ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ – Програмне Забезпечення

ОС – Операційна Система

СВВ – Система Виявлення Вторгнень

IPS – Intrusion Prevention System

HIPS – Host-Based Intrusion Prevention System

ЦПП – Центральний Процесорний Пристрій

ПрО – Предметна Область

WPFM – Web-Page Formal Model

ВСТУП

В сучасному світі разом зі поширенням використання веб-технологій у повсякденному житті, зростає і кількість шкідливих програм, що розповсюджуються по комп'ютерним мережам. Інтернет-браузери та їх розширення, як і інші додатки, мають певні вразливості, що можуть бути експлуатовані зловмисниками. Основну небезпеку представляють ті, що дозволяють виконувати довільний та підозрілий код у системі жертви.

Основним засобом боротьби зі шкідливими програмами сьогодні є сигнатурний аналіз, тобто виявлення послідовності команд, що погрожують безпеці даних. Конструювання баз даних сигнатур вірусів є предметом діяльності великої кількості організацій, що займаються впровадженням відповідного ПЗ. В основі цього підходу лежить фільтрація шкідливого програмного коду з достатньо високою ефективністю. Проте, у зв'язку з різким ростом об'єму даних, який підлягає фільтрації, цей підхід породжує деякі складності, що пов'язані з необхідністю аналізу всіх оброблених комп'ютером даних. Авжеж, це негативно впливає на продуктивність роботи системи. Також зростає частота появи нових вірусних скриптів, тож з'явилась необхідність більш частого оновлення баз даних вірусних сигнатур. Ці проблеми призводять до необхідності впровадження альтернативних методів виявлення та захисту від шкідливих скриптів.

Одним з напрямків боротьби з вірусами є розробка так званих імунних систем. Імунні системи основані на підходах, що є аналогічними до імунної системи живих організмів. Цей підхід дозволяє комп'ютерним системам «довчитися» в процесі функціонування та надалі самостійно виявляти шкідливий код.

Сьогодні вже існує декілька систем захисту інформації, що працюють по аналогії з імунними системами. Проте, ці системи мають спеціалізований характер, коли задача розробки універсальної системи залишається відкритою, та не позбавлені достатньої кількості погрішностей. Також не можна не зауважити, що і

розробка спеціалізованих систем має високу цінність, бо існують окремі класи однотипних вірусних скриптів, захист від яких залишається актуальною задачею.

1 ДОСЛІДЖЕННЯ ІСНУЮЧИХ ВИДІВ ШКІДЛИВОГО ПЗ ТА МЕТОДІВ ЙОГО ВИЯВЛЕННЯ. ПОСТАНОВКА ЗАДАЧІ.

У даному розділі буде розглянуто теоретичні основи для аналізу шкідливого ПО, включаючи механізми шкідливих програм для ускладнення виявлення антивірусними програмами, а також їх вплив на інструменти аналізу.

У даному розділі буде розглянуто теоретичні основи для аналізу шкідливого ПО, включаючи механізми шкідливих програм для ускладнення виявлення антивірусними програмами, а також їх вплив на інструменти аналізу.

1.1 Види шкідливого ПЗ та їх особливості

Шкідливе ПЗ – це будь-яке ПЗ, що викликає шкоду для користувача, комп'ютера або мережі. [1]

Для виявлення шкідливого коду необхідно володіти даними про поведінку шкідливого ПЗ в залежності від його типу, способу впровадження і зараження системи, і до яких наслідків це призводить. Аналітики шкідливого ПЗ і антивірусні компанії поділяють шкідливі програми за іменами та особливостями поведінки [2]. Розглянуті нижче поведінкові типи не є взаємовиключними: шкідливі програми можуть об'єднати в собі риси кількох

типів.

Hijacker змінює налаштування браузера без згоди користувача. Віруси цього типу замінюють стандартну сторінку пошуку (пошукову систему), домашню сторінку, сторінки помилок і додають нові закладки зі своїм контентом. Вони також ускладнюють скасування змін і можуть викликати проблеми, якщо користувач намагається видалити їх.

Приклади: Conduit Search Protect, Babylon Toolbar.

Trojan (троянський кінь, троян) – шкідлива програма, яка обманює користувачів для свого запуску, надаючи корисну функціональність або повідомляючи користувачеві, що це корисна і безпечна програма. [3, с.37] Троянські коні комбінуються з додатковими модулями, наприклад, з

завантажувачем (downloader), монтажником шкідливого ПО (dropper), запасним входом (backdoor), збирачем інформації (stealer). Троян є найпопулярнішим типом серед шкідливих програм. У квітні 2014 року PandaLabs зареєструвала «6 мільйонів нових шкідливих штамів», з них 78.92% – це трояни. [4]

Downloader (завантажувач) є частиною програмного забезпечення, яке завантажує інший шкідливий контент, наприклад, з веб-сайту, встановлює або запускає його. [3, с.39] [8, с.3]

Dropper (дроппер) це програма, яка розгортає інше шкідливе ПЗ в файлову систему. Dropper може виконувати процедури по установки та запуску шкідливих програм. [1, с.39-40]. Різниця порівняно з завантажувачем (Downloader) в тому, що Dropper вже містить шкідливий код в собі. [2]

Rootkit (руткіт) – програмне забезпечення, мета якого - приховування присутності інших шкідливих програм або дій. [1, с.4] Руткіт може приховувати активність, файли і процеси. Руткіти часто формують в поєднанні з функціональністю бекдорів.

Backdoor (запасний вхід, «чорний хід», бекдор) дозволяє отримати доступ до системи в обхід звичайних механізмів захисту доступу. [1, с.3] Бекдор використовується для отримання доступу до системи в будь-який час. Особливим видом бекдору є інструменти віддаленого адміністрування або RAT. [5, с.13] Вони дозволяють отримувати доступ до комп'ютера віддалено і використовуються для шкідливих і не зловмисних цілей. RAT дозволяють службі технічної підтримки віддалено усувати проблеми. Але RAT може використовуватися і зловмисно. Користувачі RAT можуть шпигувати за людьми через веб-камеру і намагатися налякати своїх жертв, контролюючи їх комп'ютер.

Spammer – шкідлива програма для розсилки спаму. Спамери використовують комп’ютер жертви, щоб відправити небажані повідомлення – так званий спам. [1, с.4] [3, с.40-41]). Спамери можуть відправляти свої повідомлення, використовуючи електронну пошту, SMS або пости і коментарі в інтернет-спільнотах.

Stealer (інформаційний злодій, збирач) – це шкідлива програма, яка зчитує конфіденційні дані з комп'ютера жертви і відправляє їх зловмисникові. Прикладами інформаційних збирачів є: клавіатурні шпигуни (keylogger), перехоплювачі (sniffer), збирачі хешів паролів, а також деякі види троянів. [1, с.4] Троян-викрадач переконує користувачів, що він доброякісна програма, для того, щоб споживач ввів конфіденційні дані. Прикладом є програма, яка стверджує, що може додати грошей на рахунок PayPal користувача, але, насправді, вона посилає облікові дані PayPal користувача на електронну адресу пошти зловмисника.

Botnet – це група різних машин зі встановленим backdoor, які отримують і виконують інструкції з одного сервера. [1, с.3]. Ботнет встановлюється без згоди власників комп'ютерів, і може бути використаний для виконання атак розподіленої відмови в обслуговуванні (DDoS) або для розсилки спаму.

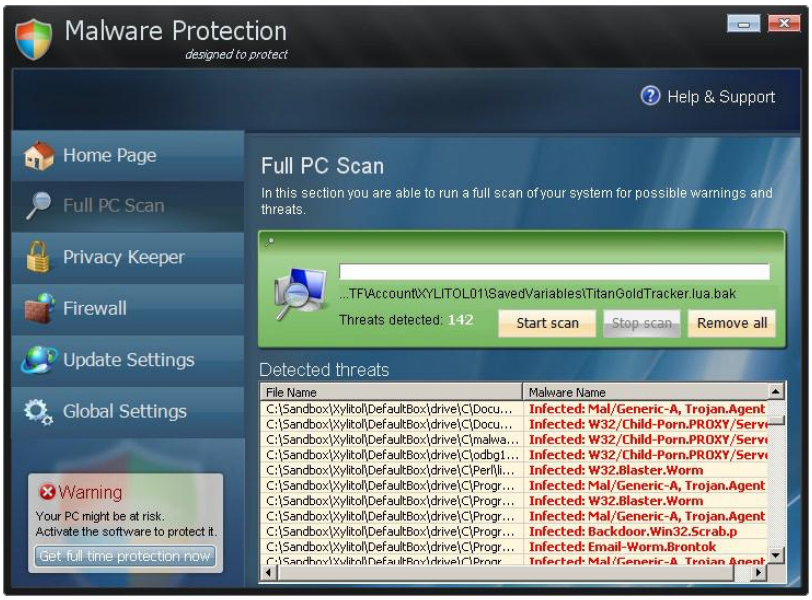

Рисунок 1.1 – Scareware, яке маскується як антивірусне програмне забезпечення

Scareware намагається обдурити користувачів, схиляючи до здійснення деякої покупки або лякаючи користувача. [1, с.4]). Типовим прикладом Scareware є програма, яка виглядає як антивірусний сканер і показує користувачеві підроблені попередження про шкідливі програми, які були знайдені у системі. Далі пропонується користувачеві купити дане програмне забезпечення для того, щоб видалити шкідливі програми. Приклад такої програми показаний на Рис. 1.1.

