Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Комплексна бакалаврська дипломна робота на тему:
«Засіб поширення візуальної новели. Частина 1. Захищений застосунок»
08-20.БДР.016.00.000

Виконав: студент 4 курсу групи 1БС-19б
Спеціальності 125 Кібербезпека
_____ Яна НАСТАЛЕНКО

Керівник: к. т. н., доцент каф ЗІ
_____ Юрій БАРИШЕВ
« 16 » червня 2023 р.

Рецензент: к. т. н., доцент каф. ПЗ
_____ Наталя БАБЮК
« 16 » червня 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., професор
_____ Володимир ЛУЖЕЦЬКИЙ
« 16 » червня 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., професор
_____ Володимир ЛУЖЕЦЬКИЙ
«20» березня 2023 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Насталенко Яні Іванівні

1. Тема роботи: «Засіб поширення візуальної новели. Частина 1. Захищений застосунок»
   керівник роботи: Баришев Юрій Володимирович, к. т. н., доцент кафедри ЗІ, затверджено наказом ректора ВНТУ від 20 березня 2023 року №67.
2. Строк подання студентом роботи 17 червня 2023 р.
3. Вихідні дані до роботи:
   - архітектура – клієнт-серверна;
   - вид автентифікації користувачів – парольна;
   - нефункціональні вимоги – кросплатформений застосунок сумісний з Windows, Linux, MacOS;
4. Зміст текстової частини: Вступ. 1. Аналіз методів захисту комп'ютерних ігор. 2. Архітектура захищеного застосунку поширення візуальної новели. 3. Алгоритми роботи захищеного застосунку поширення візуальної новели. 4. Тестування захищеного застосунку. Висновки.
   Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: результати аналізу використання методів захисту відомих комп'ютерних ігор (плакат А4), узагальнена архітектура засобу поширення візуальних новел (плакат А4), , узагальнений алгоритм роботи захищеного застосунку (плакат А4), алгоритм роботи модуля клієнта (плакат А4), алгоритм роботи модуля сервера (плакат А4), алгоритм автентифікації (плакат А4), результати блокового тестування (плакат А4), результати

тестування автентифікації користувачів (плакат А4) резул
навантажувального обфускованого коду (плакат А4), результати тестув
захищеного застосунку (плакат А4)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдан прийня
1	Баришев Ю. В., к. т. н., доц. каф. ЗІ	20.03.23	08.05.23
2	Баришев Ю. В., к. т. н., доц. каф. ЗІ	20.03.23	15.05.23
3	Баришев Ю. В., к. т. н., доц. каф. ЗІ	20.03.23	21.05.23
4	Баришев Ю. В., к. т. н., доц. каф. ЗІ	20.03.23	10.06.23

7. Дата видачі завдання 20 березня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів комплексної бакалаврської бакалаврської роботи	Строк виконання етапів роботи	Приміт
1	Аналіз завдання. Вступ	20.03.23-26.03.23	
2	Аналіз джерел за напрямком бакалаврської дипломної роботи	27.03.23-09.04.23	
3	Розробка рішень, моделей, алгоритмів	10.04.23-23.04.23	
4	Практична реалізація, моделювання, експериментування, результати	24.04.23-21.05.23	
5	Висновки	22.05.23-24.05.23	
6	Оформлення пояснювальної записки	25.05.23-31.05.23	
7	Попередній захист БДР	01.06.23-15.06.23	
8	Виправлення зауважень, підготовка ілюстративного матеріалу	16.06.23-19.06.23	
9	Представлення БДР до захисту, рецензування	20.06.23-23.06.23	

Студент _____ Яна НАСТАЛЕН

Керівник роботи _____ Юрій БАРИШ

АНОТАЦІЯ

Комплексна бакалаврська дипломна робота складається з 91 сторінок формату А4, на яких є 31 рисунок, 1 таблиця, список використаних джерел містить 53 найменування.

Комплексна бакалаврська робота присвячена розробці засобу для поширення візуальних новел, а саме розробці захищеного застосунку. В результаті аналізу поширених загроз для комп'ютерних ігор, засобів захисту прикладних програм та комп'ютерних ігор, порівняльного аналізу методів захисту найвідоміших ігор поставлено задачу, розроблено архітектуру захищеного застосунку та запропоновано метод його захисту. На основі архітектури було розроблено алгоритми роботи застосунку та його модулів. Після розробки архітектури мовою програмування Java було написано програмну частину, здійснено обфускацію та проведено тестування застосунку.

Ключові слова: захист комп'ютерних ігор, піратство, візуальна новела, обфускація, автентифікація, Java.

ABSTRACT

The comprehensive bachelor's thesis consists of 91 A4 pages, which have 31 figures, 1 table, the list of used sources contains 53 items.

The comprehensive bachelor's thesis is devoted to the development of a tool for the distribution of visual novels, namely the development of a protected application. As a result of the analysis of common computer games threats, protection tools for applications and computer games, a comparative analysis of the protection methods of the most famous games, a task was set, the architecture of a protected application was developed and a method of its protection was proposed. Based on the architecture, the algorithms of the application and its modules were developed. After developing the architecture, the software part was written in the Java programming language, obfuscation was performed, and the application was tested.

Keywords: security of computer games, piracy, visual novel, obfuscation, authentication, Java.

ЗМІСТ

ВСТУП

Ігрова індустрія – це багатомільярдний сектор в плані фінансів та багатомільйонний в плані користувачів. І тому дуже актуальною залишається тема захисту комп'ютерних та відео ігор від таких загроз, як несанкціоноване копіювання, шахрайство, хакінг, компрометація конфіденційних даних, таких, як персональні дані гравців.

Збереження безпеки ігор у сучасності демонструє високий рівень захисту та вимагає значних фінансових інвестицій. Тим не менш, незаперечною реальністю залишається те, що поява кожної щойно випущеної гри створює непереборний виклик і є справою честі для деяких ентузіастів, які палко сприймають злом таких ігор як свідчення їх хакерської майстерності. Це повторюване явище підкреслює необхідність постійної пильності та постійного адаптування та посилення заходів безпеки для протидії таким спробам.

Багатогранний характер захисту гри передбачає впровадження стійких заходів безпеки, спрямованих на зміцнення архітектури гри, пом’якшення вразливостей і збереження конфіденційності особистих даних гравців. Ці заходи охоплюють протоколи шифрування, складні механізми автентифікації, захищені серверні архітектури та суворий контроль доступу. Ретельно впроваджуючи такі засоби захисту, розробники ігор прагнуть створити надійну систему безпеки, здатну запобігти несанкціонованим вторгненням, шахрайським діям і компрометації конфіденційних даних.

Підсумовуючи, фінансове значення ігрової індустрії в поєднанні з її великою базою користувачів підкреслює нагальну потребу в захисті комп’ютерних і відеоігор від таких загроз, як несанкціоноване копіювання, шахрайські дії, спроби злому та порушення компрометації даних гравців. Незважаючи на значні інвестиції для зміцнення безпеки ігор, привабливість хакерства залишається постійною проблемою, яка потребує постійної адаптації та посилення захисних заходів. Саме тому захист програм, які поширюють контент комп’ютерних ігор є актуальним.

Отже, метою даної роботи є покращення захисту комп’ютерних ігор від основних загроз на прикладі засобу для поширення візуальних новел.

Предметом дослідження є методи та засоби захисту комп’ютерних ігор.

Метою даної комплексної бакалаврської дипломної роботи є покращення захисту комп’ютерних ігор шляхом розробки засобу для поширення візуальних новел.

Для досягнення мети потрібно розв’язати такі задачі:

- проаналізувати загрози комп'ютерним іграм;
- проаналізувати методи захисту прикладних програм;
- проаналізувати засоби захисту комп'ютерних ігор;
- розробити архітектуру засобу, враховуючи особливості захисту;
- розробити алгоритм роботи захищеного застосунку;
- реалізувати основні функції застосунку;
- реалізувати функції захисту;
- провести тестування застосунку.

Результати роботи були апробовані на 2 конференціях з публікацією тези доповідей:

- дослідження способів захисту візуальних новел від несанкціонованого доступу – LII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2023) [1];
- дослідження ефективності захисту програмного коду шляхом лексичної обфускації – LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2022) [2].

За результатами роботи було отримано 2 свідоцтва авторського права на твір (комп’ютерну програму):

- ObfusLex [3];
- KERBEROS [4].

1 АНАЛІЗ МЕТОДІВ ЗАХИСТУ КОМП'ЮТЕРНИХ ІГОР

1.1 Аналіз загроз комп'ютерним іграм

Оскільки існує велике різноманіття комп'ютерних ігор, то і загроз їм теж досить велика кількість.

Відеоігри не є захищеними від кіберзагроз, і навіть лише кілька основних створюють великі ризики для всієї ігрової індустрії. До цих загроз можна віднести такі [5]:

- захоплення облікових записів;
- DDoS-атаки;
- шахрайство та хакерство;
- несанкціоноване копіювання (піратство);
- програми-вимагачі;
- фішингові атаки;
- зловмисне програмне забезпечення в модах і програмному забезпеченні сторонніх розробників;
- крадіжка віртуальної та реальної валюти;
- крадіжка даних.

Захоплення облікових записів – хакери отримують несанкціонований доступ до облікових записів гравців, і доволі часто це відбувається за допомогою фішингових атак або підміни облікових даних. Після отримання контролю хакери отримують доступ до особистої інформації, ігрових предметів та подібного, і відповідно, як приклад, можуть їх продати, або ж використати у інших неетичних цілях [6].

Розподілені атаки на відмову в обслуговуванні (DDoS) спрямовані на ігрові сервери – вони заповнюють їх надзвичайною кількістю трафіку, чим спричиняють збій або блокують доступність серверу. Таким чином атаки цього виду порушують ігровий процес, чим унеможливлюють гру для всіх легальних гравців. Тривалий час основною загрозою для гри Apex Legends були DDoS-атаки, які завдавали

вагомої шкоди її розробникам, не дозволяючи іншим гравцям користуватись серверами гри, а оскільки гра є серверною і використання офлайн режиму є обмеженим, то і гравці елементарно не могли грати в неї [7].

Шахрайство і хакерство – одна із найбільших проблем розробників у ігровій індустрії, оскільки ігри постійно піддаються «нечесним» діям шахраїв, отримуючи великі збитки. Так, наприклад, хакери постійно зламують ігрові ресурси: текстури, карти, внутрішньоігрові валюти. Здебільшого потерпають від шахрайства мультиплеєрні ігри в мережі, де хід гри одного гравця залежить від інших. Проте, ігри офлайн також мають чіт-коди, які вважаються шахрайством та полегшують ігровий процес. Найбільш відомі ігри, такі як Apex Legends, Dota 2, CS:GO та League of Legends, Valorant, Fortnite, The Sims 3, GTA 5, хоч і відносяться здебільшого до різних жанрів, та однаково потерпають від дій шахраїв [8].

Ще однією із найбільших проблем, від якої потерпає вся ігрова індустрія, це – боротьба із піратством. Саме цій загрозі приділяють досить велику частку уваги та зусиль на її подолання, проте досі геймери з усього світу незаконно копіюють, поширюють, встановлюють і використовують ігри, чим приносять їх розробникам збитки. Боротьба між розробниками та хакерами триває вже протягом кількох десятиліть [9]. Так, найбільш відомими іграми, які постійно копіюються несанкціоновано та поширюються піратами, є: The Witcher (всі частини), FIFA (вся серія), The Sims (всі частини), Grand Theft Auto V, Call of Duty (вся серія), Assassin’s Creed (вся серія), The Elder Scrolls V: Skyrim, Far Cry (вся серія), Fallout 4, Need for Speed (всі частини) [10].

Ще один вид атак – атаки програм-вимагачів. Вони можуть впливати на розробників і видавців ігор. Зловмисники можуть проникнути в процес розробки або канали розповсюдження, щоб впровадити шкідливий код у гру, зашифрувавши цінні дані. Далі вони діють за класичним сценарієм програм-вимагачів: вимагають викуп, щоб відновити доступ або запобігти випуску скомпрометованих ігрових ресурсів [11]. Так, наприклад, у січні 2023 року відомий розробник ігор Riot Games отримав на електронну пошту лист із вимогою викупу, що призвело до передчасного завершення робочого дня та перешкодило запланованому випуску

патчів для кількох ігор [12]. Компанія зазначила, що дані гравців не було скомпрометовано, проте згодом стало відомо, що наслідки все ж були не надто приємними: хакерам дійсно вдалось викрасти конфіденційний вихідних код із двох ігор, одна із яких є доволі відомою – League of Legends [12].

Фішингові електронні листи або веб-сайти, націлені на геймерів, видають себе за законні платформи або ігрові магазини аби обманом змусити надати свої облікові дані або особисту інформацію, яка може бути використана для захоплення облікових записів або інших незаконних дій.

Що стосується зловмисного ПЗ в модах та ПЗ сторонніх розробників, то вони також становлять ризики, хоч і можуть покращити ігровий досвід. Зловмисники створюють модифікації або програмне забезпечення, що містить зловмисне ПЗ, яке може заражати пристрої гравців, викрадати особисту інформацію або порушувати безпеку систем. Однією із найбільш відомих ігор з великою спільнотою модифікаторів є Elder Scrolls V: Skyrim. У 2019 році стало відомо про відкритий мод під назвою Open Cities, що містив зловмисне ПЗ, яке могло відкривати бекдор у системах користувачів та дозволяло несанкціонований доступ або встановлення додаткових шкідливих програм [13].

Крадіжка віртуальної та реальної валюти у відеоіграх – це несанкціоноване придбання або крадіжка внутрішньоігрової або реальної валюти, пов’язаної із грою. Внутрішньоігрову валюту викрадають частіше всього, аби продати її на сторонніх платформах дешевше, ніж вона коштує в самій грі. Справжня крадіжка валюти відбувається, коли хакер отримує доступ до платіжної інформації гравців або використовують вразливі місця в платіжних системах, які можуть бути пов’язані із грою. У більшості атак зловмисники створюють підроблені веб-сайти або платіжні портали і оманою змушують гравців ввести свої платіжні дані, або ж атаки «людина посередині»: хакери перехоплюють зв’язок між гравцями та платіжною системою гри, отримуючи доступ до конфіденційної інформації під час процесу транзакції [14].

Також ігрові компанії збирають величезну кількість даних гравців, і часто в цей перелік входять платіжні дані зі всіма картками гравців, якими вони

розраховувались коли-небудь у грі. Ця інформація може бути скомпрометована, що призведе до її викрадення або ж несанкціонованого доступу до облікових записів.

Отже, найбільші загрози комп'ютерним іграм – це захоплення облікових записів, DDoS-атаки, шахрайство та хакерство, програми-вимагачі, фішингові атаки, зловмисне програмне забезпечення в модах і програмному забезпеченні сторонніх розробників, крадіжка віртуальної та реальної валюти і крадіжка даних. Саме вони приносять найбільше збитків компаніям-розробникам, і від яких вже декілька декад із помірним успіхом розробники комп'ютерних ігор намагаються створити надійний захист. Проте, найбільшою проблемою розробників ігор все ж є несанкціоноване копіювання (піратство) та шахрайство (злам карт, текстур, внутрішньоігрових валют і т.п.).

1.2 Аналіз методів захисту прикладних програм

Оскільки питання захисту прикладних програм уже тривалий час залишається піднятим, то і способи їх захисту з'являються з досить великою інтенсивністю, хоч і не всі вони є стійкими, та більшість одразу ламаються. Проте, є перелік способів, перевірених часом, розробниками та хакерами.

Варто розрізняти вбудований та навісний захист [15].

Використання наступних способів у комплексі можуть дати доволі високий рівень захищеності програмного застосунку [16]:

- обфускація коду;
- автентифікація та авторизація;
- валідація введених даних;
- контроль доступу;
- виявлення втручання та реагування;
- методи захисту від налагодження та втручання (перевірка цілісності коду, антидампінг, механізми захисту від перехоплення);
- постійні оновлення.

Перераховані вище способи захищають від несанкціонованого доступу та аналізу, копіювання і від реверсивної інженерії коду.

Обфускація – це перетворення коду програми (вихідного або двійкового) в більш складну для розуміння форму, яка зберігає його функціональність, але ускладнює розуміння алгоритму та проведення аналізу при попередній декомпіляції [17]. Цей метод захисту буває декількох видів: лексична, обфускація даних та графа потоку керування – відповідно кожен із них забезпечує свій рівень захисту, і, відповідно, лише один метод, наприклад, лексичну обфускацію, не варто використовувати [2, 18]. Проте, при захисті коду методом обфускації також зменшується рівень розуміння його розробниками та ускладнює логування, що може перешкоджати подальшому виявленню та виправленню помилок. Для здійснення обфускації існує досить велика кількість застосунків, розрахованих на різні мови програмування, наприклад: NET Reactor, ProGuard, Allatori [19, 20].

Автентифікація та авторизація є невід'ємною частиною захисту ПЗ та є першою його лінією. Автентифікація запобігає отриманню несанкціонованого доступу неавторизованим особам. Надійними механізмами автентифікації вважаються надійні паролі, багатофакторна автентифікація або біометрія [21]. Авторизація у свою чергу доповнює автентифікацію тим, що контролює, на які дії чи доступ до ресурсів можна користувачу надати права. Некоректна авторизація, або ж її відсутність може призвести до таких наслідків, як несанкціоновані модифікації, витоки даних або зловмисні дії у ПЗ (наприклад, впровадження шкідливого коду) [22]. Також автентифікацію та авторизацію можна інтегрувати із системами ліцензування ПЗ з метою запобігти несанкціонованому розповсюдженню та піратству [23].

Валідація введених даних здійснюється із метою запобігти атакам впровадження коду, таких, як SQL-ін'єкції, переповненню буфера, запобігти атакам на відмову в обслуговуванні (DDoS), зловмисному завантаженню файлів, та покращенню взаємодії із користувачем. Для ефективної перевірки введених даних зазвичай її проводять як на стороні клієнта, так і на стороні сервера, також

перевіряють дані щодо очікуваних форматів та типів даних, обмежень (зазвичай із використанням відповідних бібліотек) [24].

Контроль доступу – фундаментальний аспект захисту ПЗ, він передбачає керування доступом до ресурсів і правами користувачів на їх модифікацію та власне можливості [25]. Контроль доступу забезпечує виконання правил авторизації користувачів, запобігає несанкціонованому доступу до конфіденційних даних, сприяє зменшенню внутрішніх загроз (наприклад, зловживанню привілеями). Також контроль управління доступом допомагає ПЗ дотримуватись галузевих норм і правових вимог та дозволяє проводити аудит та моніторинг, відстежуючи дії користувачів (включаючи всі спроби доступу та внесені зміни).

Виявлення втручання та реагування на них також є важливою частиною захисту ПЗ. Механізми виявлення втручання починаються з перевірки цілісності програмних компонентів – це досягається за допомогою контрольної суми, геш-функцій або цифрових підписів [26]. Крім того, важливою складовою є методи захисту від несанкціонованого доступу та безпечна конфігурація і середовище. Відповідь на втручання забезпечується веденням журналів подій у системі та сповіщенням при виявленні несанкціонованих змін, ізоляцією втручань, виправленням та їх усуненням [27]. Ці процеси повинні завершуватись розслідуванням інциденту та супроводжуватись постійним моніторингом і вдосконаленням системи.

За допомогою постійних оновлень відбувається управління вразливостями – слабкими місцями ПЗ, якими можуть скористатись зловмисники для отримання несанкціонованого доступу чи порушення роботи системи. Постійні виправлення та оновлення допомагають їх вчасно усунути. Аналогічно відбувається виправлення помилок. Також постійне оновлення допомагає захищатись від експлойтів і зловмисного ПЗ [28].

Таким чином було визначено найбільш застосовувані та стійкі методи захисту.

1.3 Аналіз засобів захисту комп'ютерних ігор

Засоби захисту комп’ютерних ігор варто розглядати в контексті їх впровадження, тобто розділити їх на вбудовані та навісні [15].

Вбудовані засоби захисту – це засоби, застосовані на етапі розробки гри, навісний захист – це метод, який використовується для захисту комп’ютерних ігор від несанкціонованого копіювання, розповсюдження та використання [15].

Так, до вбудованих можна віднести такі [15]:

- використання серверів;
- використання безпечних платіжних шлюзів;
- використання систем виявлення шахрайства;
- вимоги до надійних паролів;
- багатофакторна автентифікація;
- процеси відновлення облікових записів.

До навісних засобів відносяться такі [15]:

- активація та ліцензування;
- онлайн-автентифікація;
- цифрові підписи;
- обфускація коду;
- перевірка цілісності;
- заходи проти налагодження;
- контрзаходи проти піратства;
- обмеження офлайн-режиму;
- обмежене встановлення (за допомогою ліцензійних ключів);
- захист завантажуваного вмісту (Downloadable Content, DLC).

Всі вищеперераховані методи є окремими інструментами, що входять до засобів керування цифровими правами (Digital rights management, DRM) [29].

Методи навісного захисту відрізняються у різних видавців і розробників ігор. Деякі реалізації бувають занадто нав'язливими, вимагаючи постійного онлайн-

з'єднання або додаткових програмних компонентів, тоді як інші зосереджуються на більш зручному підході для користувача. Зазвичай для розробників важливим завданням є знайти баланс між ефективним захистом і зручністю для користувача.

Однією із найпопулярніших технологій захисту від втручання та технологією DRM є Denuvo. Саме ця технологія використовується у дуже великій кількості комп'ютерних ігор для захисту від несанкціонованого копіювання, розповсюдження та злому. Denuvo використовує такі наступні механізми захисту: шифрування коду та обфускація, онлайн-активація та перевірка ліцензії, динамічна модифікація коду, регулярні оновлення, індивідуальна реалізація – Denuvo налаштовує реалізацію із новими особливостями для кожної гри для збільшення часу, необхідного для зламу [30]. Проте, хоч і Denuvo забезпечив ефективний засіб для декількох ігор на початку їх випуску, знаходились групи та окремі хакери, які обходили цей захист, тому ефективність Denuvo змінюється в залежності від таких факторів, як популярність гри, бажання зломщиків та часовий проміжок з моменту випуску гри. Крім того, у гравців були нарікання стосовно впливу захисту на продуктивність гри та постійному підключення до мережі і цілковитому обмеженні гри у режимі офлайн [30].

