюється токен (Генерований рядок тексту). Він повертається клієнту як результат запиту на вхід. Також він записується до бази даних. Кожен запит, окрів login, потребує для своєї роботи передати токен. Таким чином запити повернуть результат лише тим хто авторизувався.

Api.java – клас для роботи з API запитами де існує метод «getResponce», що примає рядок параметрів та повертає радок результату.

ProfileFrame.java – створює нове вікно, якщо авторизація пройшла успішно. У ньому зображено ім’я користувача та його біографія разом з кнопкою виходу.(Рисунок 3)

A screenshot of a computer

Description automatically generated

*Рисунок 2 – зовнішній вигляд вікна «profile»*

Ім’я та біографія отримані шляхом надсилання двох запитів до сервера.

В даному прикладі у користувача існує лише чотири поля – username, password, token та bio. В реальному проєкті структура бази даних буде більш складною.

Серверна частина

На сервері зберігається база даних у виглядів json файлу «database.json». Так як залежностей у базі даних немає, необхідності у використанні реальної бази даних теж немає. Структура бази даних:

{

“users” : [{

Name:string,

Token:string,

Password:string,

Bio:string

}]

}

Серверна частина написана на NestJS – це фреймворк NodeJS для створення серверної частини сайтів.

Існує три умовні частини проекту: Api, Login, App.

Main.ts – це початок програми, в ньому запускається сервер та виділяється під нього порт 3000.

App – це початкова точка серверу, до нього відносяться такі файли:

App.module.ts – цей файл додає api посилання до серверу

App.controller.ts – цей файл дозволяє налаштувати які саме api будуть викликані при відповідних параметрах запиту

App.service.ts – в цоьму файлі зберігаються всі функції що викликає контроллер

Login – має таку ж структуру, але запити повинні посилатись на адрессу сайту + «/login» для доступу до api цієї частини.

Api – аналогічна до попередньго ситуація. Цей роздів зберігає в собі основну частину запитів, а також працює з базою даних. Файл user.dto.ts використовується як тип даних для поля user.

ВИСНОВОК

В цій курсовій роботі було розглянуто способи захисту програмного забезпечення в Unix системах, а саме дискретне управління доступом, модель тип-домен, списки управління доступом, управління доступом на основі ролей, мандатне управління доступом та використання кілець захисту.

Реалізовано універсальний спосіб захисту у вигляді api-запитів до серверу. Програма написана на мовах Java та TypeScript. Було вивчено спосіб взаємодії серверу з клієнтом на прикладі api-запиту. Цей проект можна завантажити за посиланням <https://github.com/Senderion13/CourseWork3_SPZ>

Якщо потрібно запустити проект, то потрібно спочатку завантажити проект або виконати git clone.

В командному рядку виконати наступні команди для запуску серверу:

cd server-side-project

npm install

npm run start

Після чого в новому командному рядку запустити Main.java за допомогою плагіну або команди java

ВИКОРИСТАНІ ДЖЕРЕЛА

https://uk.wikipedia.org/

<https://nestjs.com/>

https://www.javatpoint.com/

ДОДАТОК А

import java.io.\*;

import java.net.\*;

public class Api {

    public String getResponse(String params) throws Exception {

        // URL of the API

        URL url = new URL("http://localhost:3000" + params);

        // Create a connection

        HttpURLConnection connection = (HttpURLConnection) url.openConnection();

        // Set the request method to GET

        connection.setRequestMethod("GET");

        // Include any necessary headers

        connection.setRequestProperty("Content-Type", "application/json");

        // Get the response code

        int responseCode = connection.getResponseCode();

        // If the request was successful, read and print the response

        if (responseCode == HttpURLConnection.HTTP\_OK) {

            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

            String inputLine;

            StringBuffer content = new StringBuffer();

            while ((inputLine = in.readLine()) != null) {

                content.append(inputLine);

            }

            // Close connections

            in.close();

            connection.disconnect();

            return content.toString();