Часним випадком Scareware у веб є так зване Adware – шкідливі скрипти на сторінках, що містять нав’язливу рекламу або також фальшиві повідомлення, схиляючи користувачів до здійснення потенціально небезпечних дій.

Malware construction kit (конструктор шкідливих програм) – це ПЗ для генерації шкідливої програми або коду шкідливої програми, ґрунтуючись на налаштуваннях користувача. Конструктори шкідливих програм дозволяють людям без знань в програмуванні створювати свої шкідливі програми. Прості конструктори дозволяють змінити тільки деякі налаштування, наприклад, адреси електронної пошти або FTP-аккаунт, куди буде відправлятися знайдена інформація з комп'ютерів жертв. Більш складні комплекти використовують методи контр-аналізу і контр-виявлення, і можуть генерувати широкий спектр різних виконуваних файлів шкідливих програм. Рис. 1.2 показує приклад конструктора шкідливих програм, який має функції контр-аналізу.

Рисунок 1.2 – Приклад Malware Construction Kit

Virus розмножується рекурсивно, заражаючи, замінюючи інші програми або редагуючи посилання на ці програми, замінюючи його на код вірусу. [3, с.27, 36] Зазвичай вірус виконується, якщо користувач запустить заражений файл. Такий заражений файл називається хост-файл (з посиланням на термінологію яка використовується для біологічних паразитів). Віруси традиційно поширюються на

інші комп'ютери, за допомогою переносних засобів зберігання інформації таких, як, наприклад, USB флеш-накопичувачі, CD або DVD диски. Вірус називається «Germ» якщо він знаходиться в його первісному вигляді, до будь-якої інфекції. [3, с.39] Початкова установка коду Germ робиться за допомогою дроппера, після чого вірус може «повторити на собі» зараження. [3, с. 40]

Так звані «сплячі віруси» в пасивному стані знаходяться на машині, не заражаючи нічого: вони очікують тригера (повідомлення) або відповідної системи для поширення. [3, с.14] Протилежні сплячим вірусам - активні.

Worm (хробак) – це мережевий вірус, який для поширення в першу чергу використовує мережі. Як правило, хробаку не потрібен хост-файл, і він виконується сам, без необхідності втручання користувача. [3, с.36] Але є винятки – це хробаки, які поширюються за допомогою пошти, та їм необхідна взаємодія з користувачем. Черв'як є підкласом Вірусу. Згідно з доповіддю PandaLabs про новостворювані шкідливі штами, у квітні 2014 року серед них 7.44% відсотків займали віруси та 10.78% – хробаки. [4]

1.2 Стратегії зараження файлів

Зараження будь-якого файлу може привести до мальформації, тому що більшість типів заражень вимагають модифікації хост-файлом. У зв'язку з цим, систематизація знань про стратегії зараження файлів вірусами є дуже важливою, так як для побудови системи виявлення шкідливого ПЗ необхідно володіти інформацією про природу виникнення певних аномалій. Залежно від стратегії зараження іноді можливо видалити або відключити шкідливий код в зараженому файлі. Цей процес називається дезінфекція або лікування і виконується антивірусним програмним забезпеченням. Слід зазначити, що лікування не обов'язково відновляє файл до його оригінальному стану.

Розглянути нижче стратегії зараження працюють на більшості форматів виконуваних файлів, у тому числі і у веб-додатках.

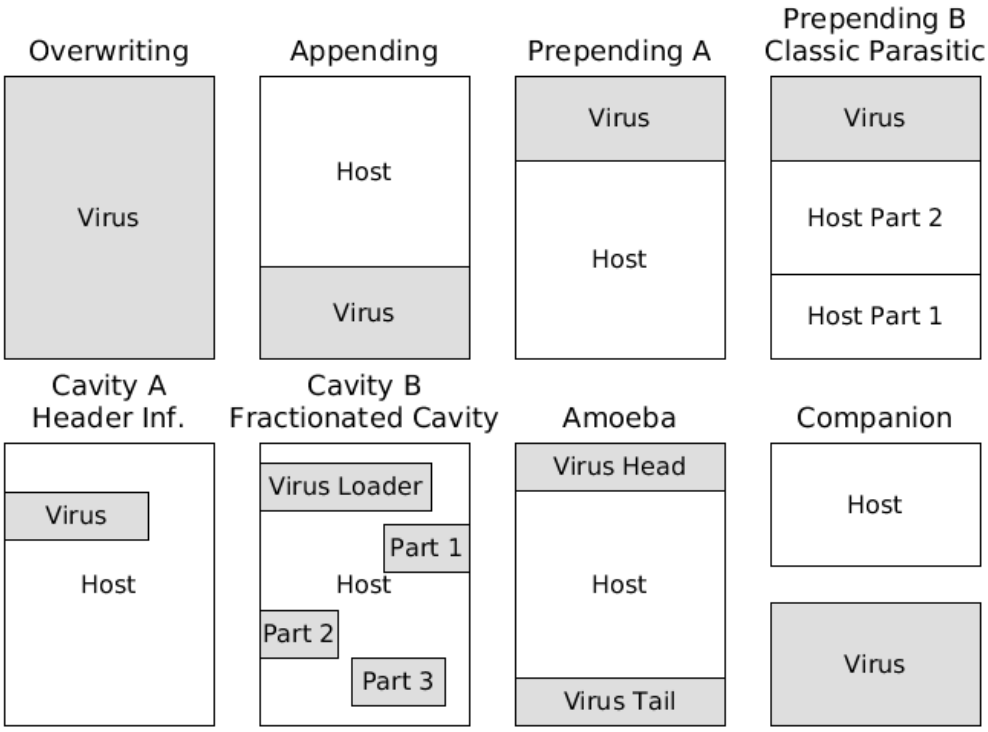

Рисунок 1.3 – Стратегії зараження файлів

Перезапис (Overwriting) – найпростіша стратегія вірусів. Вони шукають файли на диску і копіюють своє тіло, перезаписуючи оригінальний файл. [10, с.115] Зараження методом перезапису може завдати серйозної шкоди для системи, тому що відбувається знищення файлів, поверх яких вони записуються, і в даному випадку лікування неможливо. Проте, користувач дуже швидко розпізнає побічні ефекти ушкодження, тому це не дуже успішна стратегія зараження.

Перезаписуючий вірус тільки замінює початок файлу розміром даних, рівних розміру тіла самого вірусу. [3, с.116] Тож заражений файл зберігає розмір оригінального файлу. Перезаписана частину файлу у той час знищується.

Приєднання в кінці (Appending) – дана стратегія зводиться до приєднання тіла вірусу в кінець файлу (див. Рис. 1.3) і зміну інструкції переходу (jump) на початку файлу до початку тіла вірусу в кінці файлу. [3, с.117] Зазвичай вірус змінює інструкції переходу після того, як запише копію вірусу в кінець файлу, і таким чином користувач не стає свідком зараження. [3, с.118]

Деякі формати файлів визначають точку входу, яка є адресою точки старту виконуваного коду. Віруси, що приєднуються до цих файлів, змінюють адресу

точки входу виконуваного коду на адресу початку вірусного коду, або додають команду переходу відразу після точки старту на початок вірусного коду.

Зазвичай файли після зараження за допомогою цієї стратегії піддаються лікуванню.

Додавання до початку (Prepending) – у цьому випадку зараження відбувається прикріпленням тіла вірусу до початку файлу без пошкоджень останнього. [3, с.118-120] Тіло оригінального файлу залишається неушкодженим, тому лікування є можливим.

Щоб приховати зараження від користувача - вірус запускає оригінальний файл, наприклад, копіює його як тимчасовий файл на диск і використовує звернення до функції виклику system() для запуску оригінального файлу. [3, с.120]

Підтипом даного зараження є класичне паразитне зараження (див. Рис. 1.3). Вірус, який використовує цю стратегію, замінює початок файлу своїм тілом, а початок, який замінює, приєднує в кінець файлу. [3, с. 120-121].

Додавання в «печеру» (Cavity Infection) – відповідно до даної стратегії, вірус записує своє тіло найчастіше без перезапису «корисного» коду оригінального файлу. [3, с. 121] Відбувається перезапис тільки тих частин, містять нулі - в так званих «печерах» (caves) файлу [3, с.121]. Наприклад, «печера» може бути між заголовком і фактичним початком коду файлу. Якщо вірус інфікує цю область - це називається зараженням заголовка (див. «Cavity A» на Рис. 1.3). Віруси, які заражають єдину «печеру», повинні бути досить маленькими, що б відповідати розміру «печери» і не вийти за її область.

Такий вірус зазвичай зберігає оригінальну точку входу оригінального файлу і ставить елемент управління, що вказує на свій код після основного коду.

Підтипом даного зараження є фракціоноване зараження (див. «Cavity B» на Рис. 1.3), де вірус розбиває себе на частини і вписує фрагменти свого коду в кілька «печер». Перша частина містить код завантажувача, який є відповідальним за збірку частин в єдиний вірус в пам'яті під час виконання програми [3, с.122-123].