Захистом Denuvo користуються такі ігри як FIFA series, Need for Speed Heat, Star Wars Jedi: Fallen Order, Assassin's Creed Valhalla, Resident Evil Village, Hitman 3, Marvel's Avengers, Dragon Ball Z: Kakarot, Detroit: Become Human, Final Fantasy XV, Mortal Kombat 11, Middle-earth: Shadow of War, Hogwarts Legacy [31].

Крім відомого захисту від Denuvo, частина ігор також славиться своїм «особливим» підходом до захисту від піратства. Так, деякі ігри при виявленні несанкціонованого копіювання змінює свій функціонал: у грі GTA Vice City починається нескінченний дощ та відсутня можливість зберігати ігровий процес, а у піратській версії гри Crysis Warhead відсутні кулі – замість них зі зброї вилітає світська птиця, що унеможливлює проходження гри, і підштовхує охочих у неї зіграти придбати ліцензійну версію [32].

Використання засобів захисту комп’ютерних ігор від загроз є досить різноманітним, тому варто провести порівняльний аналіз найбільш відомих ігор і засобів захисту, що у них застосовуються.

Для проведення аналізу варто обрати ігри різного року видавництва та здебільшого різних виробників, оскільки варто також проаналізувати методи захисту в різних продуктах одного виробника.

Результати дослідження наведено у таблиці 1.1.

Таблиця 1.1 – Результати аналізу використання методів захисту відомих комп’ютерних ігор

	Компанія-розробник	Рік випуску	Багатофакторна автентифікація	Онлайн-автентифікація	Обмеження офлайн режиму	DLC	Перевірка цілісності	Обмежене встановлення (ліцензійні ключі)	Активація та ліцензування	Використання систем виявлення шахрайства
1	2	3	4	5	6	7	8	9	10	11
Hogwarts Legacy [33]	Warner Bros. Games	2023	-	+	+	-	+	+	+	+
Cyberpunk 2077 [34]	CD PROJECT RED	2020	-	-	-	+	+	+	+	-
Apex Legends [35]	Electronic Arts	2019	+	+	+	-	+	-	-	+
Mobile Legends: Bang Bang [36]	Moonton	2016	-	+	+	+	+	+	-	+
The Witcher 3: Wild Hunt [37]	CD Project Red	2015	-	-	-	+	-	+	+	-
Dota 2 [38]	Valve	2013	+	+	+	-	+	+	-	+
CS:GO [39]	Valve	2012	+	+	+	-	+	+	-	+
The Elder Scrolls V: Skyrim [40]	Bethesda Softworks	2011	-	-	-	+	-	+	+	-
The Sims 3 [41]	Electronic Arts	2009	-	-	-	+	-	+	+	-

Продовження таблиці 1.1

1	2	3	4	5	6	7	8	9	10	11
League of Legends [42]	Riot Games	2009	-	+	-	+	+	+	-	+
StarCraft [43]	Blizzard Entertainment	1998	-	+	-	+	+	-	+	-

Так, можна зробити наступні висновки: розробники кожної із ігор ставляться до питання їх захисту по-різному, і тому у частини досліджуваних ігор не реалізовано і половини необхідного захисту. Проте, можна відмітити, що майже у всіх іграх присутня перевірка цілісності а також обмеження встановлення за допомогою ліцензійних ключів (не більше одного застосунку на один пристрій) – ця практика могла б бути доволі дієвою, якби застосовувалась у комплексі з іншими методами. Крім того, майже ніде немає багатофакторної автентифікації.

Потрібно відмітити, що абсолютно всі перераховані ігри було зламано через деякий час після випуску. Наприклад, Hogwarts Legacy була зламана через декілька днів після виходу [44]. Проте, кожна гра піддалась різним загрозам, відповідно у Hogwarts Legacy, Cyberpunk 2077, The Witcher 3: Wild Hunt, The Elder Scrolls V: Skyrim та StarCraft хакери подолали захист від копіювання, і тому в мережі є досить велика кількість джерел із можливістю безкоштовного завантаження копій цих ігор. Apex Legends, Mobile Legends: Bang Bang, Dota 2 та CS:GO піддаються атакам хакерів, які зламують карти, текстури, використовують чіт-коди задля полегшення ігрового процесу – цю проблему розробники не можуть подолати вже досить великий проміжок часу. Тому можна помітити закономірність: ігри, у яких в якості захисту є обмеження офлайн режиму найменше потерпають від піратства, проте найбільше від шахрайства та хакінгу, отже можна зробити припущення, що тісний зв’язок гри із сервером та унеможливлювання ігрового процесу без мережі є одним із найдієвіших засобів захисту від піратства.

Також можна помітити, що Cyberpunk та The Witcher 3: Wild Hunt, розроблені однією компанією – CD Project Red, мають однакові методи захисту, так, як і Dota

2 та Counter-Strike: Global Offensive від Valve, на відміну від Apex Legends та The Sims 3, розроблених Electronic Arts, – у них захисти разюче відрізняються, що, ймовірно, пов’язано із різницею у часі, проте, це все одно не завадило хакерам зламати ці ігри.

1.4 Постановка задачі

Як показав аналіз атак, найбільш поширені атаки проявляються у формі шахрайства і хакерства, піратства та зловмисного програмного забезпечення в модах і програмному забезпеченні сторонніх розробників, крадіжці даних, внутрішньоігрової валюти. Тобто основний вектор спрямований на ігрові ресурси та власне гравців і їх даних. Отже, основна мета розробки повинна бути націленою на захист від піратства, захист ресурсів та персональних даних гравців.

Також при аналізі методів захисту було проведено порівняльний аналіз кількох найвідоміших ігор. Результати цього аналізу показали, що самим стійким захистом від піратства є з’єднання гри із сервером і обмеження функціональності у режимі офлайн. Цей підхід служить прикладом найкращих практик для захисту ігрових активів і запобігання несанкціонованому копіюванню та розповсюдженню. Посилюючи залежність гри від постійної взаємодії із сервером та обмежуючи доступ офлайн, розробники можуть досить ефективно зменшити ризик піратства та створити середовище, сприятливе для захисту інтелектуальної власності.

Оптимальним варіантом для створення захищеної гри є жанр візуальна новела, оскільки він користується достатньою популярністю для попиту та надає змогу реалізувати новий підхід до захисту комп’ютерних ігор, який буде більш стійким, до атак, спрямованих як на ігрові ресурси, так і дані гравців та від піратства, ніж попередні.

При розробці захищеного застосунку варто використати розглянуті у попередньому підрозділі засоби захисту.

Завдяки використанню безпечних платіжних шлюзів дозволить підвищити безпеку користувацьких грошових переказів, використання захищених серверів та

зберігання даних саме на них дозволить зменшити до мінімуму ймовірність викрадення даних гравців. Вимоги до надійних паролів у комплексі із багатофакторною автентифікацією також зменшать ймовірність викрадення даних гравців, платіжних даних і, зокрема, їх облікових записів. Всі перераховані вище засоби є вбудованим захистом, тому будуть застосовуватись на етапі розробки.

Захист від піратства варто реалізувати за допомогою обмеження встановлення ігри шляхом прив'язки ПЗ до облікового запису гравця, а той до пристрою, щоб на один пристрій припадав лише один обліковий запис [1]. Саме тому при проектуванні гри варто звернути увагу на забезпечення можливості гравцем відновити свій обліковий запис – як ще один метод захисту (як від викрадення, так і від помилкових дій користувача, або ж зміні пристрою). Також потрібно застосувати обов'язкову онлайн-автентифікацію, обфускацію коду та перевірку цілісності застосунку для зменшення ризику його модифікації. В якості захисту від піратства варто прив'язати гру до серверу та зробити її безкоштовною, натомість всі фінансові операції перенести в її середину – забезпечити гравцям можливість здійснювати внутрішньоігрові придбання.

1.5 Висновки з розділу

Аналіз атак на комп'ютерні ігри дозволив довести актуальність необхідності захисту поширення ігрового контенту. На основі цього аналізу було визначено основні вектори атак, які притаманні комп'ютерним іграм загалом і вектори атак, які з'явилися нещодавно і пов'язані із розвитком інформаційних технологій та їх розробки. Розглянуто було такі атаки: захоплення облікових записів, DDoS-атаки, шахрайство та хакерство, несанкціоноване копіювання (піратство), програми-вимагачі, фішингові атаки, зловмисне програмне забезпечення в модах і програмному забезпеченні сторонніх розробників, крадіжка віртуальної та реальної валюти, крадіжка даних. Також було визначено атаки із найбільшими збитками – це шахрайство і хакерство, піратство і крадіжка даних, та основний їх вектор – ігрові ресурси і дані гравців.

Для захисту від цих атак було проаналізовано можливості сучасних засобів захисту як прикладних програм, так і комп’ютерних ігор, серед яких було виділено використання серверів, використання безпечних платіжних шлюзів, використання систем виявлення шахрайства, вимоги до надійних паролів, багатофакторну автентифікацію, процеси відновлення облікових записів, активацію та ліцензування, онлайн-автентифікацію, цифрові підписи, обфускацію коду, перевірку цілісності, заходи проти налагодження, контрзаходи проти піратства, обмеження офлайн-режиму, обмежене встановлення (за допомогою ліцензійних ключів), захист завантажуваного вмісту (Downloadable Content, DLC). Також проведено порівняльний аналіз сучасних комп’ютерних ігор.

На основі аналізу засобів захисту було визначено їх недостатність та неадекватність атакам. Тому було зроблено висновок у потребі формування нового підходу до захисту комп’ютерних ігор жанру візуальна новела, на основі якого було поставлено задачу комплексної бакалаврської дипломної роботи: розробити новий підхід до захисту комп’ютерних ігор від піратства, викрадення даних та модифікація ігрових ресурсів на прикладі засобу для поширення візуальної новели.

Отже, орієнтуючись на проведеному аналізі загроз комп’ютерним іграм, методам захисту прикладного програмного забезпечення та ігор можна переходити до проектування архітектури засобу.

2 АРХІТЕКТУРА ЗАХИЩЕНОГО ЗАСТОСУНКУ ПОШИРЕННЯ ВІЗУАЛЬНОЇ НОВЕЛИ

2.1 Узагальнена архітектура засобу

Оскільки одним із методів захисту є зв’язок застосунку із сервером, то і архітектура його клієнт-серверна. Клієнт-серверну архітектуру можна означити як взаємодію набору клієнтів із сервером, який забезпечує перших послугами, за допомогою мережі [45]. Тож застосунок має два модуля: клієнта та сервера – кожен із них має свою базу даних (БД). Враховуючи особливість захисту, максимальна частина конфіденційної інформації та ігрових ресурсів знаходиться в БД сервера, саме тому для проходження візуальних новел потрібен постійний зв’язок із сервером.

Побудова архітектури за рахунок декомпозиції, тобто поділу компонентних блоків, кожен із яких має свою окрему функцію, яка мінімально впливає на інші блоки, забезпечує гнучкість, масштабованість, відмовостійкість, оптимізацію продуктивності та кращий рівень безпеки.

Розділення як серверної, так і клієнтської частини на окремі рівні, або блоки, дозволяє забезпечити модульне проектування та розробку, тобто кожен компонент можна проектувати, створювати та обслуговувати незалежно один від одного. Це полегшує керування, оновлення та відновлення системи в разі інциденту.

Крім того, вона дозволяє забезпечити ізоляцію помилок – якщо в якомусь блоці виникне помилка, вона не вплине на інший блок. Таким чином несправності ізолюються та для розробників з’являється можливість знайти помилку в рази швидше. Такий фактор дозволяє підвищити надійність системи.

Розділення компонентів також вагомо впливає на безпеку застосунку: чутливі компоненти ізолюються одна від іншої, крім того, зменшується потенційний вплив порушення безпеки на інші компоненти. Саме тому в цілях безпеки (а саме забезпечення цілісності і конфіденційності) взаємодію баз даних як із сервером на серверній частині та іншими блоками, крім репозиторіїв, на

клієнтській частині відокремлено від інших рівнів застосунку. Це зроблено для уникнення прямих запитів та власне несанкціонованого доступу до бази даних, навіть якщо веб-сервер скомпрометовано.

Схематичне представлення структури засобу зображено на рисунку 2.1.

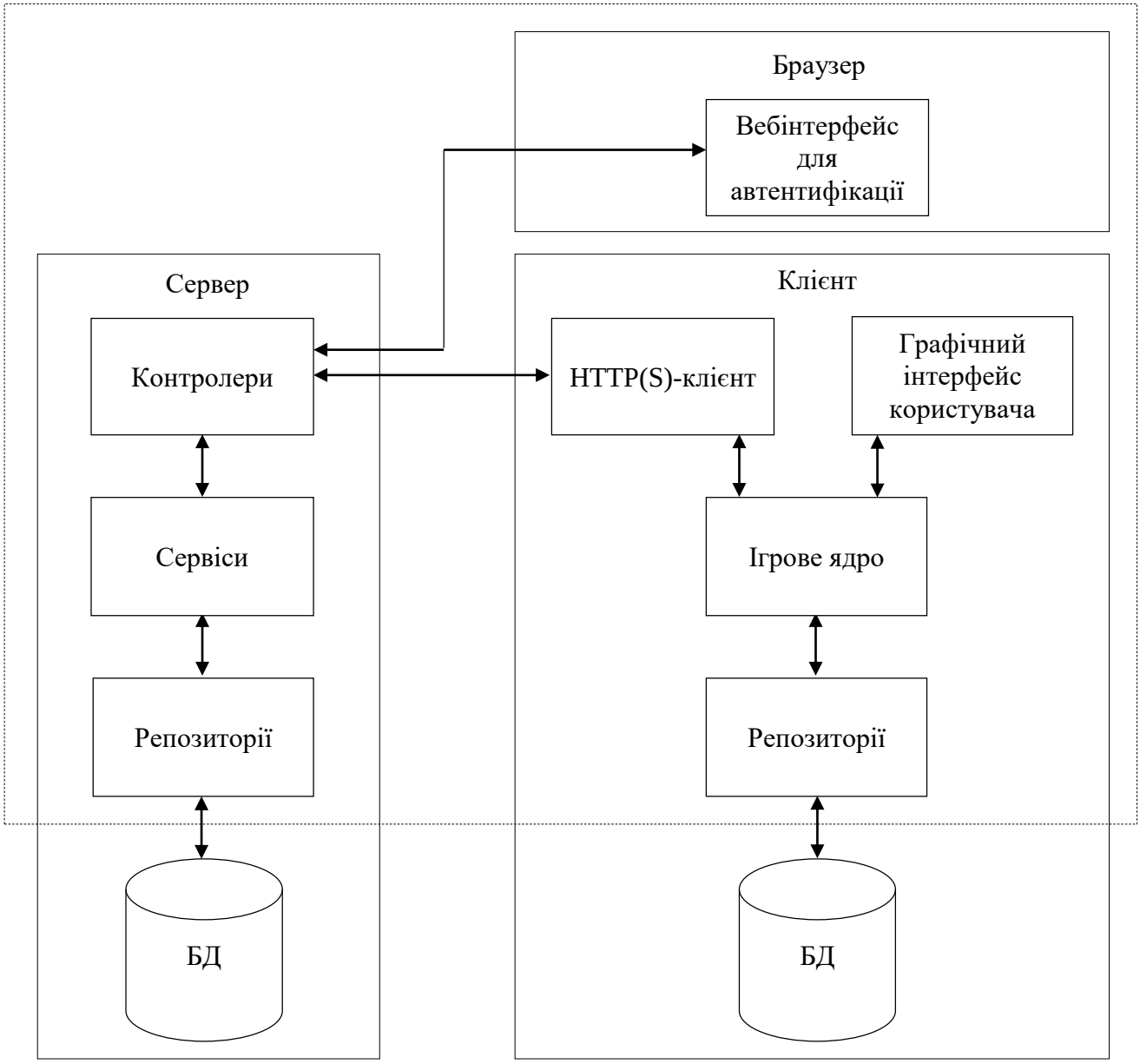


Рисунок 2.1 – Узагальнена архітектура засобу

Отже, застосунок для розповсюдження візуальних новел складається з трьох модулів – серверу, клієнта та вебінтерфейсу. Ті, у свою чергу, складаються із таких елементів: сервер – з бази даних, репозиторіїв, сервісів та контролерів, а клієнт – з

бази даних, репозиторіїв, ігрового ядра, графічного інтерфейса користувача (GUI) та HTTPS-клієнта, вебінтерфейс – веб-сторінок.

В даній роботі розглядається частина, виділена штриховою лінією, тобто контролери, сервіси та репозиторії зі сторони сервера, графічний інтерфейс користувача, HTTP-клієнт, ядро та репозиторії зі сторони клієнтської частини і вебінтерфейс для автентифікації в браузері.

2.2 Структура модуля клієнта

На основі аналізу розробленої архітектури визначено перелік задач, які має розв’язувати клієнтський застосунок:

- зв’язок із сервером (надсилання HTTP-запитів та отримання відповідей);
- обробка даних, отриманих із відповідей сервера;
- взаємодія із користувачем;
- обробка команд від користувача;
- реалізація ігрової (сюжетної) логіки;
- зберігання даних;
- взаємодія із базою даних;
- інтеграція даних у повноцінні об’єкти;
- обробка об’єктів, отриманих з бази даних;
- захист ігрових ресурсів.

Саме тому модуль клієнта складається з таких блоків:

- HTTP-клієнт;
- графічний інтерфейс користувача;
- ядро;
- репозиторії;
- вбудована база даних.

Отже, модуль розділено на окремі частини для забезпечення більшої надійності роботи застосунку, кожен блок виконує декілька задач, та має зв’язок з іншим блоком, який виконує пов’язані задачі.

Так, наприклад блок, що реалізує API для комунікації із сервером, або графічний інтерфейс, який реалізує API для взаємодії із користувачем та ядро не мають прямого доступу до бази даних, тому ймовірність несанкціонованого доступу значно зменшується. Також HTP-клієнт не має доступу до ядра, а отже не може впливати на перебіг ігрових процесів та, наприклад, на графічний інтерфейс користувача.

Отже, якщо розглядати блоки клієнтської частини застосунку окремо, то випливають такі висновки:

- в БД зберігається інформація про ігровий прогрес, ігрові налаштування та деякі дані облікового запису. Для зменшення громіздкості клієнтського застосунку використовується концепція вбудованої БД;
- репозиторії слугують для зв’язку застосунку із БД;
- графічний інтерфейс користувача надає змогу користувачеві взаємодіяти із застосунком за допомогою візуальних складових, таких як графічні зображення. GUI відображає власне візуальні новели да дозволяє користувачеві взаємодіяти із застосунком. Таким чином він включає в себе набір зображень та тексту.
- ядро відповідає за реалізацію ігрової логіки (за допомогою контролерів) та обробляє дані, отримані від графічного інтефейсу та HTTP-клієнта;
- HTTP-клієнт надсилає запити на сервер та отримує відповідь із необхідними даними та передаває їх до ігрового ядра.

Отже, кожен рівень структури модуля клієнта відповідає за свою функцію, яка лише частково пов’язана з однією, чи двома іншими, всі вони є певною мірою ізольованими один від одного, що дозволяє з легкістю проектувати та підтримувати їх, ізолювати помилки та забезпечувати безпеку.

2.3 Структура модуля сервера

Ґрунтуючись на результатах аналізу розробленої архітектури визначено перелік задач, які має розв’язувати серверна частина:

– зв’язок із клієнтом (отримання запитів та надсилання відповідей);
– обробка клієнтських запитів;
– зберігання даних;
– взаємодія із базою даних;
– інтеграція отриманої із бази даних інформації у повноцінні об’єкти;
– обробка об’єктів, отриманих із бази даних;
– захист даних, що надсилаються;
– ідентифікація;
– авторизація.

Сервер складається із таких рівнів:

– контролери;
– сервіси;
– репозиторії;
– база даних.

Отже, кожен блок модуля виконує декілька пов’язаних між собою задач, та має зв’язок із суміжним блоком – який також виконує декілька пов’язаних задач та передає дані у необхідному порядку.

Також блоки поділені між собою таким чином, щоб забезпечити можливість модульного проектування та розробки, підтримки та налагодження серверу, а також захист даних шляхом ізоляції БД від інших блоків для запобігання несанкціонованому доступу, легше виявляти помилки та ізолювати їх, та ізолювати вплив порушення безпеки.

База даних сервера – важливий компонент системи, який підтримується на серверній стороні. В ній мають зберігатись такі дані як: інформація про підписки, дані користувачів (логіни, паролі, прогреси у кожній історії), а також власне ігрові

ресурси – ці дані навмисно відмежовані для більшої безпеки та запобіганню несанкціонованого доступу, або у випадку із ігровими ресурсами – копіювання.

Рівень репозиторіїв забезпечує безперебійну взаємодію із базою даних, тобто надсилає запити до БД та формує із отриманих даних повноцінні об’єкти. Цей рівень відокремлює рівень реалізації логіки – сервісний – від прямої взаємодії із базою даних.

Контролери міститять прикладні програмні інтерфейси (API) для RESTful сервіса (для взаємодії із клієнтським десктопним застосунком). Крім того, контролери відповідатимуть архітектурному шаблону модель-представлення-контролер (MVC), отримуючи запити від браузера та надсилаючи відповіді, та братимуть участь при проведенні ідентифікації та автентифікації [46].

Сервісний рівень з’єднує контролери та репозиторії та реалізує логіку обробки інформації, отриманої з обох блоків, а також реалізухє логіку ідентифікації та авторизації.