        } else {

            return "GET request failed. Response Code: " + responseCode;

        }

    }

}

ДОДАТОК Б

public class Main {

    public static void main(String[] args) {

        new Login();

    }

}

ДОДАТОК В

import javax.swing.\*;

import javax.swing.text.Style;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Login {

    Login() {

        try {

            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        } catch (Exception e) {

            e.printStackTrace();

        }

        // Create a new JFrame (window)

        JFrame frame = new JFrame("Login Window");

        frame.setSize(300, 300);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        // Create a JPanel with a BoxLayout

        JPanel panel = new JPanel();

        panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

        panel.setBackground(Color.DARK\_GRAY);

        // Create a JLabel for the username

        JLabel userLabel = new JLabel("Username");

        userLabel.setForeground(Color.WHITE);

        userLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

        // Create a JTextField for the username input

        JTextField userField = new JTextField();

        userField.setMaximumSize(new Dimension(200, 40));

        // Create a JLabel for the password

        JLabel passLabel = new JLabel("Password");

        passLabel.setForeground(Color.WHITE);

        passLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

        // Create a JPasswordField for the password input

        JPasswordField passField = new JPasswordField();

        passField.setMaximumSize(new Dimension(200, 40));

        // Create a JButton for the login button

        JButton loginButton = new JButton("Login");

        loginButton.setAlignmentX(Component.CENTER\_ALIGNMENT);

        loginButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                String userText = userField.getText().trim();

                String passText = passField.getText().trim();

                try {

                    String response = new Api().getResponse("/login?user=" + userText + "&pass=" + passText);

                    if (response.isEmpty()) {

                        JOptionPane.showMessageDialog(frame, "Invalid Username or Password", "Error",

                                JOptionPane.ERROR\_MESSAGE);

                    } else {

                        String token = response;

                        String name = new Api().getResponse("/api/name?token=" + token);

                        String bio = new Api().getResponse("/api/bio?token=" + token);

                        new ProfileFrame(name, bio);

                        frame.dispose();

                    }

                } catch (Exception error) {

                    System.out.println("Error: " + error.getMessage());

                }

            }

        });

        // Add the components to the JPanel

        panel.add(Box.createRigidArea(new Dimension(0, 20))); // Add some space at the top

        panel.add(userLabel);

        panel.add(userField);

        panel.add(Box.createRigidArea(new Dimension(0, 10))); // Add some space between the fields

        panel.add(passLabel);

        panel.add(passField);

        panel.add(Box.createRigidArea(new Dimension(0, 20))); // Add some space between the fields and the button

        panel.add(loginButton);

        // Add the JPanel to the JFrame and make it visible

        frame.add(panel);

        frame.setVisible(true);

    }

}

ДОДАТОК Г

// Import swing components

import java.awt.BorderLayout;

import javax.swing.\*;

// Create a class that extends JFrame

public class ProfileFrame extends JFrame {

    // Declare swing components as instance variables

    private JLabel nameLabel, bioLabel;

    private JButton closeButton;

    // Constructor for the profile frame

    public ProfileFrame(String name, String bio) {

        // Set the title, size, layout, and closability of the frame

        setSize(600, 400);

        BorderLayout borderLayout = new BorderLayout();

        setLayout(borderLayout);

        // Create and initialize the swing components

        nameLabel = new JLabel(name, JLabel.CENTER);

        bioLabel = new JLabel(bio, JLabel.CENTER);

        closeButton = new JButton("Close");

        // Add action listeners to the buttons

        closeButton.addActionListener(e -> closeFrame());

        // Add the components to the frame

        add(nameLabel, BorderLayout.NORTH);

        add(bioLabel, BorderLayout.CENTER);

        add(closeButton, BorderLayout.SOUTH);

        // Make the frame visible

        setVisible(true);

    }

    // Method to close the frame

    private void closeFrame() {

        dispose();