Відновлення вихідного файлу після зараження може бути складним, а іноді і не піддається відновленню, якщо перезаписані частини неможливо відновити.

Амебне зараження (Amoeba Infection) – ця стратегія зараження використовується рідко. [3, с.124] Вірус розділяє себе на 2 частини. «Голова» приєднується до початку файлу, а друга частина приєднується до краю. Якщо файл запускається - «голова» запускає другу частину в кінці файлу.

У деяких випадках вірус створює тимчасову копію оригінального файлу і запускає його в випадку запуску зараженого файлу.

Зараження компаньйона (Companion Infection) – у випадку даної стратегії вірус не змінює оригінальний файл. Він використовує порядок запуску виконуваних файлів операційної системи для свого запуску замість оригінального файлу.

Існує 3 види вірусів, що використовують цю стратегію:

1. «Регулярний компаньйон» розміщується в тому ж каталозі, що і основний файл, маючи ідентичне ім'я, але з іншим розширенням файлу. Якщо користувач вказує для запуску тільки ім'я файлу без розширення - «регулярний компаньйон» виконається замість основного файлу. Наприклад, файли з розширенням .COM шукаються перед файлами з розширенням .EXE на MS-DOS. Вірус «Компаньйон» може заразити файли .EXE, розміщуючи себе в тому ж каталозі і з тим же ім'ям файлу з розширенням .COM. [6, секція 2.2.1]

2. «Компаньон PATH (шлях)» користується оглядом каталогів, який використовується операційною системою для пошуку файлів. Операційна система зазвичай використовує змінну PATH при пошуку для визначення пошукових запитів виконуваних файлів. «Компаньйон PATH» має те ж ім'я, що і оригінальний файл, і розміщує себе в каталозі, який знаходиться «вище», ніж каталог оригінального файлу. [6, секція 2.2.2] Наприклад, віруси наслідують відкритим текам DLL. Додатки, які імпортують функції з DLL, можуть завантажити функціонал вірусу замість функціоналу справжнього DLL файлу.

3. «Поєднуючий компаньйон» користується макросами командного рядка, які визначає користувач, вони ж псевдоніми (aliases). [6, секція 2.2.2] Псевдоніми використовуються як ярлики для довгих командних послідовностей. Вірус може

створити псевдонім, який замінить загальну команду для запуску «Компаньйона». [6, секція 2.2.3].

Вірус «компаньйон» існує поруч з оригінальним файлом. Видалення файлу вірусу або псевдоніму в разі «поєднуючого компаньйона» зупинить зараження.

1.3 Особливості існуючих методів захисту та виявлення шкідливого ПЗ

Шкідливе ПЗ призначене для несанкціонованого вторгнення до обчислювальних ресурсів комп’ютерної системи або мережі з метою їх несанкціонованого використання та/або вчинення шкоди їх власнику за допомогою видалення, копіювання, підміни інформації. [7]

Як основні методи захисту від впливу шкідливого коду буде розглянуто такі методи, як сигнатурний аналіз і проактивний захист.

1.3.1 Методи сигнатурного виявлення

Сигнатура атаки (вірусу) – це характерні ознаки атаки або вірусу, що використовуються для їх виявлення. Більшість сучасних антивірусів, сканерів вразливостей та СВВ використовують «синтаксичні» сигнатури, взяті безпосередньо з тіла атаки (файла вірусу або мережевого пакета, що належить експлойту). Також існують сигнатури, засновані на поведінці або аномаліях - наприклад, занадто агресивне звернення до якого-небудь мережевого порту на комп'ютері. [8]

Сигнатури створюються в результаті кропіткого аналізу декількох копій файлу, що належать одному вірусу. Сигнатура повинна містити тільки унікальні рядки з цього файлу, що гарантують мінімальну можливість помилкового спрацьовування.

Виявлення, основане на сигнатурах, є основою функціонування переважної більшості антивірусних програм і СВВ. За цим методом програма проводить аналіз файлу, пакета чи фрагмента, звертається до так званого «словника» існуючих вірусів. Якщо виявлено відповідність до сигнатури шкідливого коду зі словника, антивірусне ПЗ може ініціювати виконання одної з наступних операцій:

– видалення інфікованого файлу;

– відправлення файлу до «карантину», що має на увазі зробити недоступним для виконання з метою недопущення розповсюдження вірусу;

– відновлення файлу, що має на увазі видалення вірусного фрагменту з тіла файлу.

Антивірусне ПЗ, розроблене на основі метода виявлення відповідності до сигнатури, як правило, обов'язково проводить аналіз файлу у тому випадку, якщо система створює, відкриває або відправляє файл через мережу Інтернет. Таким чином, віруси можуть бути виявлені одразу після занесення до комп'ютера та до того, як вони вчинять певної шкоди.

Розробка сигнатур важко піддається автоматизації. Незважаючи на масу досліджень, присвячених автоматичної генерації сигнатур, наростаючий поліморфізм вірусів і атак лишають синтаксичні сигнатури сенсу. Існують так звані «метаморфічні» віруси, в яких деякі фрагменти коду модифікуються або шифруються так, щоб неможливо було виявити збіг з визначенням в словнику вірусів.

Таким чином, основною перевагою методу сигнатурного аналізу є можливість визначення конкретної атаки з високою точністю та малою ймовірністю помилкового виклику. Проте, цей метод має і наступних ряд недоліків:

– неспроможність виявлення деякі нові атаки;

– неспроможність захисту від поліморфних та метаморфічних вірусів;

– необхідність регулярного та оперативного оновлення;

– необхідність ручного аналізу вірусів, відсутність автоматизованого підходу.

1.4 Постановка задачі

Об'єктом дослідження даної роботи є шкідливе ПО у веб-середовищі.

Предметом дослідження роботи є методи виявлення шкідливого програмного забезпечення.

У зв’язку з різким ростом об’єму коду, який підлягає фільтрації антивірусним ПО, існуючі підходи до детектування шкідливого коду породжують проблему, що є пов’язаною з необхідністю аналізу всіх оброблених комп’ютером даних. Це негативно впливає на продуктивність роботи системи. Зі зростом частоти появи нових вірусних скриптів, тож з’явилась необхідність більш частого оновлення баз даних вірусних сигнатур. І зараз у функціонуванні антивірусного ПО має місце висока частота хибних спрацьовувань. Ці проблеми призводять до необхідності впровадження альтернативних методів виявлення та захисту від шкідливих скриптів.

Метою дипломної роботи є розробка та впровадження програмного модулю для детектування шкідливого коду, де функціонал буде базуватися на методах сигнатурного аналізу та машинного навчання. Тож, метою переддипломної практики є розробка первинного функціоналу ПЗ, а саме – аналізування коду веб-сторінок за допомогою методу сигнатурного аналізу, а також первинний користувальницького інтерфейсу програми.

Для досягнення мети необхідно вирішити заступні задачі:

– проаналізувати предметну область індивідуальної задачі;

– дослідити існуючі методи детектування шкідливого ПО;

– побудувати UML-діаграми, що відображують майбутню архітектуру програмного модуля;

– здійснити обґрунтований вибір технологій та засобів для реалізації;

– розробити програмний модуль за допомогою обраних технологій та засобів;

– здійснити тестування розробленого програмного модулю.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Бізнес-логіка програмного модулю

Було розроблено наступний перелік бізнес правил:

– Програмний модуль повинен мати можливість обробити будь-який файл за поданим користувачем локальним шляхом.

– Якщо файлу по наданому шляху не існує, система повинна відреагувати певним чином.

– Після обробки файлу програмний модуль має вивести рішення щодо класу файлу: безпечний, підозрілий чи критичний.

– Якщо клас файлу не є безпечним, то система повинна здійснити первинний аналіз та вивести приблизний перелік загроз, що можуть міститися у файлі.

– На даний момент система має розглядати три типа загроз: загальні JS-віруси, фішінгові загрози та adware.

2.2 Специфікація вимог по ПЗ

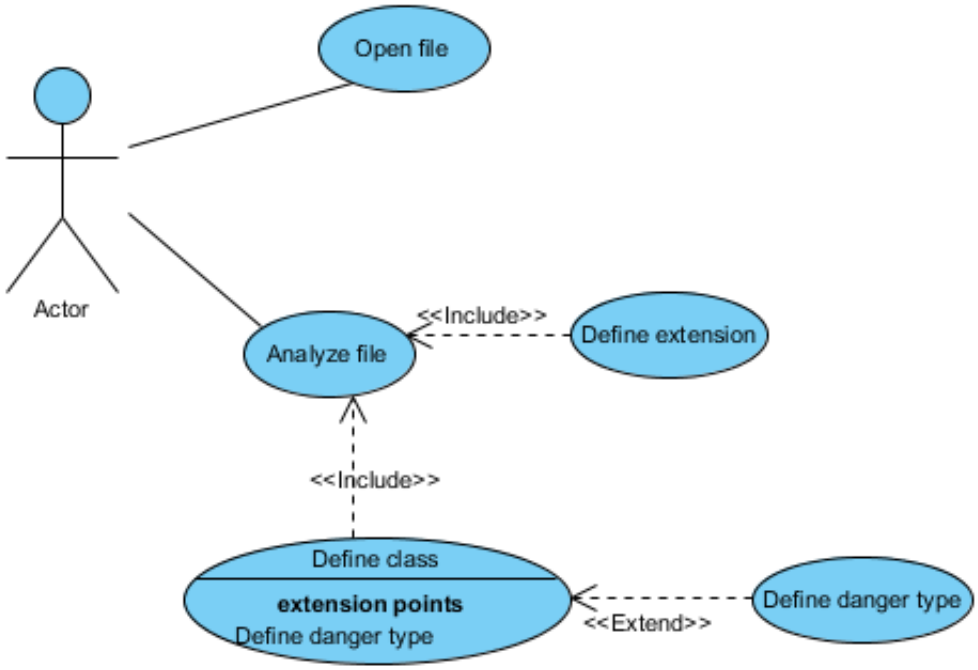

Рисунок 2.1 – Діаграма прецедентів

Основний функціонал, зображений на рисунку 2.1 на діаграмі прецедентів, має два основних прецедента: Відкрити файл та Обробити файл.