Таким чином, база даних сервера працює як єдине сховище для різних типів даних, забезпечуючи безпечне зберігання підписок, даних користувачів і ігрових ресурсів. Рівень репозиторіїв забезпечує безперебійний зв’язок із базою даних, чітко відокремлюючи рівень логічної реалізації від прямого доступу до бази даних. Контролери розміщують API, що полегшують взаємодію з клієнтською програмою, і дотримуються шаблону MVC для задоволення запитів на основі браузера. Сервісний рівень бере на себе критично важливу роль в обробці інформації, отриманої від контролерів і репозиторіїв, фактично слугуючи мостом, що об’єднує їх функціональні можливості.

2.4 Метод захисту критичних ресурсів та конфіденційних даних

Хоч і архітектура засобу побудована на принципі модульності, але окремий блок, що відповідає за захист системи відсутній. Натомість кращим варіантом є розподілення механізми захисту (ідентифікація, автентифікація та ін.) між блоками.

Варто розглянути окремо захист клієнтської частини і серверної та розділити всі заходи на покрокові етапи.

Крок 1. Основою концепції захисту є клієнт-серверна архітектура та зберігання всіх необхідних даних для роботи клієнтського застосунку на стороні сервера, в базі даних. клієнт-серверна архітектура дозволяє тісно прив’язати ігровий процес до наявності мережі інтернет, внаслідок чого зникає потреба у копіюванні десктопного застосунку, та за допомогою ідентифікації та автентифікації обмежити використання застосунку в режимі офлайн, оскільки при його запуску обов’язково необхідно пройти автентифікацію.

Крок 2. На модулі серверу частина механізму автентифікації присутня в блоці контролерів та на сервісному рівні – вони здійснюють автентифікацію та авторизацію користувачів. За допомогою вищеперерахованих рівнів та репозиторіям зі сторони сервера здійснюється реєстрація користувачів та призначення їм ролей. Дані необхідні для ідентифікації та автентифікації зберігаються у базі даних на стороні сервера. Також на сервісному рівні сервера присутній блок шифрування. Автентифікація відбувається за допомогою веб-інтерфейсу.

Крок 3. Для захисту даних, що передаються від клієнта до сервера та навпаки потрібно використати захищений протокол. Оскільки обмін запитами відбувається за допомогою HTTP-запитів, то варто використати захищений HTTPS протокол.

Крок 4. Для захисту власне коду як серверної частини, так і клієнтської, варто використати навісний захист, а саме обфускацію, а для більшої стійкості – одразу декілька її видів. Цей метод дозволить у разі компрометації коду сервера ускладнити аналіз зловмисником програмного коду, та у випадку можливої декомпіляції – коду десктопного клієнта.

2.5 Модуль веб-інтерфейсу для автентифікації

Власне автентифікація, як і реєстрація користувачів, відбувається за допомогою веб-інтерфейсу.

Веб-інтерфейс – це веб-сторінка або їх сукупність, вдалий дизайн якої полегшує взаємодію користувача із певним сайтом чи сервером [47]. Саме тому для полегшення процедури реєстрації на початку роботи із застосунком та подальшої авторизації користувача варто створити зручний та зрозумілий користувачеві веб-інтерфейс, який надасть змогу взаємодіяти із відповідними REST-контролерами серверу.

Веб-інтерфейс, як і десктопний застосунок, надсилає HTTP-запити на сервер, використовуючи протокол HTTPS. Інтерфейс являє собою декілька веб-сторінок, а саме сторінку із формою для логіну, окрему для реєстрації, для купівлі пакету послуг та ще декілька веб-сторінок для відображення персонального кабінету та можливості редагування деяких даних облікового запису користувачем (таких як аватар, та нікнейм).

Так, не відвідавши відповідну адресу у браузері та не зареєструвавшись, чи не здійснивши процедуру автентифікації, користувач не зможе використати десктопний застосунок, оскільки всі запити, що з нього надсилаються, будуть відхилені.

2.6 Висновки з розділу

Отже, в даному розділі було описано архітектуру засобу для поширення візуальних новел. Архітектура розроблялась на основі проведеного аналізу найпоширеніших загроз комп’ютерним іграм, засобів захисту прикладних програм та ігор, проведених в попередньому розділі. Також при побудові архітектури враховувався досвід вже існуючих комп’ютерних ігор.

Саме тому архітектура застосунку є клієнт-серверна. Такий підхід до її проектування дозволяє уникнути таких загроз, як несанкціоноване копіювання

застосунку та контенту гри, його редагування, оскільки весь контент знаходиться в базі даних на стороні сервера.

Кожен модуль застосунку також розділений на окремі рівні, кожен із яких має максимум два зв’язка з іншими – це запобігає поширенню можливих помилок при роботі застосунку та сприяє швидшому їх пошуку і захищає від несанкціонованого доступу до бази даних з боку сервера. Саме тому модуль сервера складається із бази даних, репозиторіїв, сервісного рівня та контролерів, а модуль сервера складається із бази даних, репозиторіїв, сервісного рівня та контролерів.

Також попередньо проведені аналізи загроз та засобів захисту прикладного ПЗ та комп’ютерних ігор вплинув на розробку методу захисту застосунку та його критичних ресурсів зокрема.

Декомпозиція архітектури слугує зменшенням складності проектування, розробки та подальшого супроводу застосунку. Цей фактор сприяє створенню більш чітких, простих та зрозумілих алгоритмів роботи застосунку, які відповідають основним концепціям розробки програмного забезпечення та сприяють пришвидшенню цього процесу.

3 АЛГОРИТМ РОБОТИ ЗАХИЩЕНОГО ЗАСТОСУНКУ ПОШИРЕННЯ ВІЗУАЛЬНОЇ НОВЕЛИ

3.1 Узагальнений алгоритм

При розробці алгоритму роботи застосунку варто враховувати особливості його структури, а саме клієнт-серверну архітектуру. Результати розробки загального алгоритму роботи застосунку зображений на рисунку 3.1.

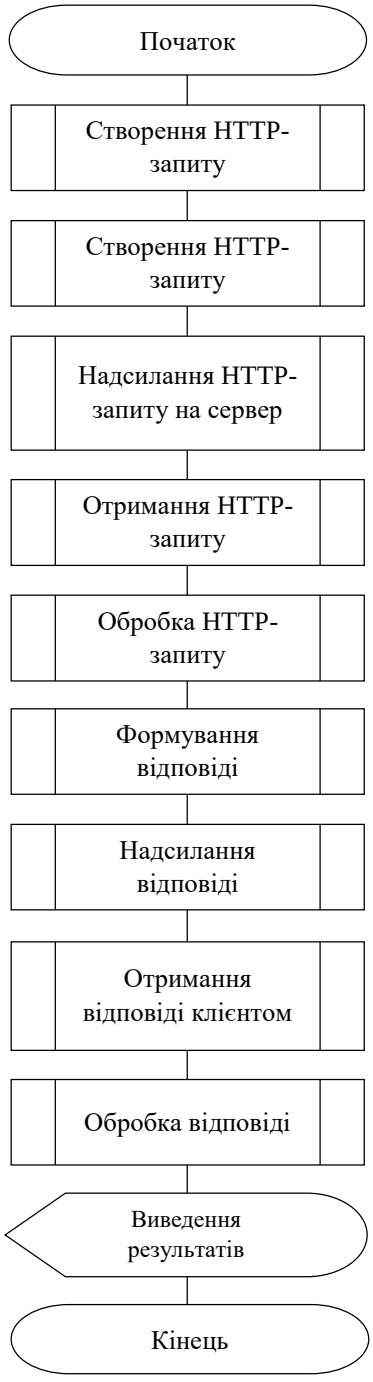


Рисунок 3.1 – Узагальнений алгоритм роботи застосунку

Отже, спочатку клієнт створює HTTP-запит та надсилає його за допомогою протоколу HTTPS на сервер. Той отримує запит і обробляє його, проводить валідацію. В залежності від валідності запиту та вхідних даних сервер формує відповідь та надсилає її клієнту. Клієнт також обробляє отриману відповідь та виводить результати на інтерфейс користувачеві.

Запит на стороні клієнта створюється за допомогою рівня контролерів та за допомогою API-функцій – вони ж приймають участь у отриманні відповіді та її обробці.

Зі сторони сервера запити отримуються за допомогою REST-контролерів а обробляється за допомогою сервісного рівня. Якщо ж запит надсилається із браузера, то обробляється він аналогічним чином, проте отримується через контролери MVC.

3.2 Алгоритм роботи модуля клієнта

Модуль клієнта відповідає за взаємодію із користувачем, отримання від нього команд, обробку їх та валідацію, формування HTTP-запитів, отримання і обробку відповідей, реалізацію сюжетної логіки та власне показ новели.

Узагальнений алгоритм роботи клієнтського десктопного застосунку зображено на рисунку 3.2.

Для початку роботи із застосунком, користувач повинен авторизуватись або зареєструватись за допомогою веб-інтерфейсу. В результаті заповнення відповідної форми на введену ним адресу електронної пошти приходить повідомлення, яке містить токен авторизації, який користувач повинен ввести у відповідне поле в клієнтському застосунку. Вікно із полем для токену відкривається першим при запуску застосунку та не дозволяє відкривати ніякі інші вікна. Далі відбувається валідація та перевірка токену і власне надання користувачеві прав. Відкривається наступне вікно.

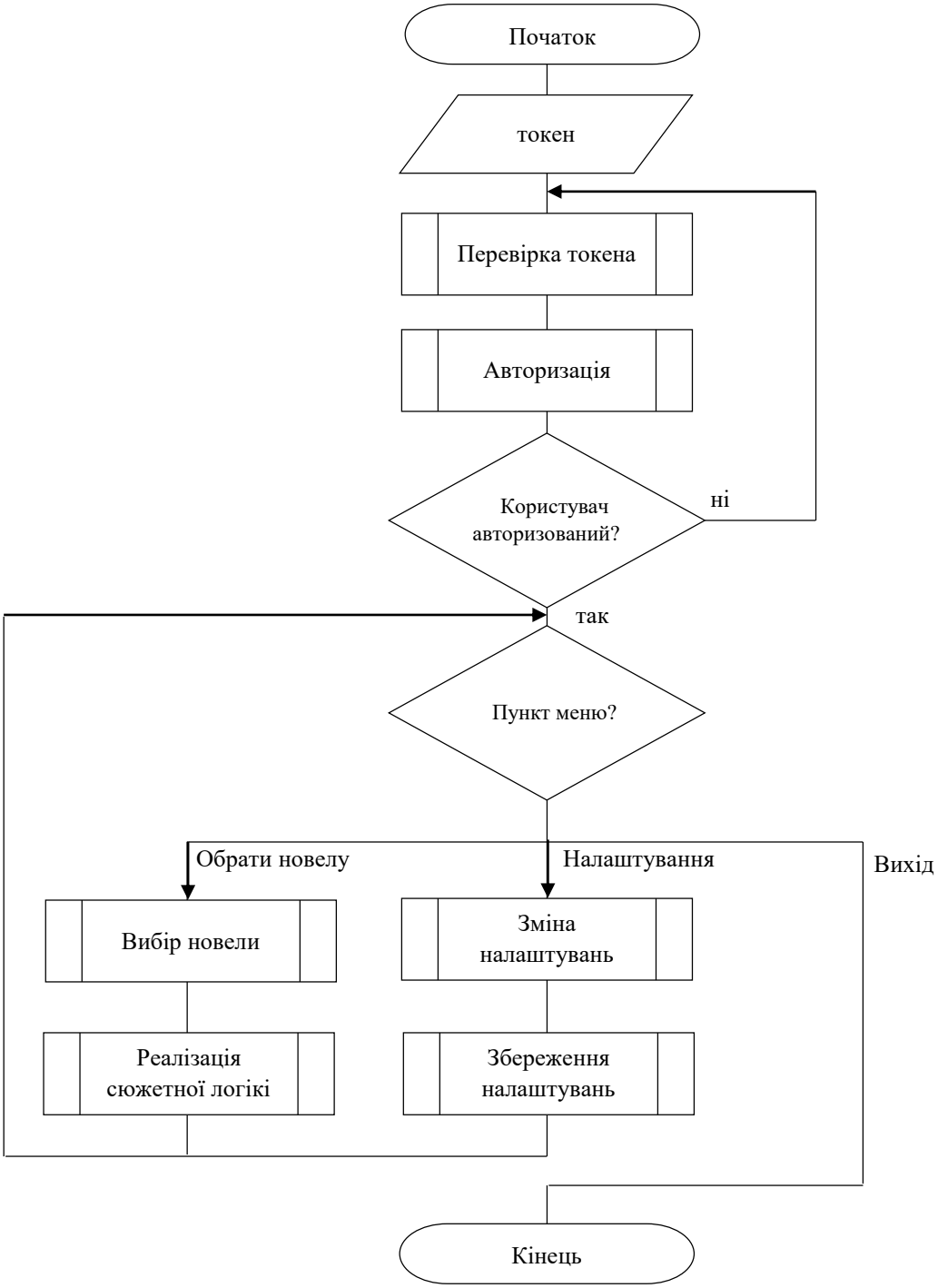


Рисунок 3.2 – Алгоритм роботи модуля клієнта

Наступним вікном користувач бачить меню із пунктами для вибору новели, зміни налаштувань та кнопкою для виходу.

Якщо користувач обирає зміну налаштувань, то переходить до відповідного вікна, де може змінити гучність звукових ефектів та швидкість виведення тексту. При виборі користувачем пункту «Вихід», застосунок завершує свою роботу.

При виборі останнього пункту – «Обрати новелу» – користувачу відкриваються перелік доступних для проходження новел з їх коротким описом. Після вибору новели виводиться екран із повідомленням про початок показу історії та з діями користувача починається реалізація сюжетної логіки, узагальнений алгоритм якої зображений на рисунку 3.3.

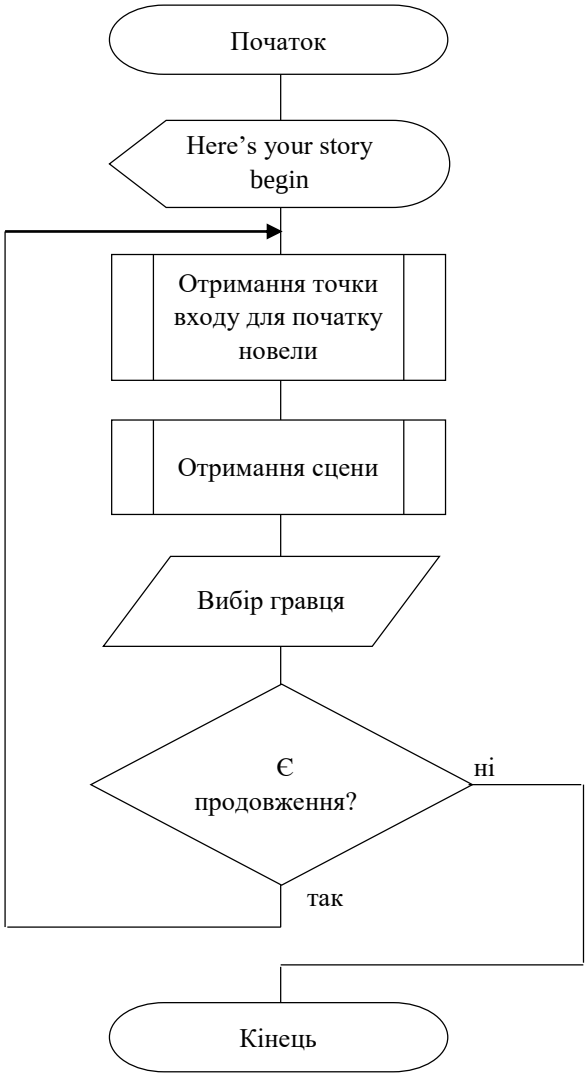


Рисунок 3.3 – Узагальнений алгоритм реалізації сюжетної логіки

Отже, спочатку виводиться екран із повідомленням про початок історії, після чого користувач натискає на екран і починається показ історії. З кожним натисканням гравця сцена змінюється відповідно до сюжету. Проте, відповідно до жанру гри, можливість розвитку сюжету нелінійна, тому він може мати певні розвилки в залежності від вибору користувача. Саме тому іноді можуть з’являтись

сцени із можливістю вибору двох або трьох варіантів – в таких випадках користувачу треба обрати один із запропонованих. Далі в залежності від сюжету виводиться нова сцена – відповідно до точки зупину – ідентифікатора сцени, на якого вказав вибір користувача у попередній.

Алгоритм реалізації сцени без вибору, або ж звичайної сцени зображено на рисунку 3.4.

Рисунок 3.4 – Алгоритм обробки сцени без вибору

Спочатку звичайна сцена перевіряється на відповідність своєму типу сцени та у разі збігу виводиться на екран, після чого надсилається HTTP-запит на сервер на отримання нової точки входу.

Алгоритми реалізації сцени з двома та трьома варіантами вибору зображено на рисунках 3.5 та 3.6 відповідно.

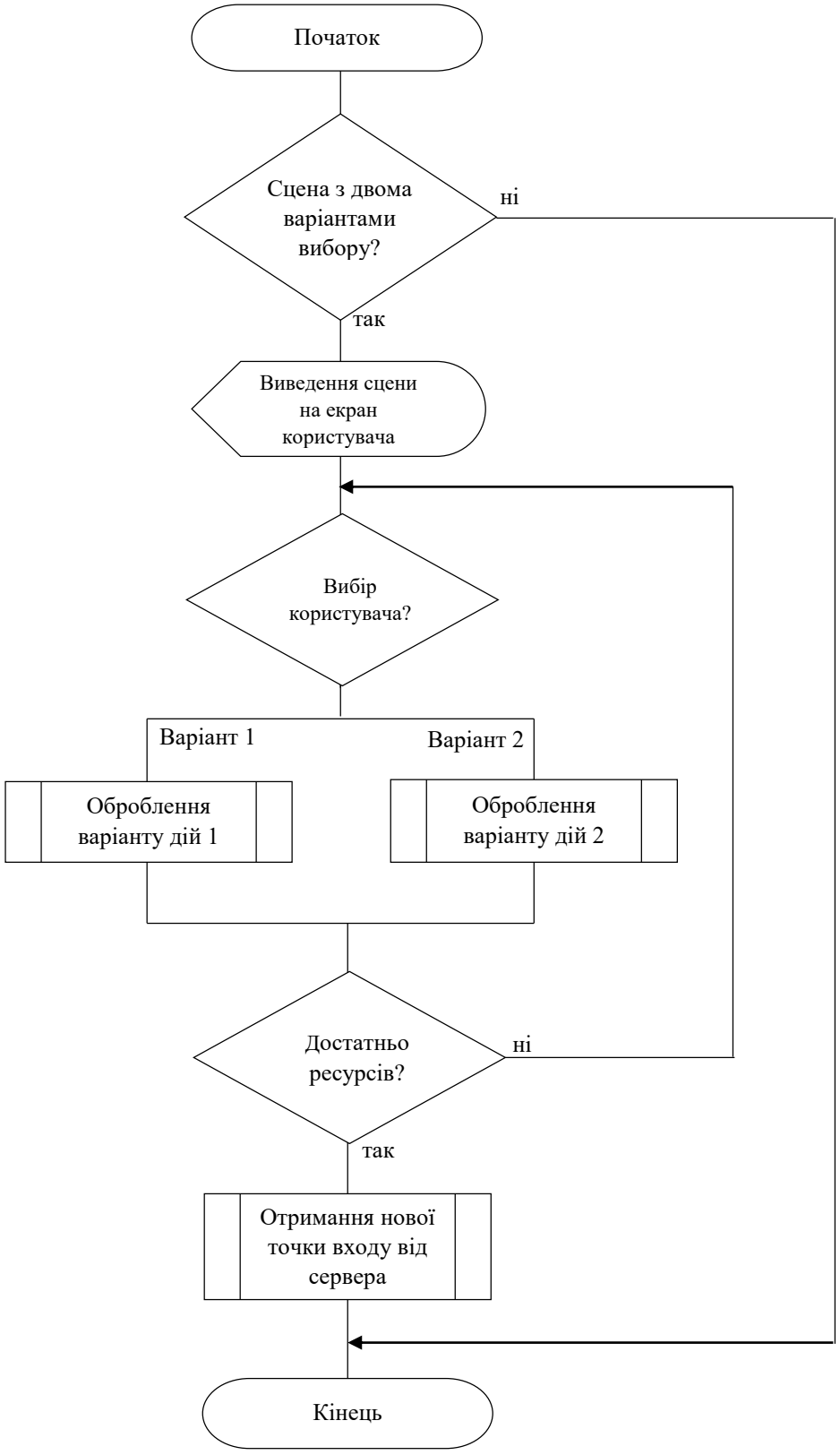


Рисунок 3.5 – Алгоритм обробки сцени з двома варіантами вибору

Реалізація сцени з двома варіантами вибору відбувається наступним чином: сцена перевіряється на відповідність своєму типу та виводиться на екран користувача. Той робить свій вибір, після чого перевіряється, чи достатня кількість

ресурсів (балів) для здійснення цього вибору: якщо ні, то користувач знову повертається до варіантів та може обрати другий. Якщо ресурсів достатньо, то на сервер надсилається запит на отримання нової точки входу та власне сцени.