    }

}

ДОДАТОК Д

import { Controller, Get, Query } from '@nestjs/common';

import { ApiService } from './api.service';

@Controller('api')

export class ApiController {

  @Get('name')

  getUserName(@Query('token') token: string) {

    return ApiService.prototype.getUserName(token);

  }

  @Get('bio')

  getUserBio(@Query('token') token: string) {

    return ApiService.prototype.getUserBio(token);

  }

}

ДОДАТОК Є

import { Module } from '@nestjs/common';

import { ApiController } from './api.controller';

import { ApiService } from './api.service';

@Module({

  imports: [],

  controllers: [ApiController],

  providers: [ApiService],

})

export class ApiModule {}

ДОДАТОК Ж

import { Injectable } from '@nestjs/common';

import \* as json from '../../database.json';

import { userDto } from './user.dto';

@Injectable()

export class ApiService {

  getUserName(token: string) {

    const usersList: userDto[] = json.users;

    return usersList.find((user: userDto) => {

      if (token == user.token) {

        return true;

      } else {

        return false;

      }

    }).name;

  }

  getUserBio(token: string) {

    const usersList: userDto[] = json.users;

    return usersList.find((user: userDto) => {

      if (token == user.token) {

        return true;

      } else {

        return false;

      }

    }).bio;

  }

}

ДОДАТОК З

export type userDto = {

  name: string;

  token: string;

  password: string;

  bio: string;

};

ДОДАТОК І

import { Controller, Get, Query } from '@nestjs/common';

import { LoginService } from './login.service';

@Controller('login')

export class LoginController {

  @Get()

  Login(@Query('user') user: string, @Query('pass') pass: string) {

    return LoginService.prototype.Login(user, pass);

  }

}

ДОДАТОК К

import { Module } from '@nestjs/common';

import { LoginController } from './login.controller';

import { LoginService } from './login.service';

@Module({

  imports: [],

  controllers: [LoginController],

  providers: [LoginService],

})

export class LoginModule {}

ДОДАТОК Л

import { Injectable } from '@nestjs/common';

import \* as json from '../../database.json';

//import \* as fs from 'fs';

@Injectable()

export class LoginService {

  Login(userName: string, userPass: string): string {

    let tempToken: string;

    const index = json.users.findIndex(({ name, password }) => {

      if (userName == name && userPass == password) {

        tempToken = rand();

        return true;

      }

    });

    if (index >= 0) {

      json.users[index].token = tempToken;

      //fs.writeFileSync('./database.json', JSON.stringify(json), 'utf-8');

      return tempToken;

    } else {

      return '';

    }

  }

}

const rand = () => {

  return Math.random().toString(36).substr(2);

};

ДОДАТОК М

import { Controller, Get } from '@nestjs/common';

import { AppService } from './app.service';

@Controller()

export class AppController {

  constructor(private readonly appService: AppService) {}

  @Get()

  getHello(): string {

    return this.appService.getHello();

  }

}

ДОДАТОК Н

import { Module } from '@nestjs/common';

import { AppController } from './app.controller';

import { AppService } from './app.service';

import { ApiController } from './Api/api.controller';

import { LoginController } from './Login/login.controller';

@Module({

  imports: [],

  controllers: [AppController, LoginController, ApiController],

  providers: [AppService],

})

export class AppModule {}

ДОДАТОК О

import { Injectable } from '@nestjs/common';

@Injectable()

export class AppService {

  getHello(): string {

    return 'Hello World!';

  }

}

ДОДАТОК П

import { NestFactory } from '@nestjs/core';

import { AppModule } from './app.module';

async function bootstrap() {

  const app = await NestFactory.create(AppModule);

  await app.listen(3000);

}

bootstrap();

ДОДАТОК Р

{"users":[{"name":"andrey","token":"qzyr6xgh94","password":"12345","bio":"some text about andrey"},{"name":"alexey","token":"","password":"qwert","bio":"some text about alexey"}]}