Другий має дві включені опції – Визначити розширення та Визначити клас, що в свою чергу має розширення Визначити тип загрози.

Більш детально описані специфікації вимог до програмного модулю розписано у Додатку 2 у форматі Software Requirement Specification по стандарту IEEE.

2.3 Діаграма IDEF0

Основна функція програмного модулю, якою є аналіз файлу на предмет шкідливості, відображена у форматі діаграми IDEF0 на рис. 2.2 та рис. 2.3.

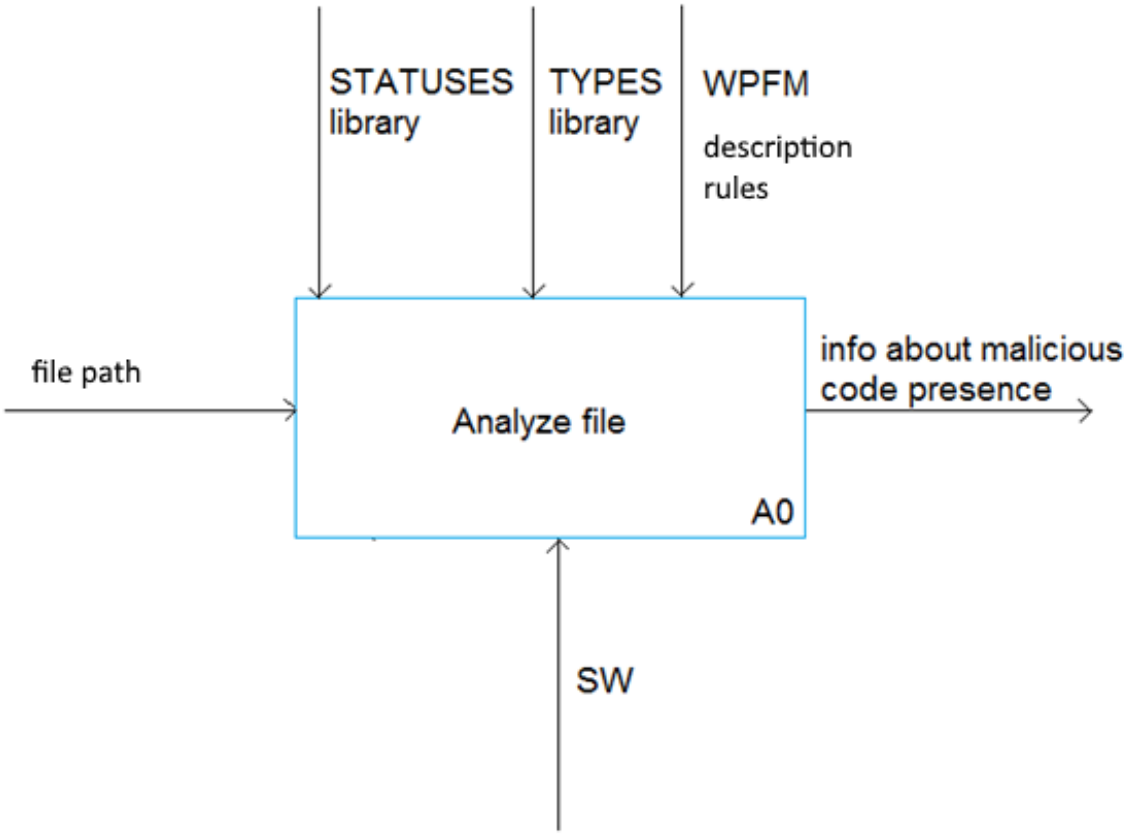

Рисунок 2.2 – IDEF0: Context Level

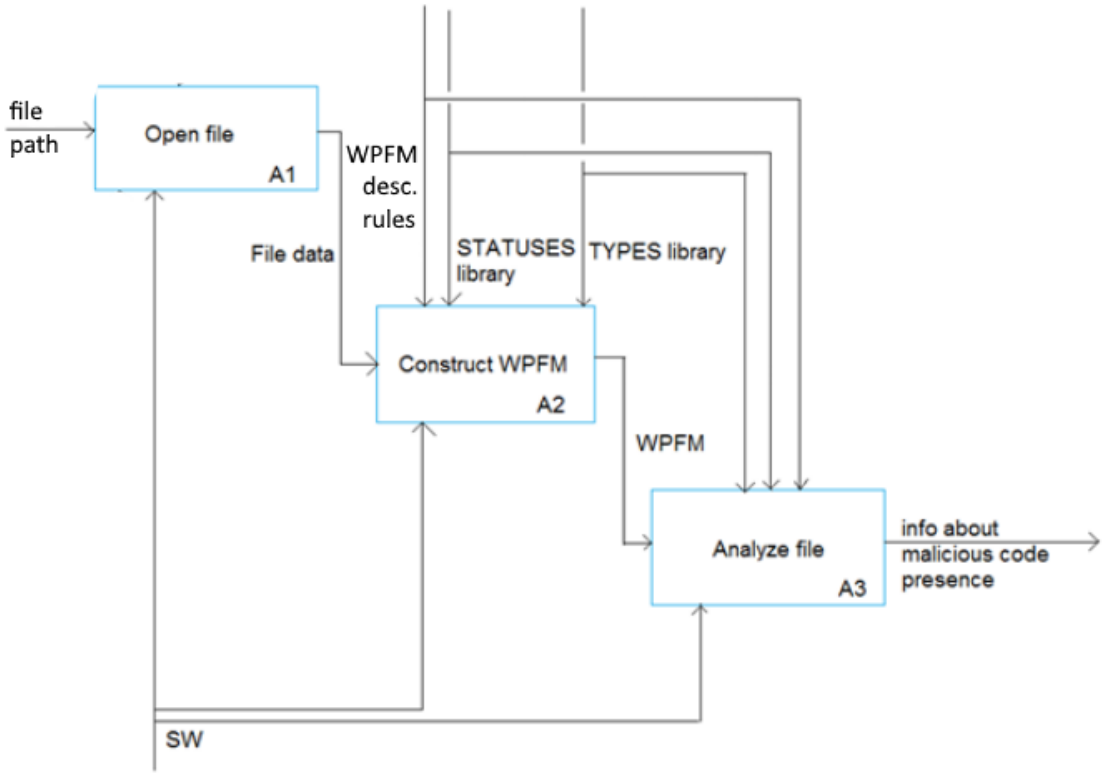

Рисунок 2.3 – IDEF0: Decomposition Level

Перший блок діаграми A1 відображає процес відкриття файлу. Вхідними даними для цього блоку є шлях до файла, що зчитується за допомогою модуля файлового потоку. Механізмами тут і в наступних блоках є ПЗ. Вихідними даними ж є зчитані дані файлу.

Другий блок діаграми A2 відображає конструювання формальної моделі веб-сторінки. Вхідними даними тут є дані файлу, управління здійснюється за допомогою правил опису моделі WPFM, що містить правила для опису формальної моделі файлу, та бібліотек, що містять можливі класи та типи файлів. Вихідними даними є формальна модель файлу.

Третій блок діаграми приймає як вхідні дані вихідні дані з попереднього блоку. Механізми та управління лишаються тими ж, що у попередньому блоку. Вихідними даними є інформація про присутність небезпечного коду у наданому файлі.