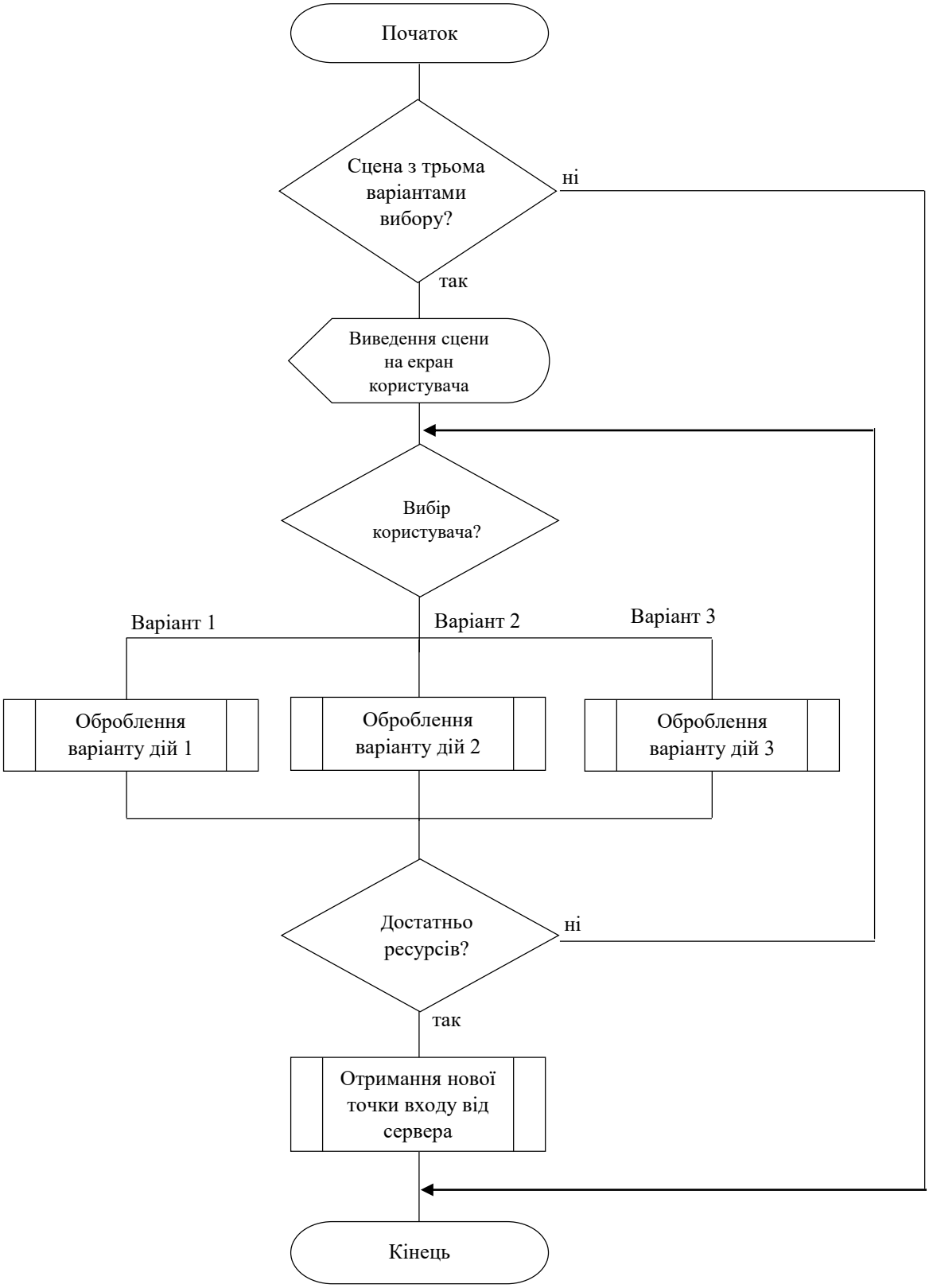


Рисунок 3.6 – Алгоритм обробки сцени з трьома варіантами вибору

Алгоритм реалізації сцени з трьома варіантами вибору працює аналогічним чином, як із двома, проте варіантів у користувача в цьому випадку три.

3.3 Алгоритм роботи модуля сервера

При створенні алгоритму роботи модуля сервера також варто опиратись на створену у попередньому розділі архітектуру застосунку.

Узагальнений алгоритм роботи модуля сервера представлено на рисунку 3.7.

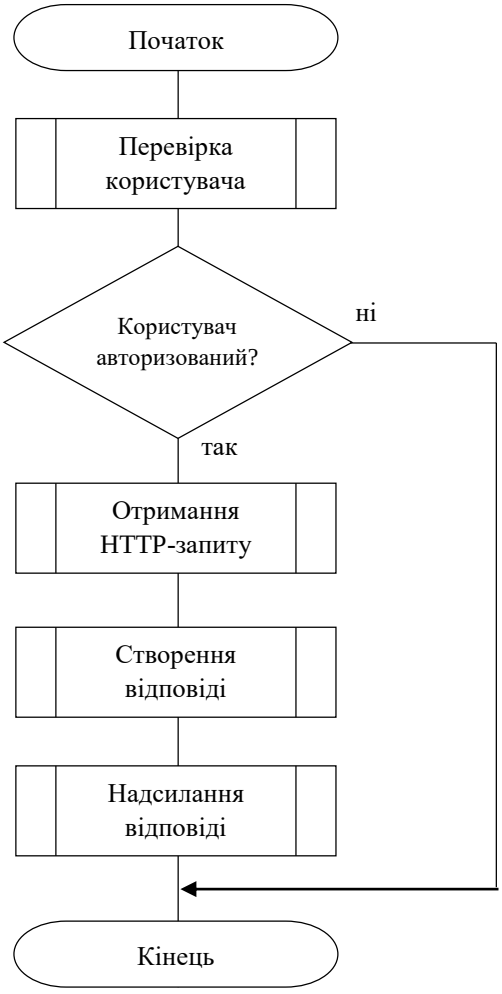


Рисунок 3.7 – Алгоритм роботи модуля сервера

Для початку, щоб отримати доступ до надсилання запити на сервер, користувач повинен авторизуватись у системі, або зареєструватись, якщо у нього ще не створено обліковий запис.

Отже, після авторизації, користувач надсилає запит – сервер його отримує та перевіряє користувача на те, чи він взагалі існує, та чи авторизований. Далі модуль обробляє запит та відповідно до прав користувача формує відповідь – якщо у користувача є права на запрошену дію, то створюється позитивна відповідь із необхідними даними, якщо ж ні, то відповідь міститиме в собі код помилки, після чого вона надсилається клієнту.

Модуль сервера складається з декількох блоків – контролерів, сервісів, репозиторіїв та бази даних. Кожен із цих блоків має свій алгоритм роботи.

Так контролери складаються з двох блоків – REST-контролерів, та тих, що відповідають концепції MVC. Алгоритм роботи REST-контролерів зображено на рисунку 3.8.

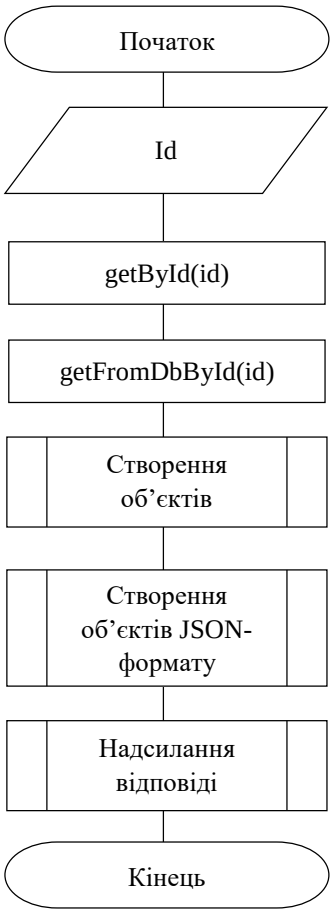


Рисунок 3.8 – Алгоритм роботи REST-контролера

REST-контролери забезпечують API для більш зручної комунікації із клієнтом, та в даному застосунку використовуються більше для взаємодії із десктопним клієнтом. Отже, для початку, контролер отримує необхідний ідентифікатор та перевіряє його валідність, після чого звертається до сервісного рівня для отримання об’єкта за заданим ідентифікатором, той у свою чергу звертається до репозиторіїв, та в результаті отримує запитаний об’єкт. Контролер формує JSON із отриманого об’єкта, далі формується відповідь, яка включає в себе новостворений JSON та надсилає його на необхідну адресу клієнта.

Контролери з концепції MVC слугують для комунікації із веб-браузером та представлення веб-сторінок клієнту. Працюють вони ідентично до REST-контролерів, лише із однією відмінністю – на вхід запитом подаються не ідентифікатори а запити на повернення необхідної сторінки (можливо із деякими параметрами).

Сервісний рівень реалізує логіку обробки всіх об’єктів, тому єдиного алгоритму роботи сервісів, як такого, немає. Кожен сервіс виконує свій перелік задач, які можуть бути досить різноманітними. Сервіси, що відповідають за роботу із об’єктами, отриманих від бази даних чи від контролерів, реалізують обробку цих об’єктів. Так, можуть виконуватись якісь підрахунки, пов’язані із об’єктами, їх зміна, модифікація і т. д. Також є сервіси, що здійснюють логіку ідентифікації та автентифікації – вони використовують сервіси, що взаємодіють із базою даних.

Рівень репозиторіїв відповідає за взаємодію із базою даних, формування об’єктів із отриманих із неї даних та передачу їх сервісному рівню.

Алгоритм роботи репозиторіїв зображено на рисунку 3.9.

Так, репозиторії отримують необхідний запит від рівня сервісів, наприклад – отримати об’єкт за певним ідентифікатором. Використовуючи запит, створений відповідною діалоговою мовою програмування для здійснення запитів, надсилає його до БД. Та, в свою чергу, надсилає репозиторіям запитані дані. Ті формують із них об’єкт та передають сервісам. Аналогічним чином можуть формуватись деякі структури даних.

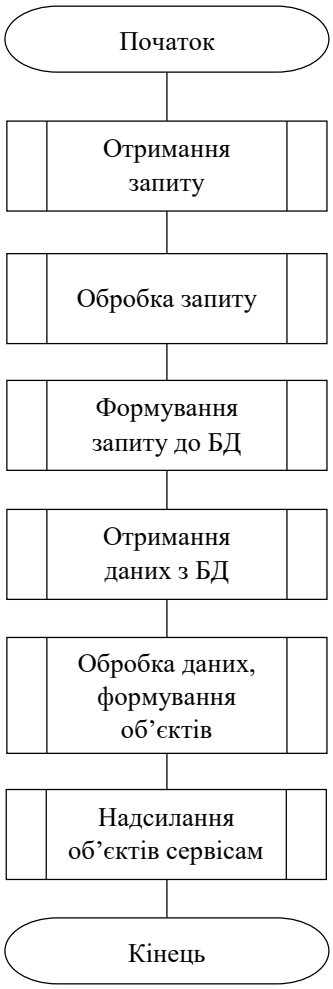


Рисунок 3.9 – Алгоритм роботи репозиторію

Таким чином кожен блок модуля має по декілька алгоритмів роботи, в залежності від поставленої задачі.

3.4 Алгоритм роботи блоку автентифікації

Блок автентифікації складається із декількох компонентів, які знаходяться в різних модулях застосунку та на різних їх рівнях. Загальний алгоритм роботи блоку автентифікації зображено на рисунку 3.10.

Для початку роботи користувач повинен зареєструватись або автентифікуватись, після чого на вказану ним адресу електронної пошти буде надіслано токен для автентифікації у десктопному застосунку, де користувач повинен ввести вміст отриманого повідомлення. Наступним кроком введений токен проходить валідацію та перевірку відповідності надісланому. Якщо результати перевірки позитивні, то користувач може переходити до ігрового

процесу, в разі протилежного результату застосунок знову очікуватиме введення токена.

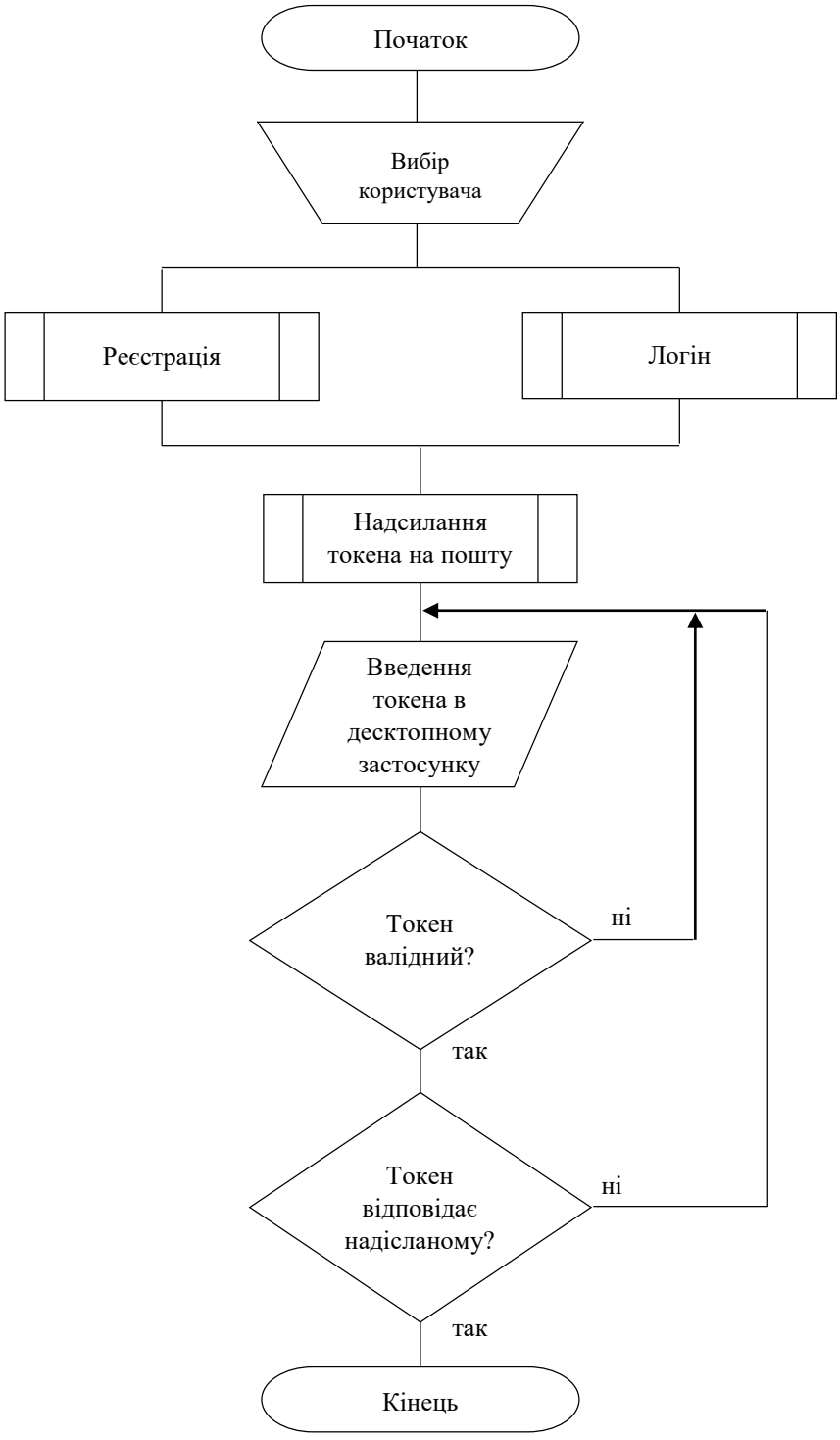


Рисунок 3.10 – Алгоритм роботи модуля автентифікації

Алгоритм має блок, що реалізує реєстрацію, та інший, що реалізує логін. Кожен із них – це окремий процес, який також має свій алгоритм. Отже, алгоритм реєстрації зображено на рисунку 3.11, а логіну – на рисунку 3.12.

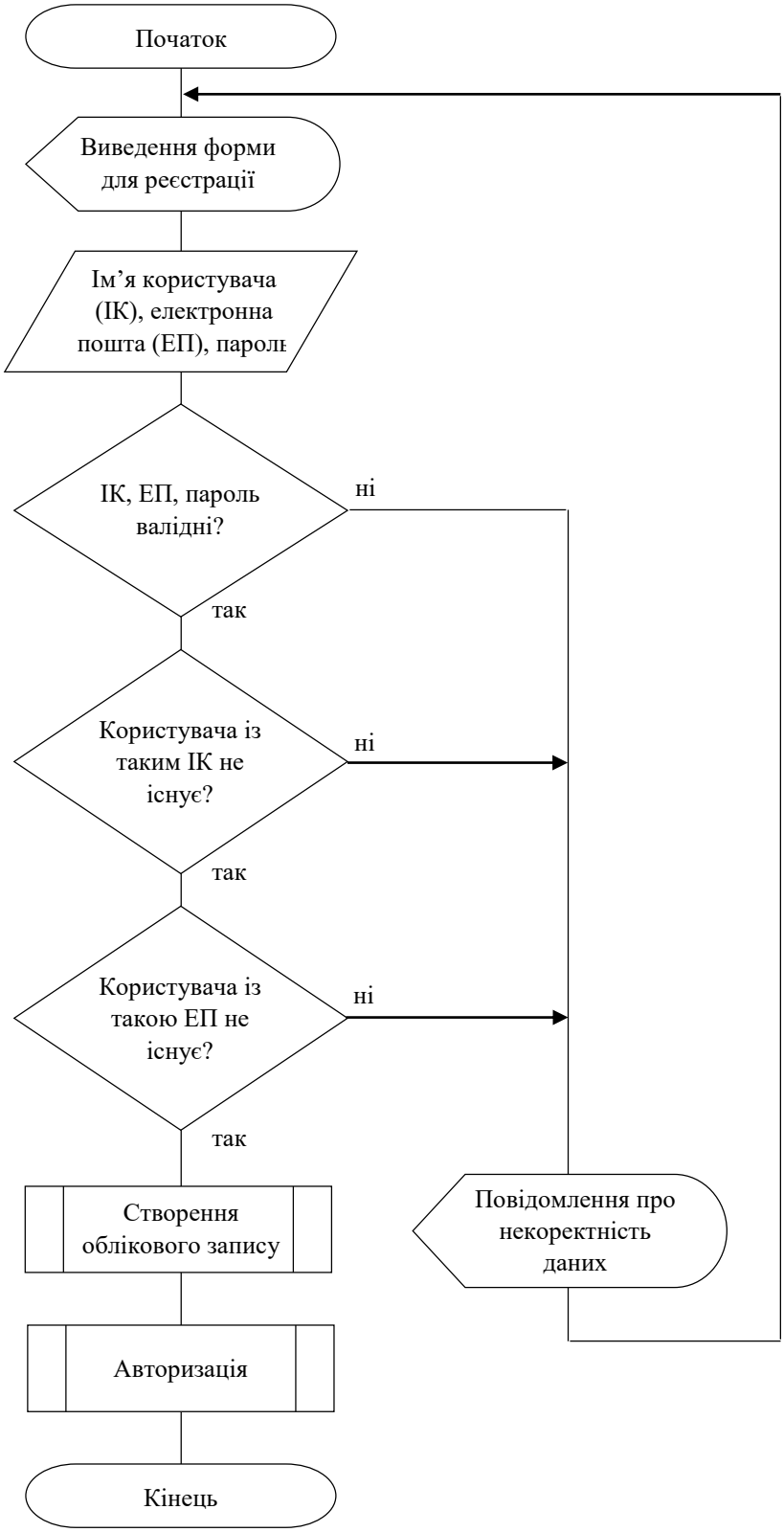


Рисунок 3.11 – Алгоритм реєстрації користувача

Так, для початку роботи із застосунком для поширення візуальних новел, користувач повинен відвідати відповідну адресу в браузері, яка відкриє йому веб-сторінку із формою, та зареєструватись, після чого йому надаються відповідні права та початковий пакет послуг. Якщо ж користувач вже зареєстрований, то йому потрібно пройти процедуру логіну.

Рисунок 3.12 – Алгоритм входу в обліковий запис користувача

Після відкриття форми для реєстрації у браузері користувач вводить свої персональні дані, необхідні для створення облікового запису – ім’я, адреса електронної пошти та пароль. Після надсилання форми отримані дані проходять

валідацію та перевірку, чи акаунтів із такими даними (ім’я та адреса) ще не існує. Якщо вони не відповідають вимогам, виводиться відповідне повідомлення та користувач повинен знову ввести необхідні дані. Далі створюється обліковий запис та користувачеві надаються необхідні права.

Якщо ж користувач відкриває форму логіну, то йому необхідно ввести ім’я, під яким він зареєстрований та пароль, надіслати ці дані на сервер. Там вони перевіряються на коректність та відбувається перевірка, чи такий користувач існує: звіряється ім’я та геш-код паролю. Якщо результати перевірки виявляються позитивними, то користувач авторизується, в іншому випадку виводиться повідомлення про некоректність введених даних і форма оновлюється.

3.5 Висновки з розділу

Побудова загального алгоритму роботи застосунку дозволила побачити більш детальні функції застосунку. Проведена у попередньому розділі побудова архітектури застосунку дозволила зрозуміти задачу кожного блоку та розробити більш детальні алгоритми роботи.

Так, узагальнений алгоритм роботи засобу складається із блоків з абстрактним описом виконуваної задачі. Кожна наступна схема уточнює кожен із цих блоків. Саме тому окремо було проведено побудову алгоритму роботи клієнтського застосунку і його блоків – реалізацію сюжетної логіки, алгоритми якої також було розглянуто спочатку загально, а після чого більш детально, та частину сервера і блоку автентифікації. З серверної частини було виділено основний алгоритм роботи та на його основі побудовано детальніші алгоритми роботи контролерів та репозиторіїв. З блоку автентифікації також розглянуто загальний алгоритм та окремо реалізацію реєстрації та входу в обліковий запис користувачем.

Така узагальнена побудова алгоритмів дозволяє підвищити ефективність розробки засобу, зменшити витрати часу та зусиль на нього, підвищити надійність застосунку, його продуктивність та допомагає зосередитись на основній функціональності.

4 ТЕСТУВАННЯ ЗАХИЩЕНОГО ЗАСТОСУНКУ

4.1 Обґрунтування засобів розробки

При обгрунтуванні засобів розробки варто розглядати десктопний застосунок, сервер та веб-інтерфейс окремо.