2.4 Діаграма розгортання

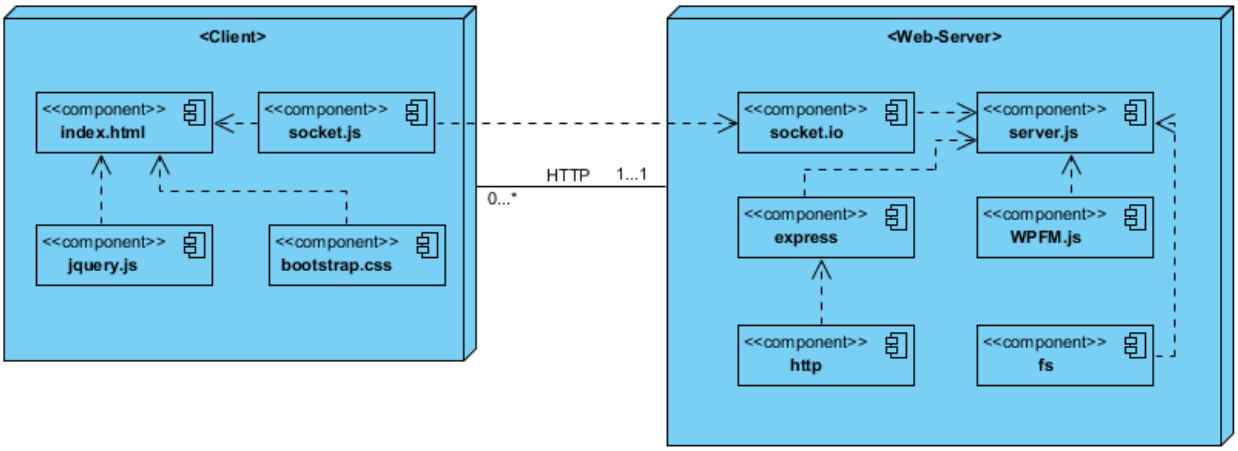

Рисунок 2.4 – Діаграма розгортання компонентів

Вузол «Client» має наступні компоненти:

– index.html – головна сторінка веб-додатку.

– jquery.js – бібліотека JavaScript, необхідна для реалізації користувацького інтерфейсу.

– bootstrap.css – бібліотека CSS, необхідна для реалізації користувацького інтерфейсу.

– socket.js – JavaScript сценарій, необхідний для реалізації комунікації клієнту та серверу.

Клієнт взаємодіє з веб-сервером за допомогою протоколу HTTP (конкретним веб-сервером, що застосований в проекті, є веб-сервер Express для Node.JS).

Вузол «Web-Server» має наступні компоненти:

– server.js – JavaScript сценарій, що описує процес роботи веб-сервера.

– express, socket.io, express, http, fs – модулі Node.JS, необхідні для реалізації функціоналу серверної частини.

– WPFM.js – модуль, що містить клас WPFM та відповідні функції.

2.5 Діаграма діяльності

Діаграму діяльності програмного модулю зображено на Рис. 2.5.

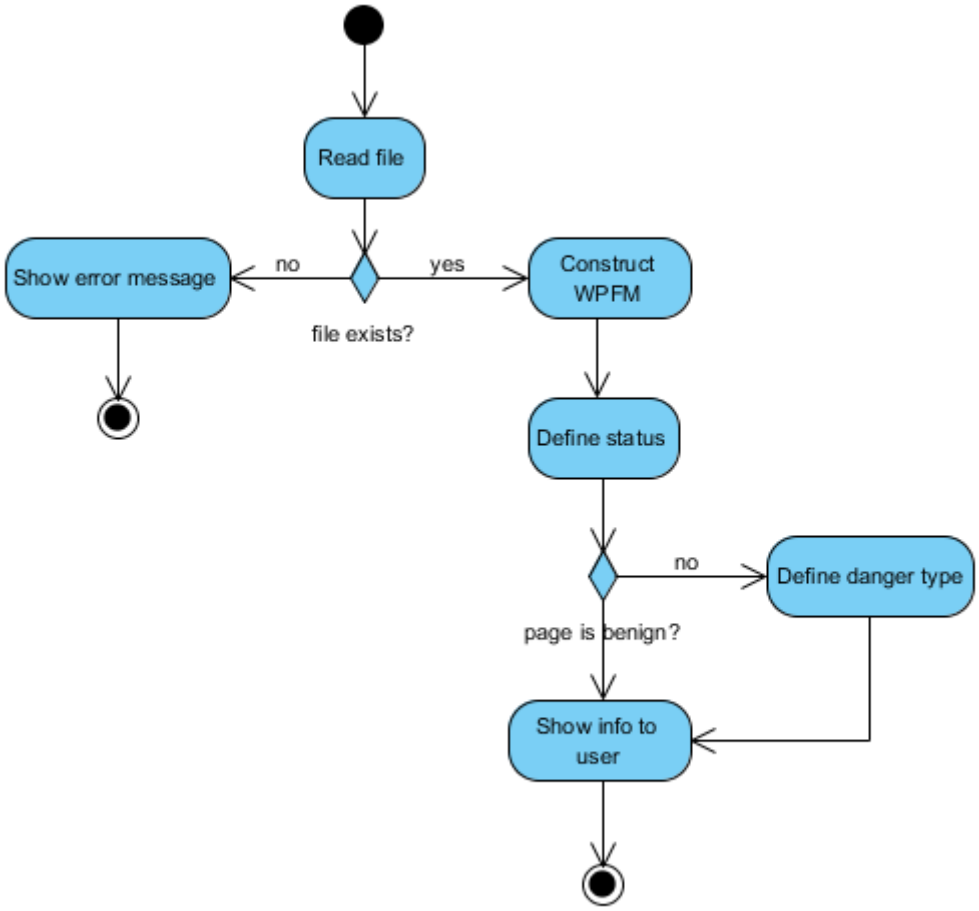

Рисунок 2.5 – Діаграма діяльності

2.6 Діаграма послідовності

Діаграму послідовності для програмного модулю зображено на Рис. 2.6.

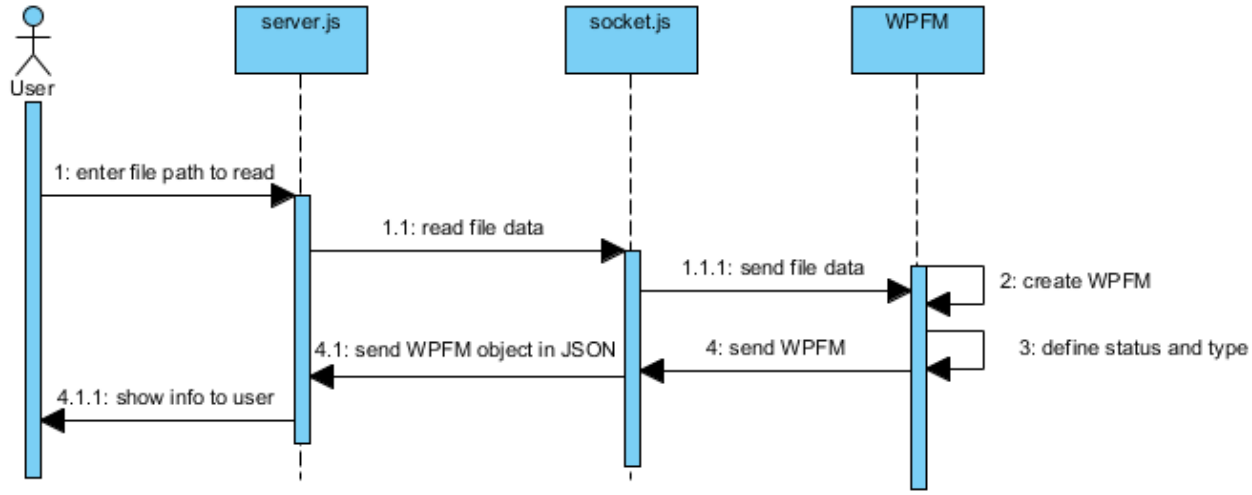

Рисунок 2.6 – Діаграма послідовності

2.7 Вибір технологій та засобів розробки ПЗ

Як основну мову програмування для імплементації програмного модулю було обрано JavaScript.

JavaScript (JS) — динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. Мова JavaScript також використовується для програмування на стороні сервера (подібно до таких мов програмування, як Java і C#), розробки ігор, стаціонарних та мобільних додатків, сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite), всередині PDF-документів тощо.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманний «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme. [9]

2.7.1 Серверна частина

Для реалізації серверної частини ПЗ було обрано Node.JS та npm як супутній пакетний менеджер.

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Платформа node.js перетворила мову JavaScript, що в основному використовувалась в браузерах на мову загального використання з великою спільнотою розробників

Node.js характеризується такими властивостями:

– асинхронна однопотокова модель виконання запитів;

– неблокуючий ввід/вивід;

– система модулів CommonJS;

– рушій JavaScript Google V8.

Для управління модулями використовується пакетний менеджер npm (node package manager). [10]

Конкретні модулі node, використані у реалізації серверної частини: express, http, socket.io, fs.

Комунікація клієнту та серверу реалізована за допомогою технології WebSockets.

2.7.2 Клієнтська частина

Юзерський інтерфейс реалізовано за допомогою HTML5 та CSS3, також було використано бібліотеку JQuery та фреймворк Bootstrap.

До складу робочої групи з HTML5 увійшли AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera та кілька сотень інших виробників.

Існує деяка плутанина щодо версійності, оскільки існують дві незалежні групи розробників — WHATWG та W3C.

28 жовтня 2014 консорціум W3C оголосив про надання набору специфікацій HTML5 статусу рекомендованого стандарту. Цікаво, що у цьому вигляді специфікації HTML 5.0 були сформовані ще два роки до того, після чого робота була зосереджена на проведенні тестування та оцінки сумісності доступних реалізацій. На час стандартизації HTML5 вже давно став стандартом де-факто і активно використовується у веб-застосунках. Фактичне затвердження стандарту лише формально поставило крапку в просуванні HTML5 і підтвердило повсюдність і коректність його реалізації.