Основним засобом розробки є мова програмування, отже почати варто саме з неї. Найпопулярнішими мовами високого рівня для написання десктопних застосунків є Java, JavaScript, C#, Kotlin та Python [48]. Кожна з перелічених мов має свої переваги та недоліки. Так, JavaScript (JS) можна виконувати на різних платформах, ця мова є доступною та має велику кількість бібліотек, проте JS має дуже багато вразливостей у коді, особливо в контексті десктопних застосунків, проблеми з керуванням пам’яттю та для розробки десктопних застосунків має доволі обмежений перелік технологій та обмеження доступу до нативних функцій, і крім того відсутнє зручне середовище для розробки [49]. До переваг C# можна віднести надійну інтеграцію із Windows, багату екосистему розробки, продуктивність та зручне середовище розробки, проте недоліків у цієї мови недоліки більш вагомі, це – обмежена крос-платформенна підтримка, пряма залежність від рішень Microsoft (розробника) та від операційної системи Windows, оскільки середовище виконання коду може розгортатись лише на даній ОС [50]. Крім того, ця мова має значні проблеми із управлінням пам’яттю. Kotlin більш орієнтований на розробку під Android, тому теж можливими є проблеми із управлінням пам’яттю та ефективністю роботи застосунку, а Python хоч і має гарну незалежність від платформи та велику стандартну бібліотеку, проте він теж більше націлений на розробку мобільних застосунків та для веб.

На відміну від всіх вищеперерахованих мов, Java має найбільшу та найвагомішу кількість переваг, а недоліки є не надто суттєвими для даного виду розробки. Отже, до переваг даної мови програмування відносяться незалежність від платформи (принцип «напишіть раз, запустіть будь-де»), велика стандартна бібліотека, що робить розробку як власне ядра застосунку, так і графічного

інтерфейсу більш легкою. Java відома своєю міцністю та надійністю, оскільки має вбудовану систему керування пам’яттю (Garbage Collector – GC), а також продуктивність застосунків покращується за допомогою Just-In-Time компілятора (JIT Compiller). Також Java має активну підтримку розробників, тому постійно створюються нові версії і, що є дуже важливим, серед яких зберігається обернена сумісність. Тобто, код написаний, наприклад, на версії Java 1.8, буде працювати в версії, наприклад, 17.

Проте, мова також має певні недоліки – це витрати на продуктивність (оскільки код виконується на віртуальній машині Java – JVM), код є доволі громіздким а дизайн графічного інтерфейсу потребує використання додаткових бібліотек і, можливо, середовищ розробки.

Оскільки ці недоліки не є критичними, а переваг значно більше, ніж у інших популярних мов програмування, то для написання десктопного застосунку обрано саме Java.

Так, для розробки використовуватиметься Java Core, бібліотека Open Feign та JavaFX. Open Feign – бібліотека, що дозволяє створити HTTP-клієнт легко, швидко та якісно – на відміну від аналогічних REST-Template та Apache HTTP-client, які вимагають багато коду та часу на розробку клієнта і допускають наявність великої кількості помилок. Також Open Feign дозволяє легко інтегрувати засоби безпеки, такі, як використання токенів та, наприклад, SSL-сертифікатів. JavaFX – одна із бібліотек, які пропонує Java для розробки графічного інтерфейсу. Вона потребує меншої кількості коду, на відміну від ідентичних бібліотек Swing та AWT а також для полегшення процесу створення інтерфейсу було розроблено середовище Scene Builder, яке значно його пришвидшує – середовище також обрано для розробки десктопного застосунку. Найкращим фреймворком для блокового тестування, що пропонує Java, є JUnit.

Для розробки власне на Java існує декілька найзручніших середовищ – це IntelliJ IDEA, Eclipse та Apache NetBeans [51]. Останні два середовища відрізняються тим, що неефективні у використанні пам’яті та компіляція і виконання програм є більш повільною, ніж у першому середовищі. IntelliJ IDEA –

найкраще середовище для розробки мовою Java, оскільки більш ефективно використовує ресурси ПК, дозволяє встановлювати зв’язок із базою даних та власне може повністю замінити СКБД, також підтримує безліч корисних плагінів (наприклад, SonarLint та SpotBug для перевірки вразливостей коду). IntelliJ IDEA пропонує повний набір функцій, спеціально створених для розробки на Java, інтелектуальну підтримку коду, що виходить за рамки простого підсвічування синтаксису, зберігає локальну історію змін файлів, реалізує повну інтеграцію із системами збирання для застосунків мовою Java (Gradle, Maven) та контролю версій (Git, SVN, Mercurial, CVS) [52]. Крім того, надає дуже зручний та гнучкий інтерфейс та підтримує безліч мов програмування, розмітки та запитів – окрім Java ще JavaScript, TypeScript, HTML/CSS, xml, fxml, Kotlin, Groovy, SQL, HQL, GraphQL, Bash, PowerShell, JSON, YAML, Markdown [52].

Якщо переходити до розробки серверу, то найкращою мовою програмування також є Java, оскільки крім всіх вище перерахованих переваг, дана мова має одну особливість: під час тривалої роботи застосунку його ефективність не втрачається і час відповіді залишається таким самим, як і на початку його роботи – це відбувається за рахунок GC, який також вже згадувався вище. Інші мови програмування цієї функції не забезпечують. Оскільки основною вимогою до сервера є постійна його робота (надійність) та швидкість обробки запиту і надсилання відповіді, то Java все ж залишається найкращим варіантом.

Для розробки серверної частини краще всього використати фреймворк Spring Boot, який дозволяє пришвидшити розробку всіх необхідних рівнів сервера, а також сумісний із ним Spring Security полегшує інтеграцію засобів безпеки, а Spring Data JPA – інтеграцію із фактично будь-якою базою даних. Також перераховані фреймворки значно економлять час розробки та підвищують ефективність і надійність застосунків, чого не можуть забезпечити їх більш низькорівневі аналоги Tomcat та Jakarta Servlet. Для написання тестів найкращими фреймворками є JUnit та Mockito.

Для розробки веб-інтерфейсу найкращими варіантами є HTML та CSS які дозволять швидко та якісно створити веб-сторінку із зручним дизайном, а JS

дозволить описати функції для динамічного оновлення контенту сторінок. Оскільки функціонал веб-клієнта не є занадто широким, то для його створення достатньо буде JS.

Також важливою частиною розробки є система контролю версій. Найпопулярнішими серед них є Git, CVS, SVN та Mercurial [53]. Кожна з низ має низку своїх переваг та недоліків, проте, наприклад, порівняно із іншими, SVN вважається застарілою та у порівнянні із Git має набагато менший об'єм інструментів. CVS має клієнт-серверну модель репозиторіїв, що є не надто зручним, оскільки Git має розподілену модель репозиторіїв, а використання Mercurial є можливим лише якщо всі застосунки написані мовою Python. Отже, найкращою системою контролю версій є Git.

4.2 Основні семантичні одиниці програмного коду

Власне застосунок складається із двох модулів – клієнта (client, пакет com.visualnovel.client) та сервера (server, пакет com.visualnovel.server) (рис. 4.1).

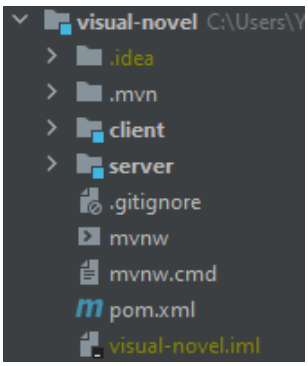


Рисунок 4.1 – Загальна структура застосунку

Модуль сервера містить в собі низку класів, які розділені по пакетам в залежності від свого призначення. Структура модуля представлена на рисунку В.1.

Основними пакетами є controllers для контролерів концепції MVC, який також містить у собі пакет rest для REST-контролерів, entities (для оголошення POJO класів та конфігурації таблиць і сутностей БД), який містить загальні класи, які стосуються всього застосунку (User, Role, Subscription Rank, Story) та тих, що стосуються історії і окремо для створення відповідей клієнту. Також модуль

містить пакет repositories для репозиторіїв, які являють собою інтерфейси, що успадковують інтерфейс JpaRepository, який представляє фреймворк Spring Data JPA для взаємодії із БД, та пакет services для класів-сервісів, які реалізовують репозиторії. Окремо в загальному пакеті розташовані класи для конфігурації, валідації, класу із змінними для файлів властивостей серверу. Там же розташований клас із точкою входу (функція public static void main()).

Зі всіх класів AuthorizationController, RegistrationController, UserRepository та SecurityConfig реалізують логіку реєстрації та автентифікації.

Проект має файли для налаштувань властивостей сервера – як загальних для визначення порту чи рівня логування, так і для налаштування взаємодії із електронною поштою та БД.

В розділі resources містяться сторінки веб-інтерфейсу.

Модуль клієнта містить пакет db.service, який забезпечує взаємодію із БД, feign_client, який складається із client_protocols для опису API-функцій для взаємодії із сервером та handlers для опису функцій із використанням API (рис. В.2). Також модуль містить окремо пакети novel_controls та handlers – перші описує компоненти графічного інтерфейсу, що використовується в застосунку, а другий використовує їх для зміни та показу сцен. Також є пакет story із класами для взаємодії із сервером та надсилання йому запитів. В загальному пакеті розміщені класи-контролери, що реалізують взаємодію клієнтського застосунку із користувачем та клас із основними конфігураціями і точкою входу.

У розділі resources містяться файли розширення fxml – вони описують власне графічний інтерфейс та його дизайн.

4.3 Здійснення обфускації програмного коду засобу

Обфускація є важливою частиною захисту ресурсів застосунку. Під час апробації було оформлено авторське свідоцтво на програму ObfusLex, яку можна було б використати для обфускації, але ця програма використовує лише лексичну

обфускацію, якої для надійного захисту буде недостатньо [3]. Тому, крім цього варто використати також обфускацію даних та графу потоку керування.

Для прикладу варто взяти метод на стороні клієнтського застосунку, який здійснює перевірку токена, проводячи його валідацію та надсилаючи на сервер. Початковий вигляд функції представлений на рисунку 4.2.

```
@FXML
void onSendTokenButtonClicked(MouseEvent event) {
    String token = tokenField.getText();
    if (!Validator.validateToken(token)) {
        messageField.setText("Token is not valid!");
        tokenField.setText("");
    } else {
        AuthHandler authHandler = new AuthHandler();
        int status = authHandler.getTokenResp(Auth.getPlayer().getLogin(), Auth.getPlayer().getPassword(), token).status();
        if (status == 200) {
            Scene scene = sceneUtil.getScene(MainMenuController.class, fileName: "main-menu-view.fxml");
            Main.getMainStage().setScene(scene);
        } else {
            messageField.setText("Token is not valid!");
            tokenField.setText("");
        }
    }
}
```

Рисунок 4.2 – Початковий вигляд методу

Для того, щоб зберегти функціональність методу, варто розпочати із обфускації потоку керування та даних. Отже, для початку, потрібно ввести додаткові логічні оператори, після чого створити додаткові змінні, присвоїти їм значення та зв’язати таким чином, щоб в результаті вийшло число, з яким проводиться перевірка в методі.

Також потрібно додати недосяжний код – частина функцій, які не виконують ніякого сенсового навантаження, не виконуються під час запуску застосунку та тільки підвищують складність аналізу зловмисником.

Вигляд методу після обфускації даних та графа потоку керування представлено на рисунку 4.3.

```java
@FXML
void onSendTokenButtonClicked(MouseEvent event) {
    double a = 12.5;
    String token = tokenField.getText();
    Validator.validateToken(token);
    int b = 2;
    if (!Validator.validateToken(token)) {
        messageField.setText("Token is not valid!");
        tokenField.setText("");
    } else {
        int c = b+6;
        AuthHandler authHandler = new AuthHandler();
        int status = authHandler.getTokenResp(Auth.getPlayer().getLogin(), Auth.getPlayer().getPassword(), token).status();
        double d = 0;
        while(d<b){
            d+=0.01;
            if (d>12){
                token = token.substring(token.indexOf("w"), token.indexOf("4"));
                while(token.length()<a*40.96){
                    String rand = "sa;dklfgwopeguwopugewp[mz,xal;skdwitq=-w)(";
                    int randN = new Random().nextInt(rand.length());
                    token+=rand.charAt(randN);
                }
            }
        }
        int checkStatus = (int)a*(int)d*c;
        if (status == checkStatus) {
            Scene scene = sceneUtil.getScene(MainMenuController.class,  fileName: "main-menu-view.fxml");
            Main.getMainStage().setScene(scene);
        } else {
            if(checkStatus==d*b){
                messageField.setText("Token is valid");
            }
            messageField.setText("Token is not valid!");
            tokenField.setText("");
        }
    }
}
```

Рисунок 4.3 – Вигляд методу після обфускації даних та графа потоку керування

Далі варто здійснити лексичну обфускацію – перейменувати методи, класи, їх екземпляри та інші змінні (примітиви) на випадкові послідовності символів та прибрати форматування (рис. 4.4).

```java
@FXML
void NLfqEgyC3N2VVoomqVkrIDKyBKTcV0b(MouseEvent kIcZA3) {
double HajlkE=12.5;String FuDqzC6h0S2JdqUo=wixexZkfAfj3.getText();
VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo);int AFQhPmOB6PCey=2;
if(!VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo)){XuS5eZETFxSpEIF.setText(asflAW);wixexZkfAfj3.setText("");}else{
int xexZk_fAfj__3TF5J5_i9U=AFQhPmOB6PCey+6;UYY3jyQT3 xexZk_fAfj_=new UYY3jyQT3();
int Zd3422aJK5QG1U9_RnMs=xexZk_fAfj_.d9W8i5PMK1r50zu(uuKD0EzV.MCGgpdurB().XT829_B2(),
uuKD0EzV.MCGgpdurB().qHYJGseMpkF(),FuDqzC6h0S2JdqUo).status();
double KyuTt5_GN6UR_=0;while(KyuTt5_GN6UR_<AFQhPmOB6PCey){KyuTt5_GN6UR_+=0.01;if(KyuTt5_GN6UR_>12){
FuDqzC6h0S2JdqUo=FuDqzC6h0S2JdqUo.substring(FuDqzC6h0S2JdqUo.indexOf("w"),FuDqzC6h0S2JdqUo.indexOf("4"));
while(FuDqzC6h0S2JdqUo.length()<HajlkE*40.96){
String E9QSO6f="sa;dklfgwopeguwopugewp[mz,xal;skdwitq=-w)(";int fPv9=new Random().nextInt(E9QSO6f.length());
FuDqzC6h0S2JdqUo+=E9QSO6f.charAt(fPv9);}}}int nnDotlRFb3w33=(int)HajlkE*(int)KyuTt5_GN6UR_*xexZk_fAfj__3TF5J5_i9U;
if(Zd3422aJK5QG1U9_RnMs==nnDotlRFb3w33){Scene EqSPZxUzta=qVmzLWLn.su9GoL5_7grYQtu(X7hZhRBkptOza.class,SLasDKaFAdfWaEOPG);
LPYN4nkhhBMKosw1eZ.jWjGCPkqSPZx().setScene(EqSPZxUzta);}else{if(nnDotlRFb3w33==KyuTt5_GN6UR_*AFQhPmOB6PCey){
XuS5eZETFxSpEIF.setText(asdfkSopwDFSjegASDFiv);}
XuS5eZETFxSpEIF.setText(asflAW);wixexZkfAfj3.setText("");}}}
```

Рисунок 4.4 – Вигляд методу після проведення всіх етапів обфускації

І хоч форматування повертається за допомогою засобів середовища розробки, аналіз функції все одно ускладнюється за рахунок заплутування її логіки.

Крім того, в результаті проведеної обфускації продуктивність застосунку не змінилась, і все ще займає мілісекунди (рис. 4.5-4.6).

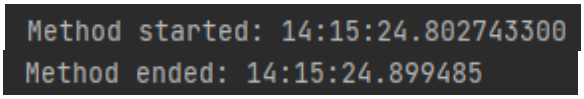


Рисунок 4.5 – Час виконання методу до обфускації

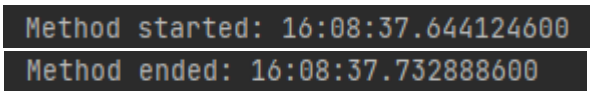


Рисунок 4.6 – Час виконання методу після обфускації

Як видно з рисунків 4.5 та 4.6 тривалість виконання не зазнала негативного впливу, навіть більше вона змінилась з 0,0964 с до 0,0888 с, що пояснюється завантаженням процесора фоновими задачами, і оскільки вони більше впливають на тривалість виконання методу, ніж обфускація, то можна вважати, що захист виконано вдало.

Отже, в такий спосіб обфускація може підвищити рівень захищеності коду від несанкціонованого аналізу.

4.4 Блокове тестування захищеного застосунку

Блокових функцій у застосунку невелика кількість, проте всі вони обов’язково потребують проведення процедури блокового тестування. Класи, що потребують такого тестування відносяться до частини клієнтського застосунку, це – Validator, ConditionUtil та абстрактний клас Provider.

Перший клас містить статичні методи для валідації адреси електронної пошти, імені користувача та паролю, другий – містить методи для парсингу умов для виборів. Ці два класи тестуються напряму за допомогою простого виклику методів. Для проведення тестування останнього названого класу необхідно створити його імплементацію, і тільки тоді здійснювати процедуру.

Для проведення тестування створюються методи – параметризовані тести із тест-кейсами. Так, для всіх класів тест-кейси будуть позитивними, тобто являтимуть собою коректні дані та різні їх варіації і комбінації, та негативними – некоректними, включно із «пустими» об’єктами.

Так, наприклад, для методу, що проводить валідацію паролю позитивні тестові випадки мають такий вигляд:

– p@ssw0rD;
– p1A2s3s4#ord;
– p@sswo1Rd.

Перелік негативних виглядає так:

– passworD;
– p@ssworD;
– password;
– p@ssword;
– p@ssw0rd;
– password1;
– passw0rD.

Також обов’язково метод повинен обробляти випадки, коли на вхід подається пустий рядок, або напряму null, тому для тесту є ще декілька тест-кейсів:

– “”;
– “ ”;
– “ ”;
– null.

При позитивному результаті, тобто якщо пароль пройшов валідацію, метод повертає true, якщо ж ні – false, тому при проведенні тестування перевіряється саме відповідність очікуваному результату та відсутність помилок при виконанні методу.

Так, якщо виконати тести із перерахованими вище тестовими випадками, то буде виведено результат, зображений на рисунку 4.7.

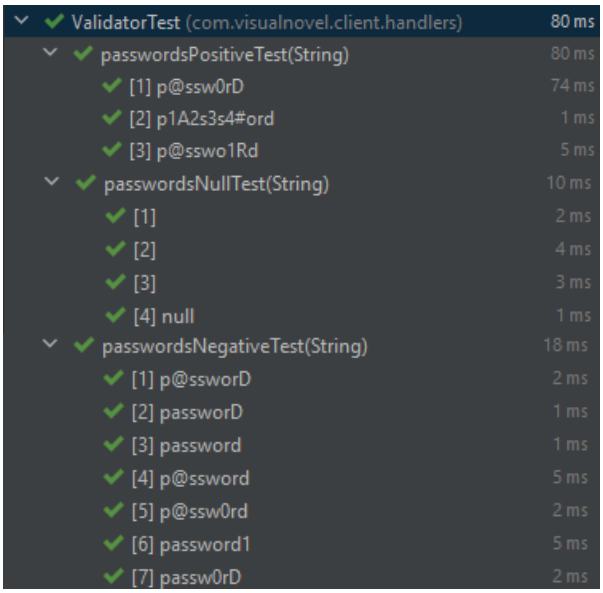


Рисунок 4.7 – Результати виконання тестів для метода валідації паролів

Аналогічним чином створюються та виконуються тести для інших методів класу Validator, та всіх методів класу ConditionUtil. Результати їх виконання зображено на рисунку 4.8 і рисунку 4.9 відповідно.

Тест-кейси всіх методів охоплюють максимальну кількість можливих комбінацій як для позитивних, так і негативних їх видів.

Варіанти для перевірки обробки методами «порожніх» вхідних даних ідентичні у всіх тестах.

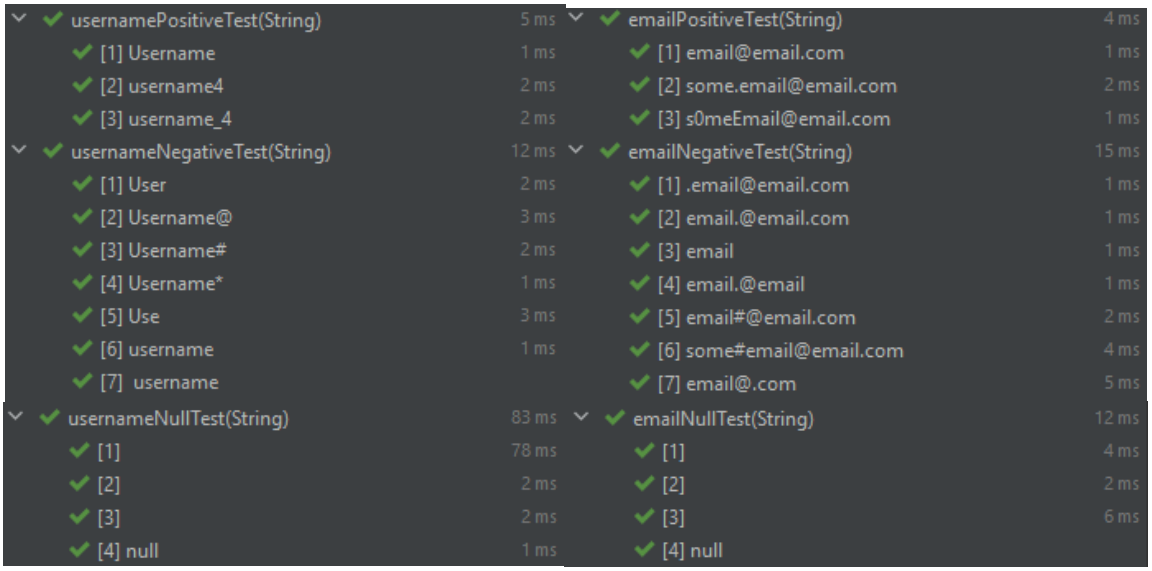


Рисунок 4.8 – Результати виконання тестів для методів валідації адрес електронної пошти та імен користувачів

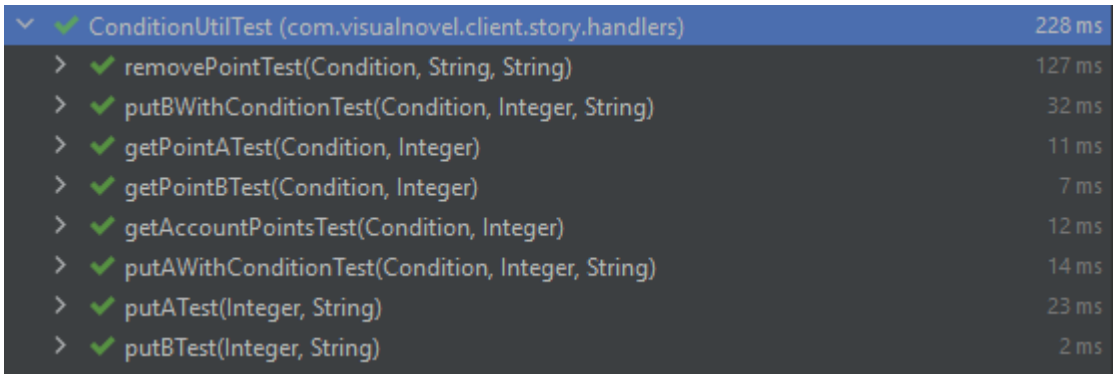


Рисунок 4.9 – Результати виконання тестів для методів класу ConditionUtil

Для виконання тестів для методів класу Provider створено внутрішній клас-нащадок, якого вже можуть використовувати тести. Результати виконання тестів зображено на рисунку 4.10.

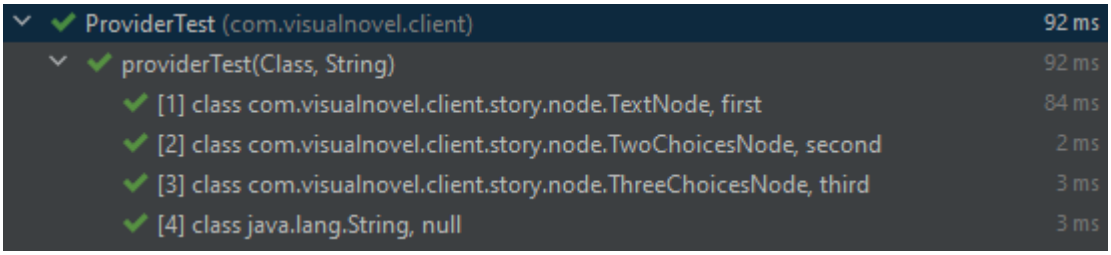


Рисунок 4.10 – Результати виконання тестів для методів класу Provider

Отже, результати всіх виконаних тестів є позитивними, тому можна припустити, що розробка виконана коректно.

4.5 Інтеграційне тестування захищеного застосунку

Інтеграційне тестування – це тестування взаємодії різних модулів застосунку. В розробленому засобі окремої уваги та обов’язкового тестування потребує зв’язок застосунку із базою даних, тобто потрібно провести тестування роботи репозиторіїв та сервісів. Тестування варто проводити окремо на частині клієнта та сервера.

Так, БД клієнта є досить компактною та містить лише одну таблицю, тому описано лише один клас-репозиторій – SettingsRepository, тестування якого варто провести. Це клас має два методи: для зміни існуючого запису та його отримання.

І оскільки запис лише один, і ідентифікатор його не змінюється, то тест-кейс для другого методу буде лише один – виклик методу getById(1). Тестові випадки для першого методи виглядають таким чином:

– new Settings(0.3);
– new Settings(0));
– new Settings(1);
– new Settings(-1).

Результати виконання тестів представлено на рисунку 4.11.

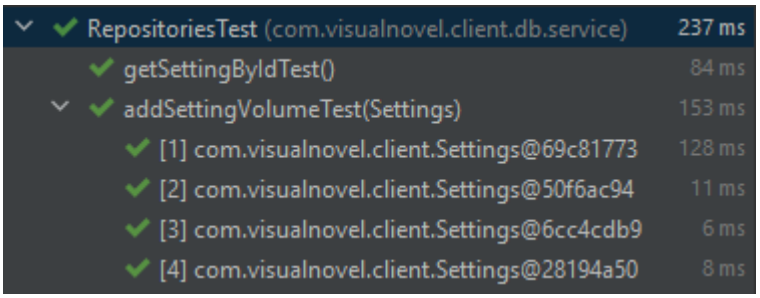


Рисунок 4.11 – Результати тестування класу SettingsRepository

Далі варто перейти до сервера. Запускається він за 7.4 секунди (рис. 4.12).

```
2023-06-17 02:10:45.666  INFO 13172 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 9.321 seconds (JVM running for 10.761)
2023-06-17 02:11:51.586  INFO 10696 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 7.516 seconds (JVM running for 8.88)
2023-06-17 02:12:19.640  INFO 15508 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 6.287 seconds (JVM running for 7.274)
2023-06-17 02:12:45.713  INFO 16768 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 6.593 seconds (JVM running for 7.574)
```

Рисунок 4.12 – Результати декількох запусків серверу

Оскільки всі класи-репозиторії реалізовані за одним алгоритмом, то тестування потребують лише основні та ті, що беруть участь у автентифікації, а саме класи TextNodeService, RoleService та UserService.

TextNodeService – клас, що надсилає запити на отримання чи додавання сцени до бази даних. Сцена містить у собі персонажів та зображення, тому під час використання даного класу викликаються методи інших сервісів та репозиторіїв, саме тому в центрі уваги при тестуванні буде саме цей клас.

Отже, сервіс реалізує два методи репозиторію – для отримання та додавання об’єктів до бази даних. На вхід методів подаються елементи об’єктів, тому для виконання тестів тест-кейси такі:

– 100L, "100TextNode", personageService.getById(10L), imageDataService.getImage(21L), imageDataService.getImage(21L);
– "TextNode without ID", personageService.getById(10L), imageDataService.getImage(21L), imageDataService.getImage(21L);
– 101L, "", personageService.getById(10L), imageDataService.getImage(21L), imageDataService.getImage(21L);
– 102L, "102 TextNode", null, imageDataService.getImage(21L), imageDataService.getImage(21L);
– 103L, "103 TextNode", personageService.getById(10L), null, imageDataService.getImage(21L);
– 104L, "104 TextNode", personageService.getById(10L), imageDataService.getImage(21L), null.

Результати виконання тестів зображено на рисунку 4.13.

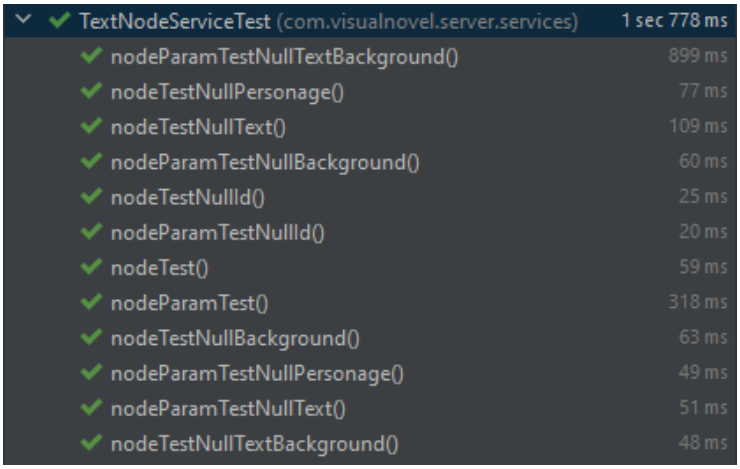


Рисунок 4.13 – Результати тестування класу TextNodeService

Аналогічним чином розробляються тести для класів RoleService та UserService. Результати виконання тестів представлені на рисунку 4.14.

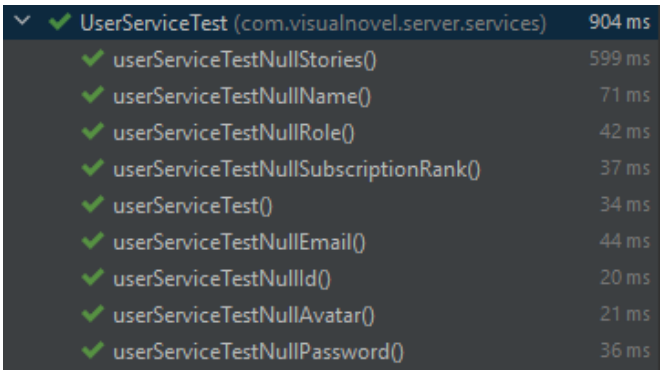


Рисунок 4.14 – Результати тестування взаємодії із БД за допомогою класів UserService, RoleService, SubscriptionRankService та ImageDataService

Отже, результати тестування є позитивними, тому можна зробити припущення, що взаємодія застосунку із БД реалізована коректно.

4.6 Мануальне тестування захищеного застосунку

Мануальне тестування є важливою частиною розробки та тестування застосунку.

Отже, почати варто із веб-інтерфейсу. Для цього потрібно в браузері спробувати перейти за такими посиланнями:

– http:/localhost:8085/;
– http:/localhost:8085/registration;
– http:/localhost:8085/visual-novels-store;
– http:/localhost:8085/user-home;
– http:/localhost:8085/user-home/edit.

При спробі неавторизованого користувача перейти за адресою http:/localhost:8085/ сервер перенаправляє його за адресою http:/localhost:8085/login – це власне форма для автентифікації (рис. Г.1).

Також, якщо користувач не зареєстрований, він може її здійснити, перейшовши за адресою http:/localhost:8085/registration – йому відкриється форма для реєстрації (рис. Г.2).

Після проведення реєстрації чи автентифікації користувач перенаправляється на введену спочатку адресу – але оскільки саме та адреса перенаправляє на http:/localhost:8085/visual-novels-store, то користувачу виводиться результат, зображений на рисунку Г.3.

Також користувач має змогу відредагувати персональні дані (рис. Г.4).

Отже, все відображається, та працює коректно. Далі варто перейти до тестування десктопного застосунку.

При автентифікації користувачеві на вказану адресу електронної пошти було надіслано лист із токеном для автентифікації (рис. 4.15). Цей токен потрібно ввести при вході у застосунок – у вікні, що відкривається, є відповідне поле для вводу.

Рисунок 4.15 – Вигляд отриманого електронного листа із токеном

Але для початку потрібно прив’язати застосунок до облікового запису, ввівши у початковому вікні логін та пароль (рис. Г.5). Після введення і валідації даних з’являється вікно для введення токена (рис. Г.6).

Після введення користувачем токена відбувається його перевірка та в залежності від результату відкривається вікно із меню вибору, або поле очищується і виводиться повідомлення про невірний токен (рис. Г.7).

Меню представлено на рисунку Г.8. Вікно, що відкрилось, має три пункти: обрати історію, змінити налаштування та вихід.

Для початку варто перевірити налаштування. Відкривається вікно, вигляд якого зображено на рисунку Г.9. Налаштування дають змогу змінити гучність звуку, яким супроводжується ігровий процес, тому потрібно спробувати його змінити.

Автоматично звук налаштований на максимум, а для прикладу можна поставити відмітку на 80 – при збереженні результату гучність музики зменшилась.

Кнопка «Save» зберегла зміни та повернула користувача до головного меню. Отже, далі – вибір новели. При виборі цього пункту відкривається вікно з доступними історіями (рис. 4.16).

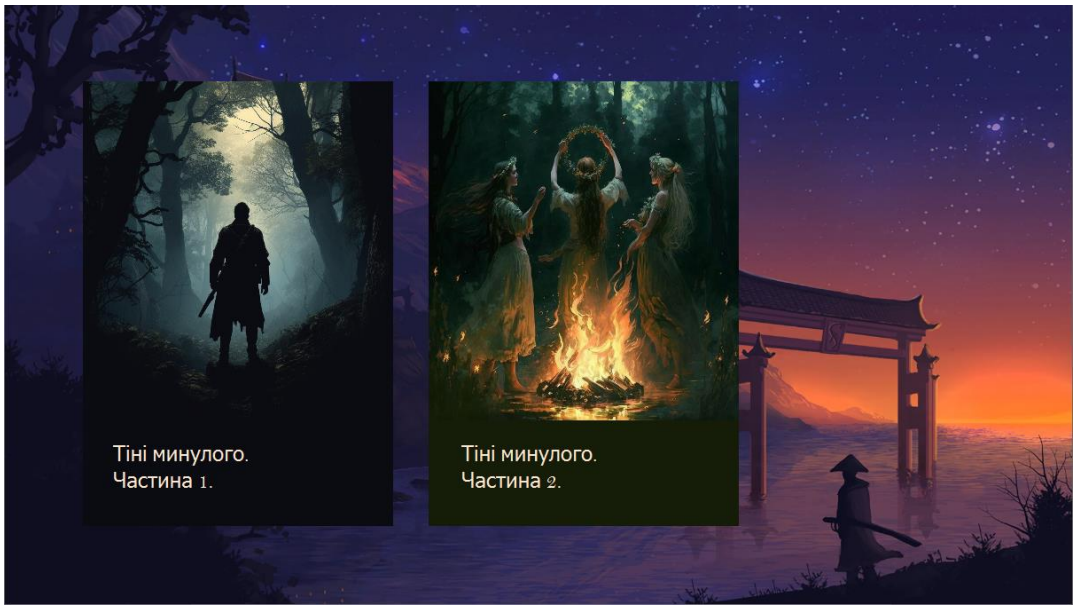


Рисунок 4.16 – Вигляд вікна із вибором доступних історій

Для початку – перша історія. При натисненні на історію «Тіні минулого. Частина 1» відкривається вікно із повідомленням «Heres you story begin» про початок історії, зображеним на рисунку Г.10.

Щоб почати саму історію – потрібно натиснути на будь-яке місце у вікні. Ту ж саму дію потрібно повторити для переходу до наступної сцени.

При проходженні історії варто перевірити не тільки реалізацію перебігу сюжету, а і виведення зображень та тексту на екран. Так, текст друкується поступово по символу із достатньою швидкістю, зміна зображень відбувається плавно, отже анімація працює коректно. Також варто перевірити власне зміну зображень. В історії присутні сцени, де герой як такий відсутній, і розповідь іде від оповідача (третьої особи), тому ім’я персонажа та його зображення повинні бути відсутніми (рис. Г.11). На сцені із персонажем присутні як ім’я, так і зображення (рис. 4.17).

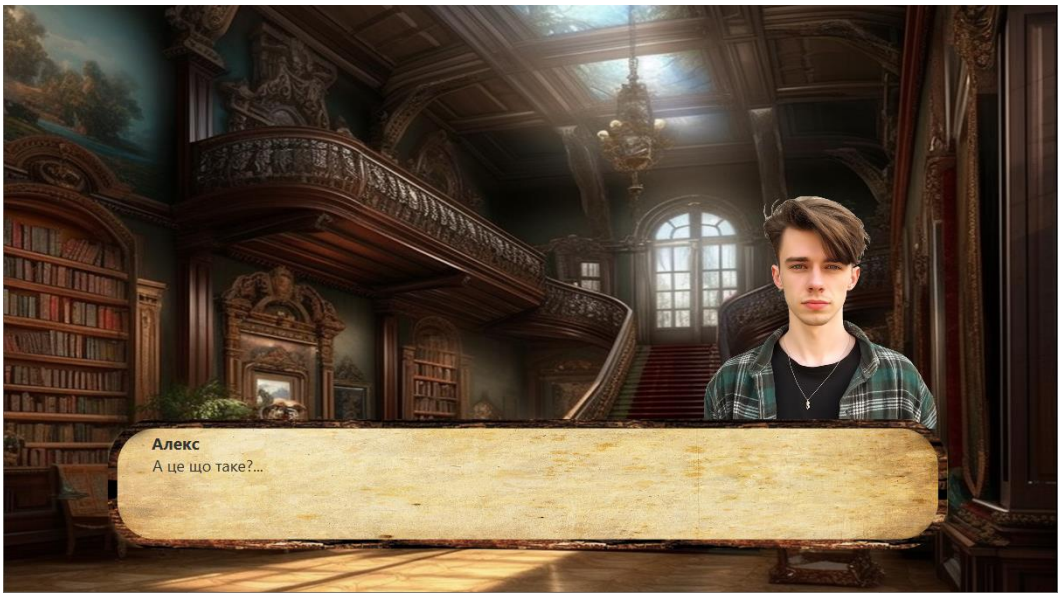


Рисунок 4.17 – Вигляд сцени із персонажем

Також персонажі змінюють свої емоції, тому варто перевірити роботу цієї функції також. Приклад зміни емоцій персонажа зображено на рисунку 4.18.

Оскільки обрана історія працює та відображається коректно, варто перейти до наступної.

Рисунок 4.18 – Приклад зміни емоції персонажа

При виборі «Тіні минулого. Частина 2» також відкривається вікно із повідомленням про початок історії. Із натисненням в будь-якому місці вікна відкривається перша сцена. Якщо звернути увагу на сюжет, то можна помітити його розвиток, динамічну зміну персонажів, велику кількість сцен та можливість

вибору шляху, за яким буде розвиватись історія. Так, у обраній історії присутні сцени із двома та трьома варіантами вибору, в залежності від яких відбуватиметься розвиток подій, тому варто також перевірити сюжетну логіку. Отже, на рисунку Г.12 зображено приклад розвитку сюжету та динамічну зміну сцен: змінюються не тільки персонажі та їх репліки, а і розміщення, а також присутня можливість вибору.

Хід історії повинен відрізнятись в залежності від вибору, який зробить гравець. Отже, варто перевірити обидва варіанти: при виборі першого вигляд сцени змінюється на зображену на рисунку Г.13.

Якщо обрати другий варіант, то наступна сцена відрізнятиметься (рис. Г.14).

Оскільки при мануальному тестуванні не було виявлення збоїв у показі сцен, перебігу сюжету та переході між вікнами, то можна зробити вигляд, що застосунок працює коректно.

4.7 Висновки з розділу

Правильний вибір мови програмування та технологій для розробки застосунку є одним із ключів до вдалого результату. Так, Java ідеально підходить для розробки коду серверів та надає достатній та зручний стек технологій для розробки десктопних застосунків.

Також важливим фактором є структура застосунку – розділення сервера та клієнта на модулі дозволяє фізично відділити функціонал сервера та клієнта, але їх розміщення в одному проекті на час розробки дозволяє полегшити цю процедуру.

Важливою частиною розробки застосунку є його тестування (блокове, інтеграційне та мануальне). Проведення тестування та його результати дають змогу оцінити стан готовності застосунку, робити припущення щодо його надійності та перевірити, чи застосунок працює загалом. Так, блокове тестування показало, що окремі модулі працюють так, як передбачувалось та без помилок, виконують свої функції, інтеграційне показало взаємодію між розробленими та протестованими раніше модулями, а мануальне дало змогу побачити, що графічний інтерфейс

відображається так, як потрібно, а більшість процесів (ті, які було перевірено), працюють та виконують свої функції. Отже, можна зробити висновок, що застосунок працює.

Важливою частиною застосунку є його безпека – захист від загроз, проаналізованих у першому розділі. Так, однією із них є несанкціонований доступ та аналіз зловмисником коду. Попередити та зменшити ймовірність їх виникнення дозволяє обфускація. Важливою умовою при здійсненні обфускації є збереження функціоналу коду, що захищається, а оскільки тестування проводилось після обфускації, можна зробити висновок, що умова виконується. Також в результаті захисту код став менш зрозумілим та важчим для аналізу, тому захищеність застосунку, що є основною ціллю роробки, – зросла.

ВИСНОВКИ

Важливими аспектами стану захищеності є цілісність, доступність та конфіденційність. Запропонований у даній роботі підхід до захисту засобу для поширення візуальної новели забезпечує останніх два, в той час як захист цілісності покладений на іншу частину комплексної бакалаврської дипломної роботи.

Під час аналізу загроз комп'ютерним іграм було визначено основні загрози та на їх основі проаналізовано основні засоби захисту прикладних застосунків та комп'ютерних ігор, а також проведено порівняльний аналіз відомих ігор та методів їх захисту: всі проаналізовані ігри зламані, проте у жодної не реалізовано повноцінний захист (від несанкціонованого доступу, копіювання, захисту від шахрайства). Отже, було поставлено задачу розробити захищений засіб для поширення новел та на його прикладі запропонувати новий підхід для захисту комп'ютерних ігор від несанкціонованого копіювання.

На основі проведених аналізів та із врахуванням їх результатів було побудовано архітектуру застосунку – клієнт-серверна, що є основою концепції запропонованого захисту. Оскільки всі дані, необхідні для ігрового процесу, зберігаються в базі даних на стороні серверу, а частина клієнтського застосунку фактично порожня з точки зору кількості ігрового контенту, то копіювати його сенсу немає, отже розробники не втрачатимуть прибуток від розробки. Такий підхід частково вирішує проблему із піратством в індустрії комп'ютерних ігор. Але із такою концепцією постає питання виникнення інших загроз: несанкціонований доступ до сервера та перехоплення даних. Саме тому було запропоновано метод захисту конфіденційних ресурсів та даних: реалізація автентифікації на сервері, використання протоколу HTTPS при запитах та використання навісного захисту для програмного коду – обфускації.

Керуючись результатами розробки архітектури застосунку, було розроблено алгоритми роботи окремо сервера та клієнта, модуля автентифікації і їх блоків: взаємодії із базами даних, реалізація сюжетної логіки, контролери та HTTP-клієнт.

Розроблені алгоритми дозволили полегшити процес написання коду та тестування всіх модулів застосунку. В результаті проведення тестування було зроблено висновок, що розроблений застосунок працює коректно, що дозволило експериментально довести коректність ухвалених технічних рішень під час процесу проєктування захищеного застосунку.