Специфікації HTML5 не обмежуються тільки розміткою і включають в себе низку веб-технологій, котрі у сукупності формують відкриту Веб-платформу — програмне оточення для роботи крос-платформових застосунків, здатних взаємодіяти з обладнанням, і які підтримують засоби для роботи з відео, графікою і анімацією, що надає розширені мережеві можливості. [11]

Каскадні таблиці стилів (англ. Cascading Style Sheets або скорочено CSS) — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці веб-сторінок. Головна перевага блочної верстки — розділення змісту сторінки (даних) та їхньої візуальної презентації. [12]

Bootstrap — це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків.

Bootstrap — це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. Репозиторій з даним фреймворком є одним з найбільш популярних на GitHub. Серед інших, його використовують NASA і MSNBC. [13]

jQuery — популярна JavaScript-бібліотека з відкритим сирцевим кодом. Вона була представлена у січні 2006 року у BarCamp NYC Джоном Ресіґом (John Resig). Згідно з дослідженнями організації W3Techs, JQuery використовується понад половиною від мільйона найвідвідуваніших сайтів. jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день.

jQuery є вільним програмним забезпеченням під ліцензією MIT (до вересня 2012 було подвійне ліцензування під MIT та GNU General Public License другої версії).

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок. [14]

2.7.3 Вибір архітектурних патернів

Таблиця 1 – Обрані архітектурні патерни

Назва патерну	Опис	Обґрунтування
Декоратор	Призначений для динамічного підключення додаткової поведінки об'єкта. Надає гнучку альтернативу практиці створення підкласів з метою розширення функціональності.	Патерн декоратор розглядається я альтернатива об’єктно-орієнтованому підходу щодо динамічного додавання властивостей об’єкту WPFM в залежності від класу її небезпечності.
Ітератор	Дає можливість послідовно перебрати всі елементи складеного об'єкта, не розкриваючи його внутрішнього представлення.	WPFM (Формальна Модель Веб-сторінки) є складним об’єктом, що репрезентує множину елементів та характеристик веб-сторінки. Коли система сканує деяку веб-сторінку, відповідний модуль повинен ітеративно обробити кожен елемент множини, - що й обґрунтовує використання цього патерну.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Скріншоти процесу роботи програмного модулю

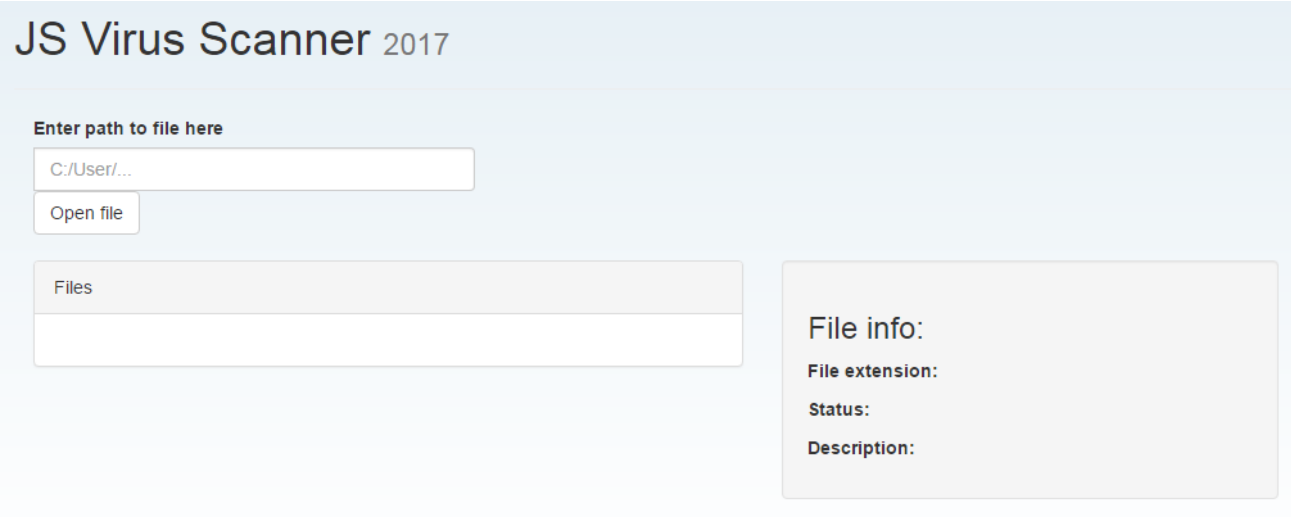

Рисунок 3.1 – Веб-додаток після запуску

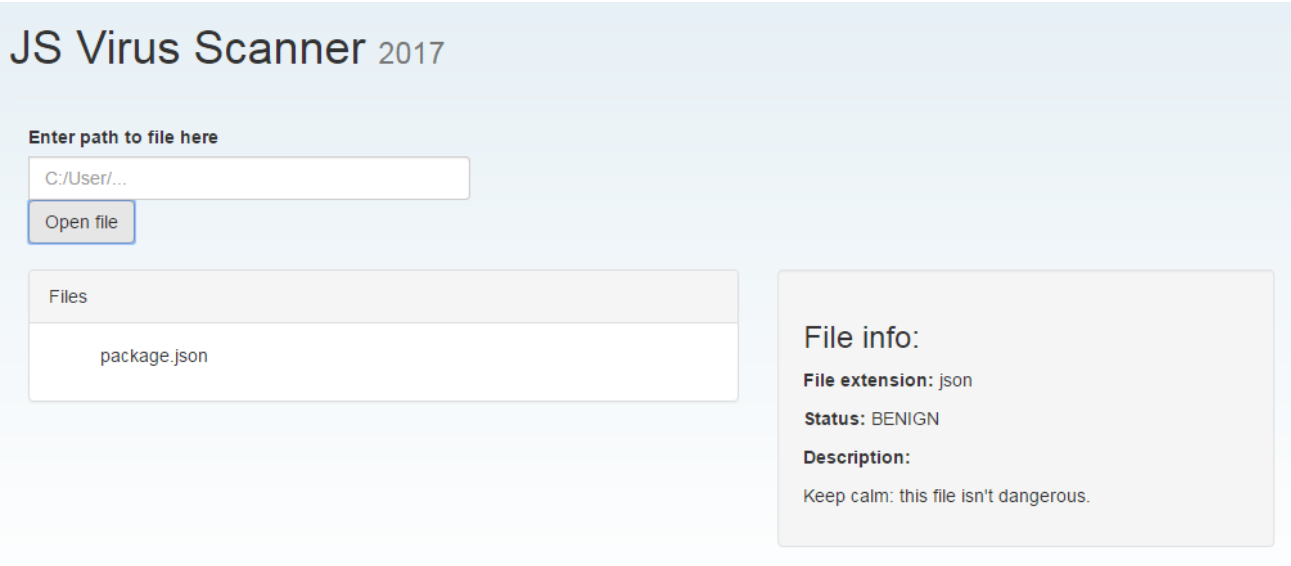

Рисунок 3.2 – Приклад сканування файлу: загрози не виявлено

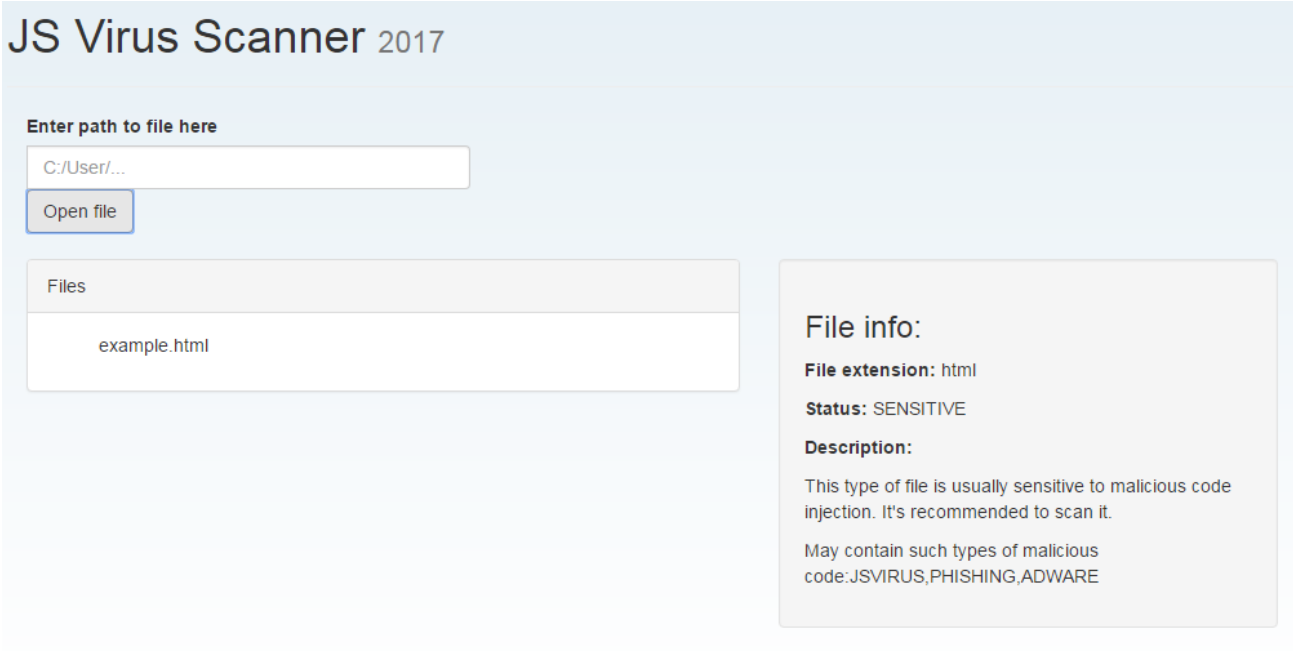

Рисунок 3.3 - Приклад сканування файлу: виявлено потенційно небезпечний файл

Задля коректної роботи веб-додатку необхідно запустити веб-сервер за допомогою команди “node server.js” у терміналі.