Отже, запропонований підхід до захисту, а саме розміщення всіх ігрових ресурсів на стороні сервера, автентифікація користувачів при вході до сервера та заборона надсилати запити на сервер не автентифікованим користувачам, використання HTTPS протоколу та здійснення обфускації коду, забезпечує конфіденційність та цілісність ігрових ресурсів. Недоліком такого захисту є те, що він все одно залишає частину потенційних вразливостей зі сторони веб-інтерфейсу, а тому потребує динамічного дослідження своєї безпеки з використанням інструментарію етичного хакера.

Застосунок має перспективи для розвитку як в плані функціоналу, так і захисту, це – розширення веб-інтерфейсу, його захист, реалізація внутрішньоігрових придбань та реалізація захисту грошових транзакцій відповідно до відомих стандартів та найкращих практик.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Насталенко Я. І. Дослідження способів захисту візуальних новел від несанкціонованого доступу. м. Вінниця / Наук. керівник Ю. В. Баришев. Вінниця, ВНТУ 2023.
2. Насталенко Я. І. Дослідження ефективності захисту програмного коду шляхом лексичної обфускації. м. Вінниця / Наук. керівник В. А. Каплун. Вінниця, 2022.
3. Комп'ютерна програма "ObfusLex" : а. с. 112909 Україна / Я. І. Насталенко, В. А. Каплун. Опубл. 11.05.2022.
4. Комп’ютерна програма "KERBEROS" : а. с. 108551 Україна / А. В. Остапенко-Боженова, Я. І. Насталенко. Опубл. 11.10.2021.
5. Increasing Cyberattacks Targeting the Gaming Industry in 2022. *SOCRadar*. URL: https://socradar.io/increasing-cyberattacks-targeting-the-gaming-industry-in-2022/ (accessed: 23.05.2023).
6. Account Takeover. *imperva*. URL: https://www.imperva.com/learn/application-security/account-takeover-ato/ (accessed: 25.05.2023).
7. Targetted DDoS attacks are still a problem in Apex Legends, ImperialHal has trouble playing the game. *Sportskeeda*. URL: https://www.sportskeeda.com/esports/targetted-ddos-attacks-still-problem-apex-legends-imperialhal-still-play-game (accessed: 25.05.2023).
8. Study Reveals Games With the Most Cheaters. *Gamerant*. URL: https://gamerant.com/video-game-cheats-gta-sims/ (accessed: 27.05.2023).
9. Tsotsorin M. V. Piracy and Video Games: Is There a Light at the End of the Tunnel? *SSRN Electronic Journal*. 2012. URL: https://www.researchgate.net/publication/2-96876800_Piracy_and_Video_Games_Is_There_a_Light_at_the_End_of_the_Tunnel#pf3 (accessed: 27.05.2023).
10. Top 10 most pirated PC games. *MyGaming | The best gaming website and forum in South Africa*. URL: https://mygaming.co.za/news/features/70046-top-10-most-pirated-pc-games (accessed: 27.05.2023).

11. What Is Ransomware? *Trellix*. URL: https://www.trellix.com/en-us/security-awareness/ransomware/what-is-ransomware.html (accessed: 25.05.2023).
12. Riot Games confirms ransomware. *Cybernews*. URL: https://cybernews.com/news/riot-games-ransom-demand/ (accessed: 25.05.2023).
13. What are game mods? Are mods safe to install?. *Atlas VPN*. URL: https://atlasvpn.com/blog/malware-in-game-mods-and-unlicensed-software (accessed: 25.05.2023).
14. Javeed D. Man in the Middle Attacks: Analysis, Motivation and Prevention. *International Journal of Computer Networks and Communications Security*. 2020. Vol. 8, no. 7. P. 52–58. URL: https://www.researchgate.net/publication/347006863_Man_in_the_Middle_Attacks_Analysis_Motivation_and_Prevention (accessed: 25.05.2023).
15. Дудатьєв А. В., Каплун В. А., Семеренко В. П. Захист програмного забезпечення. Частина 1 : навчальний посібник. Вінниця : ВНТУ, 2005. 140 с.
16. Ted Harrington. Hackable: How to Do Application Security Right. Lioncrest Publishing, 2020. 288 p.
17. Finn Brunton, Helen Nissenbaum Obfuscation: a user’s guide for privacy and protest. Gildan Media, LLC, 2015. 123 p.
18. Каплун В. А., Дмитришин О. В., Баришев Ю. В. Захист програмного забезпечення. Частина 2 : навчальний посібник. Вінниця : ВНТУ, 2014. 105 с.
19. In the Jungle of .NET Obfuscator Tools. *Ndepend*. URL: https://blog.ndepend.com/in-the-jungle-of-net-obfuscator-tools/ (accessed: 27.05.2023).
20. Obfuscation and Java Obfuscators. *Incusdata*. URL: https://incusdata.com/blog/obfuscation (accessed: 27.05.2023).
21. Boonkrong S. Authentication and Access Control: Practical Cryptography Methods and Tools. Apress, 2020. 248 p.
22. Mahalle P. N., Bhong S. S., Shinde G. R. Authorization and Access Control Foundations, Frameworks, and Applications. CRC Press, 2022. 86 p.

23. Eriksson F., Östlund J. Evaluation of Software License Management Frameworks for Grid Environments: The Four Ts for Agile Systems : Master's thesis. 2014. 140 p. URL: https://www.diva-portal.org/smash/get/diva2:696982/-FULLTEXT01.pdf (accessed: 27.05.2023).
24. The importance of input validation and error handling. *Bootcamp*. URL: https://bootcamp.uxdesign.cc/the-importance-of-input-validation-and-error-handling-5359a4cd7a80 (accessed: 27.05.2023).
25. What is access control?. *Citrix*. URL: https://www.citrix.com/solutions/secure-access/what-is-access-control.html (accessed: 27.05.2023).
26. Understanding the significance of anti-tamper solutions. *Appsealing*. URL: https://www.appsealing.com/anti-tamper-solutions/ (accessed: 27.05.2023).
27. Виявлення, стримання, відновлення: основні етапи в процесі управління інцидентами. *ISSP Training Center*. URL: https://www.issp.training/post/виявлення-стримання-відновлення-основні-етапи-в-процесі-управління-інцидентами (дата звернення: 27.05.2023).
28. Patch Management: Definition & Best Practices. *Rapid*. URL: https://www.rapid7.com/fundamentals/patch-management/ (accessed: 27.05.2023).
29. Digital Rights Management (DRM). *Fortinet*. URL: https://www.fortinet.com/resources/cyberglossary/digital-rights-management-drm (accessed: 26.05.2023).
30. What Is Denuvo and Why Do Some Gamers Hate It?. *Make Use Of*. URL: https://www.makeuseof.com/what-is-denuvo/ (accessed: 27.05.2023).
31. Denuvo. *Codex Gamicus*. URL: https://gamicus.fandom.com/wiki/Denuvo (accessed: 27.05.2023).
32. The Best Anti-Piracy Techniques Ever Seen in Video Games. *MUO. January 19, 2023.* : URL : https://www.makeuseof.com/best-anti-piracy-techniques-in-video-games. (accessed: 27.05.2023).
33. Додому. *Hogwarts Legacy*. URL: https://www.hogwartslegacy.com/uk-ua (дата звернення: 27.05.2023).

34. Cyberpunk 2077 – from the creators of The Witcher 3: Wild Hunt. *Cyberpunk 2077 – from the creators of The Witcher 3: Wild Hunt*. URL: https://www.cyberpunk.net/us/en/ (accessed: 27.05.2023).
35. Electronic Arts. Apex Legends - The Next Evolution of Hero Shooter - Free to Play. *Electronic Arts Home Page - Official EA Site*. URL: https://www.ea.com/en-gb/games/apex-legends (accessed: 27.05.2023).
36. Mobile Legends: Bang Bang. *Mobile Legends: Bang Bang*. URL: https://m.mobilelegends.com/en (accessed: 27.05.2023).
37. The Witcher 3: Wild Hunt - Official Website. *The Witcher 3: Wild Hunt - Official Website*. URL: https://www.thewitcher.com/us/en/witcher3 (accessed: 27.05.2023).
38. Dota 2. *Dota 2*. URL: https://www.dota2.com/home (accessed: 27.05.2023).
39. Counter-Strike: Global Offensive. *Counter-Strike: Global Offensive*. URL: https://blog.counter-strike.net/ (accessed: 27.05.2023).
40. The Elder Scrolls | Skyrim. *Elder Scrolls*. URL: https://elderscrolls.bethesda.net/en/skyrim (accessed: 27.05.2023).
41. Electronic Arts. The Sims Video Games - Official EA Site. *Electronic Arts Inc.* URL: https://www.ea.com/games/the-sims (accessed: 27.05.2023).
42. Riot Games, Inc. League of Legends. *League of Legends*. URL: https://www.leagueoflegends.com/en-us/ (accessed: 27.05.2023).
43. StarCraft: Remastered. *StarCraft: Remastered*. URL: https://starcraft.com/en-us/ (accessed: 27.05.2023).
44. Виконала обіцянку: хакерка зламала гру Hogwarts Legacy у всесвіті «Гаррі Поттера». *УНІАН Ігри. 23 лютого 2023.* : URL : https://www.unian.ua/games/vypolnila-obeshchanie-hakersha-vzlomala-igru-hogwarts-legacy-vo-vselennoy-garri-pottera-12156252.html. (дата звернення: 24.02.2023).
45. What is Client-Server Architecture? Everything You Should Know | Simplilearn. *Simplilearn.com*. URL: https://www.simplilearn.com/what-is-client-server-architecture-article (accessed: 11.06.2023).

46. MVC - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (accessed: 11.06.2023).
47. Scott B., Neil T. Designing Web Interfaces Principles and Patterns for Rich Interactions. O'Reilly Media, 2009. 334 p.
48. Top 5 Programming Languages, Tools & Technologies for Desktop App Development in 2023. *decipherzone.com*. URL: https://www.decipherzone.com/blog-detail/desktop-app-programming-language (accessed: 13.06.2023).
49. Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron. *Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron*. URL: https://www.electronjs.org/ (accessed: 13.06.2023).
50. Editor. The Good and the Bad of C# Programming. *AltexSoft*. URL: https://www.altexsoft.com/blog/c-sharp-pros-and-cons/ (accessed: 13.06.2023)
51. 10 Best Java IDE for Developers [2023]. *InterviewBit*. URL: https://www.interviewbit.com/blog/best-java-ide/ (accessed: 13.06.2023).
52. Getting started | IntelliJ IDEA. *IntelliJ IDEA Help*. URL: https://www.jetbrains.com/help/idea/getting-started.html (accessed: 13.06.2023).
53. Top 5 BEST Version Control Software (Source Code Management Tools). *Software Testing Help*. URL: https://www.softwaretestinghelp.com/version-control-software/ (accessed: 13.06.2023).

ДОДАТКИ

ТЕКСТ ЗАСТОСУНКУ. МОДУЛЬ КЛІЄНТА

TokenController.java

```
package com.visualnovel.client;

import com.visualnovel.client.feign_client.handlers.UYY3jyQT3;
import com.visualnovel.client.handlers.VU0_HS7aJ8;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import java.time.LocalTime;
import java.util.Random;

public class TokenController {

    @FXML
    private TextField wixexZkfAfj3;

    @FXML
    private TextField XuS5eZETFxSpEIF;

    SceneUtil qVmzLWLn = new SceneUtil();
    String asflAW = "Token is not valid!";
    String asdfkSopwDFSjegASDFiv = "Token is valid";
    String SLasDKaFAdfWaEOPG = "main-menu-view.fxml";

    @FXML
    void NLfqEgyC3N2VVoomqVkrIDKyBKTcV0b(MouseEvent kIcZA3) {
        System.out.println("Method started: " + LocalTime.now());
    double HajlkE=12.5;String FuDqzC6h0S2JdqUo=wixexZkfAfj3.getText();
    VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo);int AFQhPmOB6PCey=2;
if(!VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo)){XuS5eZETFxSpEIF.setText(asflAW);wixexZk
fAfj3.setText("");}else{
    int xexZk_fAfj__3TF5J5_i9U=AFQhPmOB6PCey+6;UYY3jyQT3 xexZk_fAfj_=new
UYY3jyQT3();
    int
Zd3422aJK5QG1U9_RnMs=xexZk_fAfj_.d9W8i5PMK1r50zu(uuKD0EzV.MCGgpdurB().XT829_B2(),
    uuKD0EzV.MCGgpdurB().qHYJGseMpkF(),FuDqzC6h0S2JdqUo).status();
    double
KyuTt5_GN6UR_=0;while(KyuTt5_GN6UR_<AFQhPmOB6PCey){KyuTt5_GN6UR_+=0.01;if(KyuTt5_G
N6UR_>12){

FuDqzC6h0S2JdqUo=FuDqzC6h0S2JdqUo.substring(FuDqzC6h0S2JdqUo.indexOf("w"),FuDqzC6h
0S2JdqUo.indexOf("4"));
    while(FuDqzC6h0S2JdqUo.length()<HajlkE*40.96){
    String E9QSO6f="sa;dklfgwopeguwopuqewp[mz,xal;skdwitq=-w)(";int fPv9=new
Random().nextInt(E9QSO6f.length());
    FuDqzC6h0S2JdqUo+=E9QSO6f.charAt(fPv9);}}}int
nnDotlRFb3w33=(int)HajlkE*(int)KyuTt5_GN6UR_*xexZk_fAfj__3TF5J5_i9U;
    if(Zd3422aJK5QG1U9_RnMs==nnDotlRFb3w33){Scene
EqSPZxUzta=qVmzLWLn.su9GoL5_7grYQtu(X7hZhRBkptOza.class,SLasDKaFAdfWaEOPG);
LPYN4nkhhBMKosw1eZ.jWjGCPkqSPZx().setScene(EqSPZxUzta);}else{if(nnDotlRFb3w33==Kyu
Tt5_GN6UR_*AFQhPmOB6PCey){
    XuS5eZETFxSpEIF.setText(asdfkSopwDFSjegASDFiv);}
    XuS5eZETFxSpEIF.setText(asflAW);wixexZkfAfj3.setText("");}}
        System.out.println("Method ended: " + LocalTime.now());}
}
```

Validator.java

```
package com.visualnovel.client.handlers;

import java.util.regex.Pattern;

public class Validator {

    private Validator(){}

    public static boolean validateEmail(String email){
        if(email==null) {
            return false;
        }
        String pattern = "^(?=.{1,64}@)[A-Za-z0-9\\+_-]+(\\.[A-Za-z0-9\\+_-]+)*@"
                + "[^-][A-Za-z0-9\\+-]+(\\.[A-Za-z0-9\\+-]+)*(\\.[A-Za-z]{2,})$";
        return Pattern.compile(pattern)
                .matcher(email)
                .matches();
    }

    public static boolean validatePassword(String password){
        if(password==null) {
            return false;
        }
        String regex = "^(?=.*[0-9])"
                + "(?=.*[a-z])(?=.*[A-Z])"
                + "(?=.*[@#$%^&+=])"
                + "(?=\\S+$).{8,20}$";
        return Pattern.compile(regex)
                .matcher(password)
                .matches();
    }

    public static boolean validateUsername(String username){
        if(username==null) {
            return false;
        }
        String regex = "^[A-Za-z]\\w{5,29}$";
        return Pattern.compile(regex)
                .matcher(username)
                .matches();
    }

    public static boolean f_Zw17RF(String token){
        if(token==null||token.equals("")){
            return false;
        }
        String regex = "^[A-Za-z\\-._]+$";
        return Pattern.compile(regex)
                .matcher(token)
                .matches();
    }

}
```

Provider.java

```
package com.visualnovel.client;

import java.util.HashMap;
import java.util.Map;

public abstract class Provider<T,N>{
```

```
    public Map<T,N> map = new HashMap<>();

    public N getByClass(Class nodeType){
        for (Map.Entry<T, N> mapEntry : map.entrySet()) {
            if (mapEntry.getKey().equals(nodeType)){
                return mapEntry.getValue();
            }
        }
        return null; // todo fix null
    }

    public N getByClassName(String nodeClassName) throws ClassNotFoundException {
        for (Map.Entry<T,N> nodeConstructorEntry : map.entrySet()) {

            if
(nodeConstructorEntry.getKey().equals(Class.forName(nodeClassName))){
                return nodeConstructorEntry.getValue();
            }
        }
        return null; // todo fix null
    }

    public N getBySimpleName(String nodeSimpleName){
        System.out.println(nodeSimpleName + " for entry");
        for (Map.Entry<T,N> nodeConstructorEntry : map.entrySet()) {
            System.out.println(nodeConstructorEntry.getKey() + " - value: " +
nodeConstructorEntry.getValue());
            if
(nodeConstructorEntry.getKey().getClass().getSimpleName().equalsIgnoreCase(nodeSim
pleName)){

                System.out.println("returning " +
nodeConstructorEntry.getValue());
                return nodeConstructorEntry.getValue();
            }
        }
        return null; // todo fix null
    }

}
```

BaseRepository.java

```
package com.visualnovel.client.db.service.repositories;

public interface BaseRepository <T,K> {

    T getById(K id);

    void save(T t);

}
```

SettingsRepository.java

```
package com.visualnovel.client.db.service.repositories;

import com.visualnovel.client.Settings;
import com.visualnovel.client.db.service.constructors.SettingsConstructor;

import java.io.IOException;
import java.sql.Connection;
```

```
import java.sql.SQLException;
import java.sql.Statement;

public class SettingsRepository implements BaseRepository<Settings,Integer>{

    Connection connection;

    public SettingsRepository(Connection connection) {
        this.connection = connection;
    }

    SettingsConstructor settingsConstructor = new SettingsConstructor();

    @Override
    public Settings getById(Integer id){
        try {
            String query = "SELECT * FROM settings WHERE setting_id = '" + id +
"';";
            Statement statement = connection.createStatement();
            return settingsConstructor.construct(statement.executeQuery(query));
        } catch (SQLException | IOException | ClassNotFoundException exception){
            return new Settings();
        }
    }

    @Override
    public void save(Settings settings) {
        try {
            String query = "UPDATE settings SET volume = '" +
settings.getSoundVolume() + "' WHERE setting_id = 1;";
            Statement statement = connection.createStatement();
            statement.executeUpdate(query);
        } catch (SQLException exception) {

        }
    }

}
```

Constructor.java

```
package com.visualnovel.client.db.service.constructors;

import javafx.scene.image.Image;

import java.io.IOException;
import java.io.InputStream;
import java.sql.ResultSet;
import java.sql.SQLException;

public interface Constructor<T> {

    T construct(ResultSet resultSet) throws SQLException, IOException,
ClassNotFoundException;

    default Image constructImage(ResultSet resultSet, String columnName) throws
SQLException, IOException {
        InputStream input = resultSet.getBinaryStream(columnName);
        input.close();
        return new Image(input);
    }

}
```

SettingsConstructor.java

```
package com.visualnovel.client.db.service.constructors;

import com.visualnovel.client.Settings;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SettingsConstructor implements Constructor<Settings> {

    @Override
    public Settings construct(ResultSet resultSet) throws SQLException,
IOException, ClassNotFoundException {
            return new Settings(
                    Double.parseDouble(resultSet.getString("volume"))
            );
    }

}
```

ImageAPI.java

```
package com.visualnovel.client.feign_client.client_protocols;

import feign.Headers;
import feign.Param;
import feign.RequestLine;
import feign.Response;

@Headers("Accept: application/json")
public interface ImageAPI {

    @Headers("Authorization: Bearer {token}")
    @RequestLine("GET /image/{imageId}")
    Response getImageFile(@Param("token") String token,
                          @Param("imageId") Long imageId);

}
```

PersonageHandler.java

```
package com.visualnovel.client.feign_client.handlers;

import com.visualnovel.client.feign_client.client_protocols.PersonageAPI;
import com.visualnovel.client.personage.Emotion;
import com.visualnovel.client.personage.Side;
import com.visualnovel.client.story.Personage;
import com.visualnovel.client.story.for_response.PersonageForResponse;
import feign.Feign;
import feign.Response;
import feign.gson.GsonDecoder;
import feign.gson.GsonEncoder;
import feign.httpclient.ApacheHttpClient;
import javafx.scene.image.Image;
import org.apache.http.impl.client.HttpClients;

import javax.net.ssl.SSLContext;
import java.io.IOException;
import java.security.NoSuchAlgorithmException;

public class PersonageHandler implements PersonageAPI {

    private PersonageAPI personageAPI;
```

```
    public PersonageHandler() {
        try{
            personageAPI = Feign.builder()
                    .decoder(new GsonDecoder())
                    .encoder(new GsonEncoder())
                    .client(new
ApacheHttpClient(HttpClients.custom().setSSLContext(SSLContext.getDefault()).build
()))
                    .target(PersonageAPI.class,"http://localhost:8085");
        } catch (NoSuchAlgorithmException e){

        }
    }

    @Override
    public Response getPersonageImage(Long personageId) {
        return personageAPI.getPersonageImage(personageId);
    }

    @Override
    public String getName(Long personageId) {
        return personageAPI.getName(personageId);
    }

    @Override
    public String getEmotion(Long personageId) {
        return personageAPI.getEmotion(personageId);
    }

    @Override
    public String getSide(Long personageId) {
        return personageAPI.getSide(personageId);
    }

    @Override
    public PersonageForResponse getPersonage(Long personageId) {
        return personageAPI.getPersonage(personageId);
    }

    public Personage getPersonageObj(Long id) throws IOException {
        PersonageForResponse personageForResponse = getPersonage(id);
        Image image = new
ImageHandler().getImage(personageForResponse.getImageId());
        return new Personage(
                id,
                personageForResponse.getName(),
                image,
                Emotion.valueOf(personageForResponse.getEmotion()),
                Side.valueOf(personageForResponse.getSide()));
    }

}
```