Програмний код деяких основних модулів розміщено у додатку 1.

3.2 Тестування програмного забезпечення

3.2.1 Серверна частина: модульне тестування.

Для модульного тестування процесу роботи сервера Express для Node.js було використано JavaScript бібліотеку Mocha та node-модуль Supertest.

Деякі тестові приклади зображено на Рисунку 3.4.

Процес тестування має бути запущений за допомогою команди “mocha -R spec spec.js” у терміналі, де spec.js – назва Javascript сценарію, де містяться модульні тести.

Результат тестів показано на Рисунку 3.5.

```
var request = require('supertest');
describe('loading express', function () {
  var server;
  beforeEach(function () {
    server = require('./server');
  });
  afterEach(function () {
    server.close();
  });
  it('responds to /', function testSlash(done) {
  request(server)
    .get('/')
    .expect(200, done);
  });
  it('404 everything else', function testPath(done) {
    request(server)
      .get('/foo/bar')
      .expect(404, done);
  });
});
```

Рисунок 3.4 – тестові приклади

```
$ mocha -R spec spec.js
  loading express
Example app listening at port 3000
    ✓ responds to /
    ✓ 404 everything else
  2 passing (109ms)
```

Рисунок 3.5 – Результат тестування

3.2.2 Клієнтська частина: функціональне тестування

Було здійснено тестування програмного забезпечення відповідно до наступних тестових прикладів:

Таблиця 2.1 – Тестовий приклад 1.1

Назва тесту:	#1.1
Опис тесту:	Перевірка можливості введення користувачем шляху до перевіряємого файлу

Передумови:	користувач має ввести коректний шлях до файлу
Очікуваний результат:	файл має бути считано та проаналізовано системою, інформація повинна бути виведена на сторінку веб-додатку
Актуальний результат:	після 10 тестових ітерацій помилок не було виявлено

Таблиця 2.2 – Тестовий приклад 1.2

Назва тесту:	#1.2
Опис тесту:	Перевірка можливості введення користувачем шляху до перевіряємого файлу
Передумови:	користувач має ввести некоректний шлях до файлу
Очікуваний результат:	повідомлення про помилку має бути виведено до терміналу
Актуальний результат:	після 10 тестових ітерацій помилок не було виявлено

Таблиця 2.3 – Тестовий приклад 2.1

Назва тесту:	#2.1
Опис тесту:	Перевірка виявлення потенційно небезпечного файлу
Передумови:	користувач має ввести коректний шлях до безпечного файлу
Очікуваний результат:	файл має бути считано та проаналізовано системою, користувачеві має бути повідомлено про відсутність загрози у наданому файлі
Актуальний результат:	після 10 тестових ітерацій помилок не було виявлено

Таблиця 2.4 – Тестовий приклад 2.2

Назва тесту:	#2.2
Опис тесту:	Перевірка виявлення потенційно небезпечного файлу
Передумови:	користувач має ввести коректний шлях до потенційно небезпечного файлу
Очікуваний результат:	файл має бути считано та проаналізовано системою, користувачеві має бути повідомлено про потенційно небезпечний файл
Актуальний результат:	після 10 тестових ітерацій помилок не було виявлено

3.2.3 Клієнтська частина: тестування сумісності

Веб-додаток було протестовано у таких веб-браузерах, як Google Chrome v.58, Mozilla Firefox v.57, Internet Explorer v.11 у середовищі ОС Windows 7.

Критичних помилок при тестуванні виявлено не було.

ВИСНОВКИ

У ході виконання даної роботи був проведений аналіз предметної області, проектування, реалізація та тестування програмного забезпечення.

У роботі було розглянуто існуючі види шкідливого ПЗ та методи його виявлення. Було детально розглянуто метод сигнатурного виявлення, його переваги і недоліки.

У результаті було розроблено набір первинних бізнес-правил проекту, а також основні необхідні для подальшої розробки моделі у вигляді діаграм UML, а саме: діаграма прецедентів або варіантів використання, діаграми IDEF0 контекстного рівня та рівня декомпозиції, діаграма розгортання компонентів, діаграма діяльності, діаграма послідовності. Було здійснено вибір необхідних для розробки архітектурних патернів програмування. Було здійснено огляд та обґрунтований вибір технологій та засобів для розробки програмного забезпечення.

Відповідно до попереднє розробленої бізнес-логіки та моделей було розроблено програмне забезпечення з використанням обраної мови програмування та обраних супутніх технологій.

Було здійснене тестування програмного забезпечення, а саме: модульне тестування для серверної частини, функціональне тестування та тестування сумісності для клієнтської частини

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1 Michael Sikorski and Andrew Honig. Practical Malware Analysis / M. Sikorski, A. Honig // No Starch Press, Inc., – 2012.

2 Microsoft Corporation. Naming malware 12207 [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx, 15.04.17

3 Peter Szor. The Art of Computer Virus Research and Defense. / P. Szor // Addison Wesley Professional, – 2005.

4 PandaLabs. Quarterly Report; April-June 2014 [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: http://www.pandasecurity.com/mediacenter/src/uploads/2014/07/Quarterly-Report-PandaLabs-April-June-2012.pdf, 22.04.17

5 Symantec Antivirus Research Center. Expanded threat list and virus encyclopedia. [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://securityresponse.symantec.com/avcenter/venc/data/cih.html, 01.05.17

6 Vesselin Bontchev. Possible Virus Attacks Against Integrity Programs and How to Prevent Them. [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://bontchev.nlcv.bas.bg/papers/attacks.html, 13.11.16

7 Mihai Christodorescu, Somesh Jha. Malware Detectors. Proceedings of the ACM SIGSOFT International / M. Christodorescu, S. Jha // Symposium on Software Testing and Analysis (ISSTA’04): Boston, Massachusetts, USA. – July 11-14, 2004. – 11 p.

8 Сигнатура атаки [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D1%82%D1%83%D1%80%D0%B0_%D0%B0%D1%82%D0%B0%D0%BA%D0%B8,01.10.16

9 JavaScript [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript

10 Node.js [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js

11 OpenNews: Консорциум W3C утвердил стандарт HTML 5 [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: http://www.opennet.ru/opennews/art.shtml?num=40954

12 Каскадні таблиці стилів [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS

13 Bootstrap [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Bootstrap

14 JQuery [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

15 Docs | Node.js [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://nodejs.org/en/docs/

16 Современный учебник JavaScript. Модули [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://learn.javascript.ru/modules

17 IEEE / IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. – IEEE Computer Society. – 1998.

ДОДАТОК 1

“server.js”:

```
var express = require('express');
var http = require('http');
var io = require('socket.io');
var wpfm = require('./WPFM/WPFM.js');
var fs = require('fs');

var app = express();
app.use(express.static('./public'));

var server =http.createServer(app).listen(8124);
io = io.listen(server);

io.sockets.on("connection",function(socket) {
        console.log('Socket.io Connection with the client established');

        socket.on("message", function(response) {
                response = JSON.parse(response);

                if (response.action == "openFile") {
                        fs.readFile(response.path, 'utf8', function (err,data) {
                                if (err) {
                                        return console.log(err);
                                } else {
                                        console.log(data);
```

```
                                    var ext =
response.path.slice(response.path.lastIndexOf('.') + 1, response.path.length);
                                    var currentPage = new wpfm(ext, data);
                                    console.log(currentPage);
                                    currentPage.defineStatus();

                                    if (currentPage.status.title != "BENIGN") {
                                        currentPage.defineType();
                                    }

                                    var fileInfo = {
                                        fileName: response.path,
                                        extension: currentPage.extension,
                                        status: currentPage.status.title,
                                        description: currentPage.status.desc,
                                        type: currentPage.type
                                    }
                                    socket.send(JSON.stringify(fileInfo));
                                }
                            });
                        }


                        var ack_to_client = {
                        data:"Server Received the message"
                        }
                        socket.send(JSON.stringify(ack_to_client));
                    });
                });
```

“WPFM.js”:

```
const STATUSES = {
    benign: {
        title: "BENIGN",
        desc: "Keep calm: this file isn't dangerous."
    },
    sensitive: {
        title: "SENSITIVE",
        desc: "This type of file is usually sensitive to malicious code injection. It's recommended to scan it.",
        exts: ['php', 'js', 'htaccess', 'html', 'htm', 'tpl', 'inc', 'css', 'txt', 'sql']
    },
    critical: {
        title: "CRITICAL",
        desc: "This type of file may contain malicious code. It is strongly recommended to scan it.",
        exts: ['cgi', 'pl', 'o', 'so', 'py', 'sh', 'phtml', 'php3', 'php4', 'php5', 'php6', 'php7', 'pht', 'shtml', 'susp', 'suspected', 'infected', 'vir']
    },
    warning: {
        title: "WARNING",
        desc: "The scaner found malicious code in this file!"
    }
}

const TYPES = {
    JSvirus: {
        title: "JSVIRUS",
```

```
                entries:
'<\s*script|<\s*iframe|<\s*object|<\s*embed|fromCharCode|setTimeout|setInterval|locati
on\.|document\.|window\.|navigator\.|\$\(this\)\.'
            },
            phishing: {
                title: "PHISHING",
                entries: '<\s*title|<\s*html|<\s*form|<\s*body|bank|account'
            },
            adware: {
                title: "ADWARE",
                entries:
'<\s*script|<\s*iframe|<\s*object|<\s*embed|fromCharCode|setTimeout|setInterval|locati
on\.|document\.|window\.|navigator\.|\$\(this\)\.'
            }
        }

        function WPFM(ext, content) {
            this.extension = ext;
            this.content = content;
            this.type = [];
            this.status = STATUSES.benign;

            //decorator: status
            this.defineStatus = function () {
                for (var status in STATUSES) {
                    if (STATUSES[status].exts) {
                        for (var i = 0; i < STATUSES[status].exts.length; i++) {
                            if (this.extension == STATUSES[status].exts[i]) {
                                this.status = STATUSES[status];
                                break;
```

```
                                }
                            }
                        }
                    }
                }

                //decorator: type
                this.defineType = function () {
                    for (var type in TYPES) {
                        if (TYPES[type].entries) {
                            var pattern = new RegExp(TYPES[type].entries);
                            if (this.content.search(pattern) != -1) {
                                this.type.push(TYPES[type].title);
                            }
                        }
                    }
                }
}

module.exports = WPFM;
```

ДОДАТОК 2

1.0. Вступ

1.1. Мета

Метою даного документа є представити докладний опис програмного модулю для виявлення потенційно шкідливих файлів. Він пояснить призначення і функції програмного модуля, його інтерфейси, обмеження, при яких вын повинен коректно працювати і як система буде реагувати на зовнішні подразники. Цей документ призначений як для стейкхолдерів так і розробників системи.

1.2. Призначення проекту

ПЗ буде представляти з себе веб-додаток для локального користувача комп'ютера або адміністратора сайту. Цей модуль ПЗ буде розроблений з ціллю максимізувати здатність майбутнього ПЗ прогнозувати рівень небезпеки конкретного файлу і дати користувачеві ПЗ кращий досвід сканування файлів на наявність шкідливого коду фрагментів.

Більш конкретно, цей додаток призначений для того, щоб дозволити користувачеві завантажити конкретний файл і просканувати його за допомогою веб-додатку, щоб передбачити рівень його небезпечності для подальшого сканування. Після кожного успішного сканування інформація про конкретний файл буде показана користувачеві. SW також містить сховище даних, що містить можливі статуси файлів відповідно до рівня небезпеки і можливі типи небезпечного коду у шкідливих файлах.

1.3. Глосарій

Термін	Визначення
Формальна Модель Веб-Сторінки (WPFM)	Клас, що описує колекцію важливих для ПЗ елементів та характеристик веб-сторінки або іншого поданого до веб-додатку файлу.
Веб-сторінка	Інформаційний ресурс, який можна переглянути у веб-браузері. Зазвичай, інформація веб-сторінки записана в форматі

	HTML, XHTML.
Шкідлива Веб-Сторінка	Веб-сторінка, що містить деякий потенціально шкідливий JavaScript-сценарій.
Шкідливий код	Ділянка коду в JavaScript-сценарії, розташованому на веб-сторінці, яка реалізує шкідливий для файлів або даних користувача механізм.
Software Requirements Specification	Документ, який повністю описує всі функції пропонованої системи і обмеження, відповідно до якмх він повинен працювати. Наприклад, цей документ.
Стейкхолдер	Будь-яка людина з інтересом у проекті, який не є розробником.
Юзер	Користувач ПК або Адміністратор сайту.

1.4. Посилання

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5. Огляд документу

Наступний розділ, розділ Загального Опису, у цьому документі дає огляд функціональних можливостей продукту. Він описує неформальні вимоги, і використовується для створення контексту для специфікації технічних вимог в наступному розділі.

Третій розділ, розділ Специфікації Вимог, у цьому документі написаний в першу чергу для розробників і описує деталі функціональності продукту в технічному плані.

Обидві частини документа описують один і той же програмний продукт в повному обсязі, але призначені для різних аудиторій.

2.0. Загальний Опис

2.1. Системне оточення

Програмний модуль має один тип дійової особи – користувач. Користувач має дві основні опції взаємодії з системою – Відкрити файл (Open File) та

Проаналізувати файл (Analyze File). Відповідні до цих опцій прецеденти описані далі у цьому розділі.

2.2 Функціональні вимоги

У цьому розділі описуються випадки використання для основної дійової особи веб-додатку.

2.2.1 Прецедент: Відкрити файл (Open File)

Короткий опис

Користувач запускає веб-додаток та вводить локальну адресу файла, який необхідно просканувати.

Початковий поетапний опис

До початку ініціювання цього прецеденту, користувач повинен запустити веб-додаток

1.Користувач вводить локальну адресу файла, який необхідно просканувати.

2. Користувач натискає кнопку “Open file”.

Xref: Розділ 3.2.1, Завантаження файлу

2.2.2 Прецедент: Проаналізувати файл (Analyze file)

Короткий опис

Веб-додаток аналізує наданий користувачем файл та виводить інформацію про присутність і тип загрози.

Початковий поетапний опис

До початку ініціювання цього прецеденту, користувач повинен запустити веб-додаток та ввести локальну адресу файла, який необхідно просканувати.

1. Веб-додаток зчитує та обробляє файл.

2. Інформація про присутність і тип загрози виводиться до юзерського інтерфейсу.

Xref: Розділ 3.2.2, Аналіз файлу

2.3 Опис користувачів

Даний веб-додаток має лише один тип користувача, що має доступ до усіх функцій модулю ПЗ – відкриття файлу та його аналіз.

2.4 Нефункціональні вимоги

Веб-додаток буде знаходитися на сервері з підключенням до високошвидкісного Інтернету. Фізична машина для використання буде визначатися користувачем. Програмне забезпечення, розроблене тут, передбачає використання такого інструменту, як Node.JS Web Server для з'єднання між користувальницьким інтерфейсом та логічними частинами програмного модулю. Швидкість з'єднання користувача буде залежати від використовуваного обладнання.

Для використання веб-додатку на клієнтському комп’ютері не потрібно проводити додаткового встановлення супровідного ПО. Для доступу для веб-додатку користувач має мати на локальному комп’ютері будь-який веб-браузер, що підтримує JavaScript сценарії (рекомендовано: Chrome, Mozilla) та мати підключення до мережі Інтернет. Взаємодія з користувальницьким інтерфейсом здійснюється за допомогою клавіатури та миші.

3.0. Специфікація вимог

3.1 Вимоги до зовнішнього інтерфейсу

Задля коректного функціонавуння програмний модуль не потребує зв’язків з абиякими зовнішніми інтерфейсами.

3.2 Функціональні вимоги

Логічна структура даних описана у розділі 3.3.1.

3.2.1 Завантаження файлу

Назва перецеденту	Завантаження файлу
XRef	Розділ 2.2.1, Відкрити файл
Трігер	-
Передумова	Користувач запускає веб-додаток та вводить локальну адресу файла, який необхідно просканувати.
Базовий шлях	Користувач вводить локальну адресу файла, який необхідно просканувати. Користувач натискає кнопку “Open file”, тим самим відсилає запит на відкриття файлу до сервера. За допомогою filestream сервер знаходить файл по наданому шляху та зчитує його.

Альтернативний шлях	У шагу 3, якщо користувач ввів шлях невірно, або якщо файлу по наданому шляху не існує, програмний модуль повинен вивести повідомлення про помилку у консоль браузера та консоль серверу.
Післяумова	Файл зчитано програмним модулем.
Шляхи виключення	-

3.2.2 Аналіз файлу

Назва перецеденту	Аналіз файлу
XRef	Розділ 2.2.1, Проаналізувати файл
Трігер	Файл зчитано програмним модулем.
Передумова	Файл завантажено успішно.
Базовий шлях	1. Програмний модуль конструює об'єкт WPFM, що представляє собою сукупність важливих характеристик наданого файлу. 2. Програмний модуль аналізує контент та розширення файлу за допомогою включених функцій. 3. Програмний модуль виносить висновок та виводить його користувачу разом із структурованою інформацією про оброблений файл.
Альтернативний шлях	-
Післяумова	Інформація про оброблений файл виведено до користувальницького інтерфейсу.
Шляхи виключення	-

3.3 Опис інших нефункціональних вимог

3.3.1 Захищеність

Сервер, на якому знаходяться дані веб-додатку свою власну безпеку для запобігання несанкціонованого запису / видалення доступу.

3.3.2 Масштабованість

Веб-додаток може бути запущений у будь-якій Операційній Системі, що має веб-браузер з підтримкою JavaScript та вихід в інтернет.