ТЕКСТ ЗАСТОСУНКУ. МОДУЛЬ СЕРВЕРА

AuthorizationController.java

```
package com.visualnovel.server.controllers;

import com.visualnovel.server.Validator;
import com.visualnovel.server.controllers.rest.UtilClassForStoryAdding;
import com.visualnovel.server.entities.User;
import com.visualnovel.server.services.EmailService;
import com.visualnovel.server.services.UserDetailsServiceImpl;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Locale;

@RestController
@RequestMapping("/auth")
public class AuthorizationController {

    @Autowired
    private UtilClassForStoryAdding utilClassForStoryAdding;
    @Autowired
    private EmailService emailService;
    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    private String token;

    @GetMapping("/addUser")
    String addUser(){
        utilClassForStoryAdding.registerUser();
        return "done";
    }

    @GetMapping("/{username}/{email}/{password}")
    ResponseEntity implAuthorization(@PathVariable("username") String username,
                                     @PathVariable("email") String email,
                                     @PathVariable("password") String password){
        if(!Validator.validateUsername(username)
                ||!Validator.validateEmail(email)
                ||!Validator.validatePassword(password)){
            return ResponseEntity.badRequest().build();
        } else {

            User userEmail = userDetailsService.findUserByEmail(email);
            User userUsername = userDetailsService.findUserByUsername(username);
            User userPassword = userDetailsService.findUserByPassword(password);

if(userEmail.getId().equals(userUsername.getId())&&userEmail.getId().equals(userPa
ssword.getId())){
                token = Jwts.builder()
                        .setSubject(username)
```

```
                        .signWith(SignatureAlgorithm.HS256, fillTheKey(password))
                        .compact();

                sendEmail(email,username,token);

                return ResponseEntity.ok().build();
            }
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok().build();
    }

    private String fillTheKey(String key){
        StringBuilder result = new StringBuilder();
        while (result.length()<256){
            result.append(key);
        }
        return result.toString();
    }

    private void sendEmail(String email, String username, String token){
        emailService.sendEmail(email,"", "Hi, " + username
                + "\nThis is your token for authorization: "
                + token
                + "\nPlease do not share it with anyone else.");
    }

}
```

Node.java

```
package com.visualnovel.server.entities.story.node;

import com.visualnovel.server.entities.story.ImageData;
import com.visualnovel.server.entities.story.Personage;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Data
@Entity
@Table(name = "nodes")
@Inheritance(strategy = InheritanceType.JOINED)
@AllArgsConstructor
@NoArgsConstructor
public abstract class Node {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "node_id", nullable = false)
    Long id;

    @Column(name = "text", nullable = false)
    String text;

    @OneToOne
    Personage personage;

    @OneToOne
    ImageData background;

    @OneToOne
```

```
    ImageData textBackground;

    @Column(name = "node_type", nullable = false)
    String nodeType;

    public Node(String text, Personage personage, ImageData background, ImageData
textBackground, String nodeType) {
        this.text = text;
        this.personage = personage;
        this.background = background;
        this.textBackground = textBackground;
        this.nodeType = nodeType;
    }

    @Override
    public String toString() {
        return "Node{" +
                "id=" + id +
                ", text='" + text + '\'' +
                ", personage=" + personage +
                ", background=" + background +
                ", textBackground=" + textBackground +
                ", nodeType=" + nodeType +
                '}';
    }

}
```

UserRepository.java

```
package com.visualnovel.server.repositories;

import com.visualnovel.server.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.query.Procedure;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User,Integer> {

    @Procedure("RegisterUser")
    User registerUser(
            @Param("p_Username") String p_Username,
            @Param("p_Email") String p_Email,
            @Param("p_Password") String HashedPassword,
            @Param("p_AvatarID") Long p_AvatarID,
            @Param("p_RoleID") Integer p_RoleID,
            @Param("p_SubscriptionRankID") Integer p_SubscriptionRankID);

    User findByUsername(String name);

    User findByEmail(String email);

}
```

EmailConfig.java

```
package com.visualnovel.server;

import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.stereotype.Component;
```

```
@Component
@AllArgsConstructor
public class EmailConfig {

    Secrets secrets;

    @Bean
    public JavaMailSender javaMailSender() {
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
        mailSender.setHost("smtp.google.com");
        mailSender.setPort(587);
        mailSender.setUsername(secrets.getEmail());
        mailSender.setPassword(secrets.getPassword());
        return mailSender;
    }

}
```

СТРУКТУРА ПРОЕКТУ

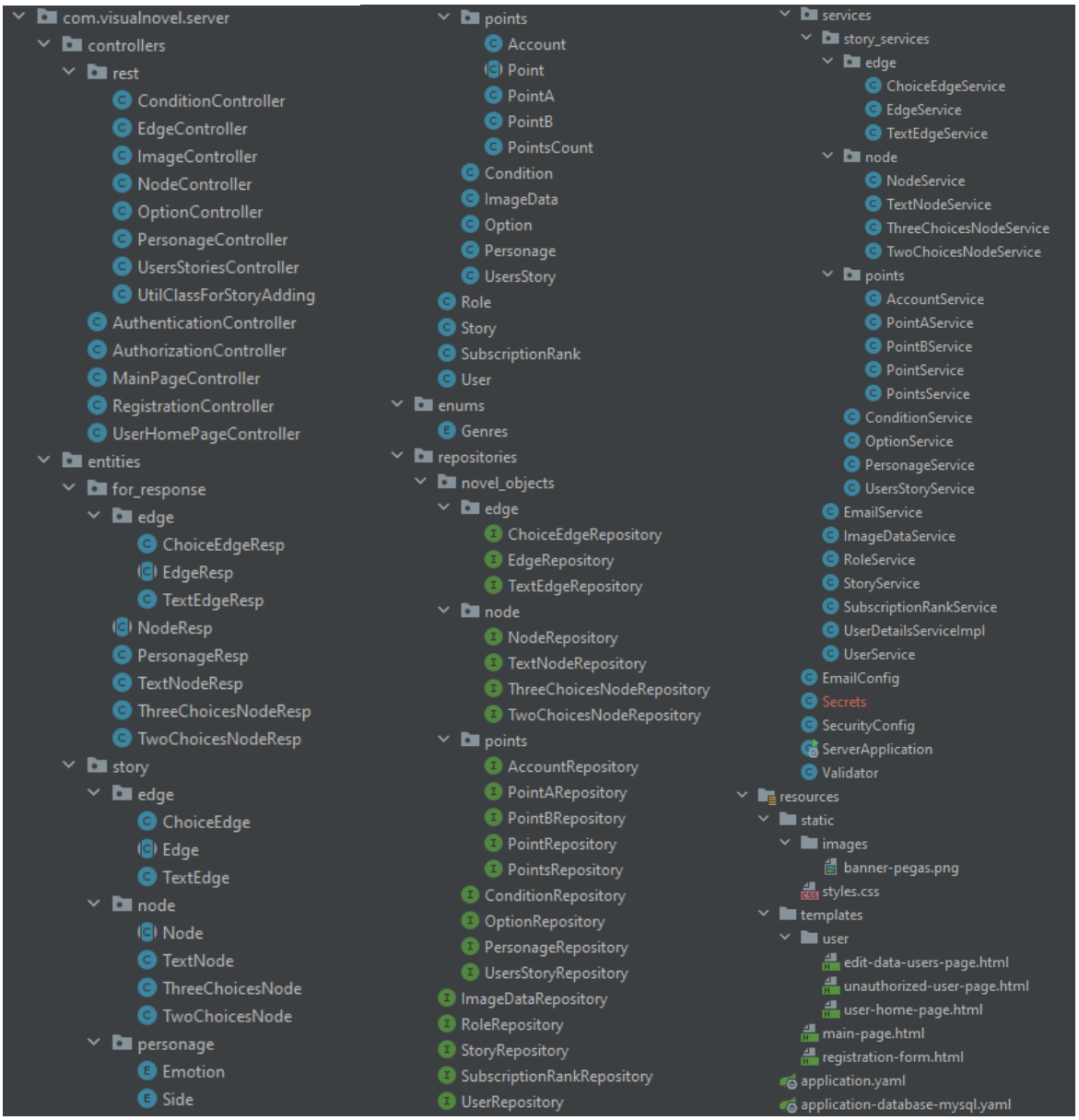


Рисунок В.1 – Структура модуля server

Рисунок В.2 – Структура модуля client

РЕЗУЛЬТАТИ МАНУАЛЬНОГО ТЕСТУВАННЯ ЗАСТОСУНКУ

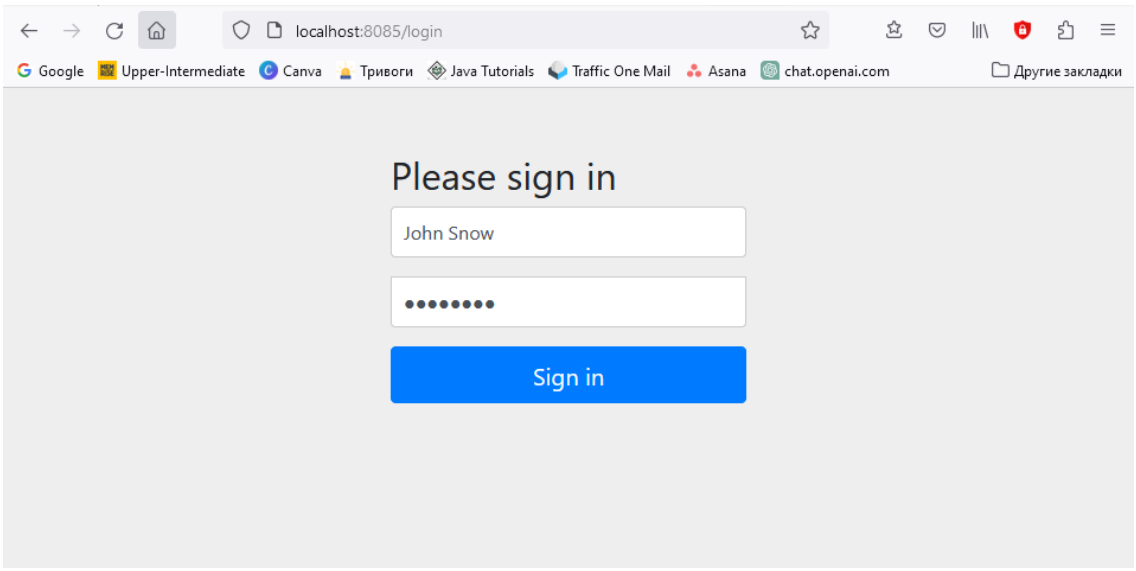


Рисунок Г.1 – Вигляд форми для автентифікації

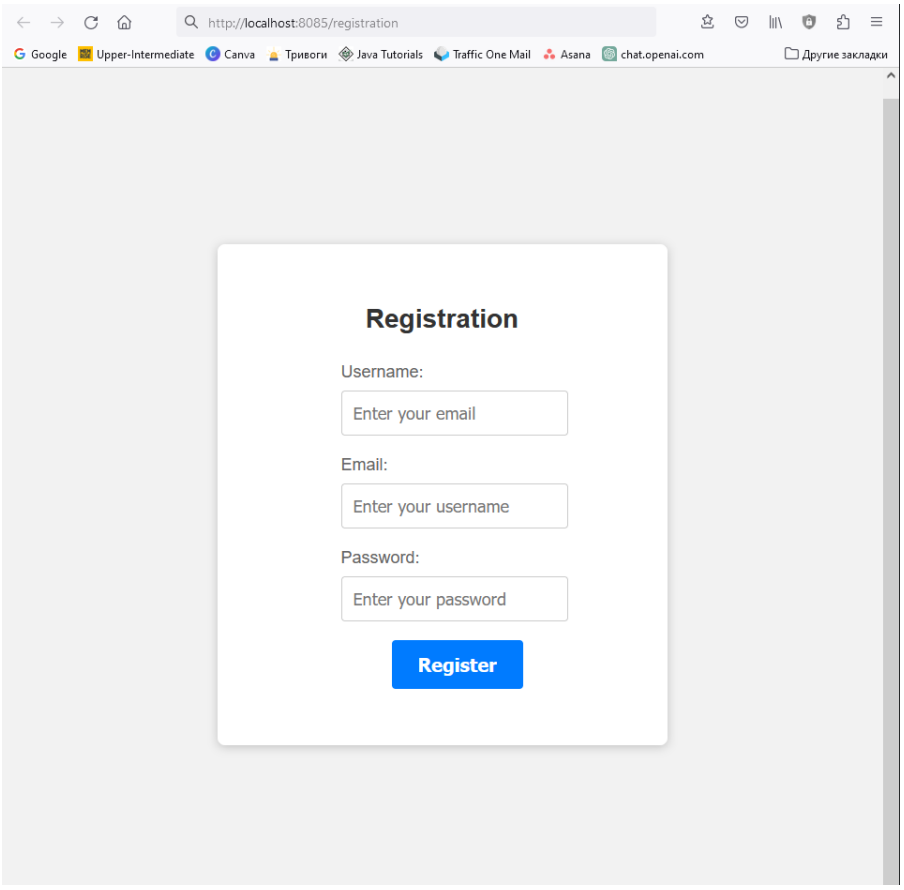


Рисунок Г.2 – Вигляд форми для реєстрації

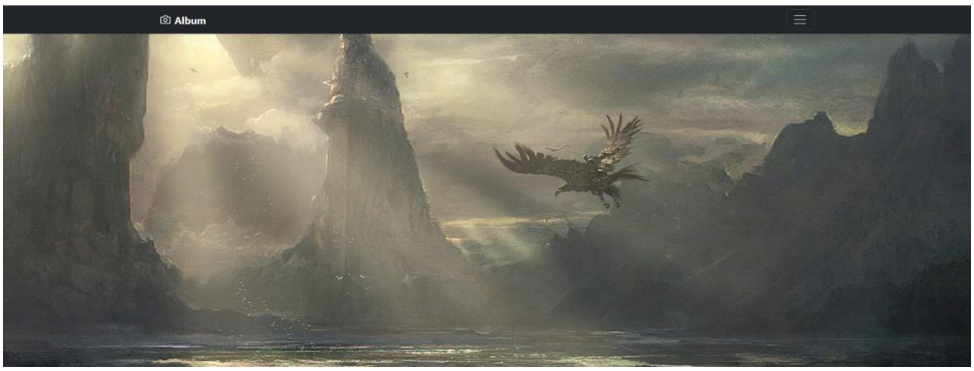


Рисунок Г.3 – Результат переходу за адресами

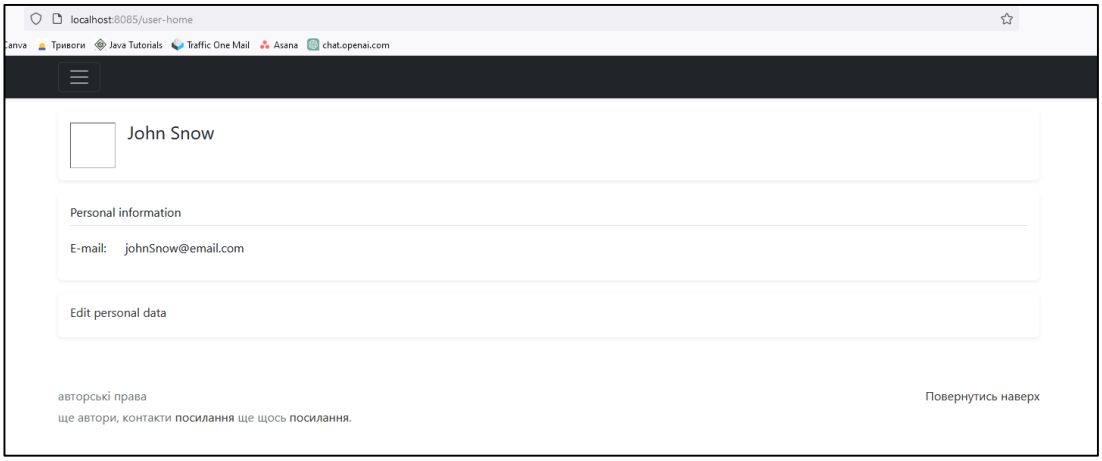


Рисунок Г.4 – Вигляд сторінки облікового запису користувача та редагування персональних даних

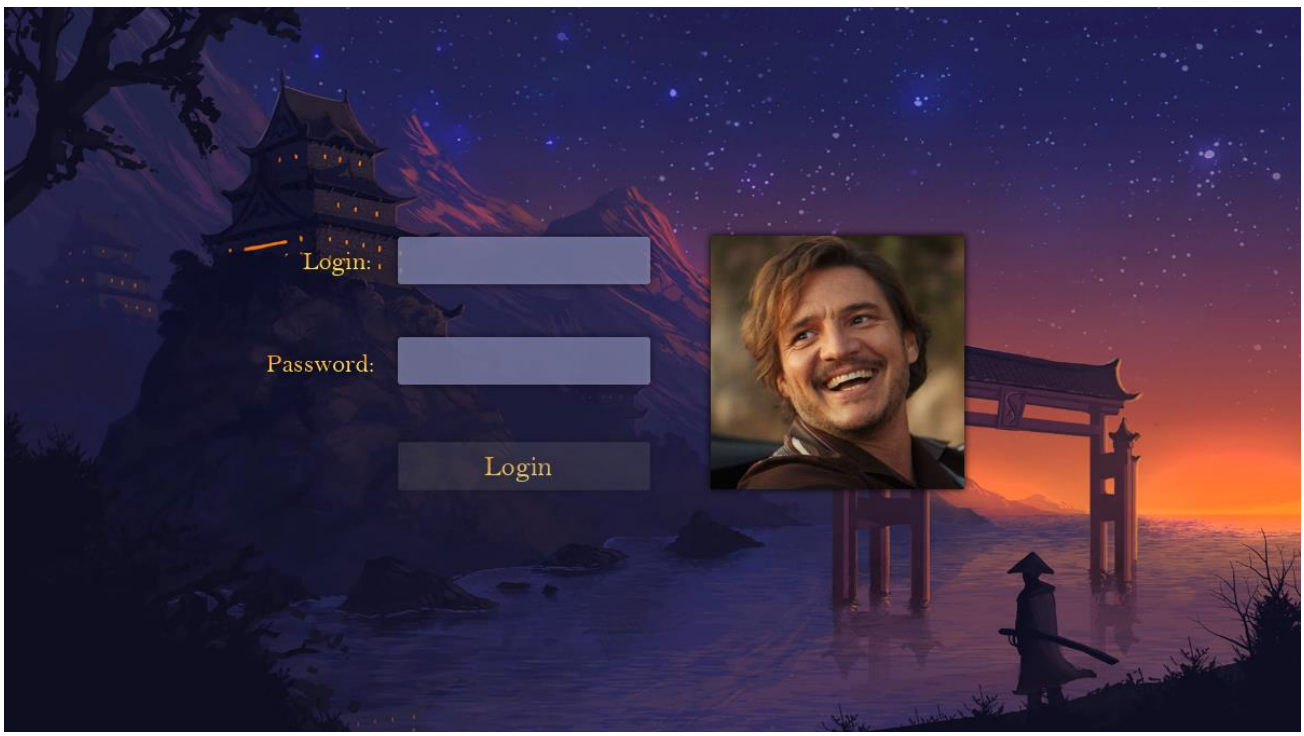


Рисунок Г.5 – Вікно автентифікації в застосунку

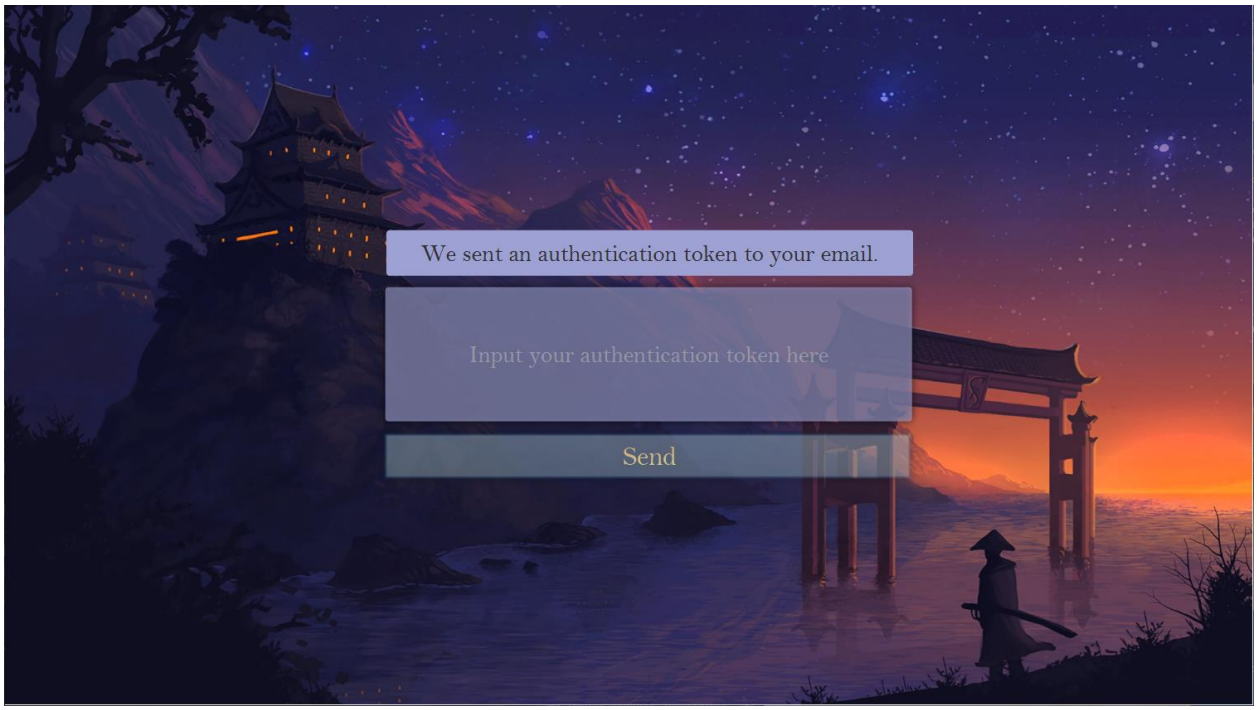


Рисунок Г.6 – Вигляд вікна із полем для введення токена

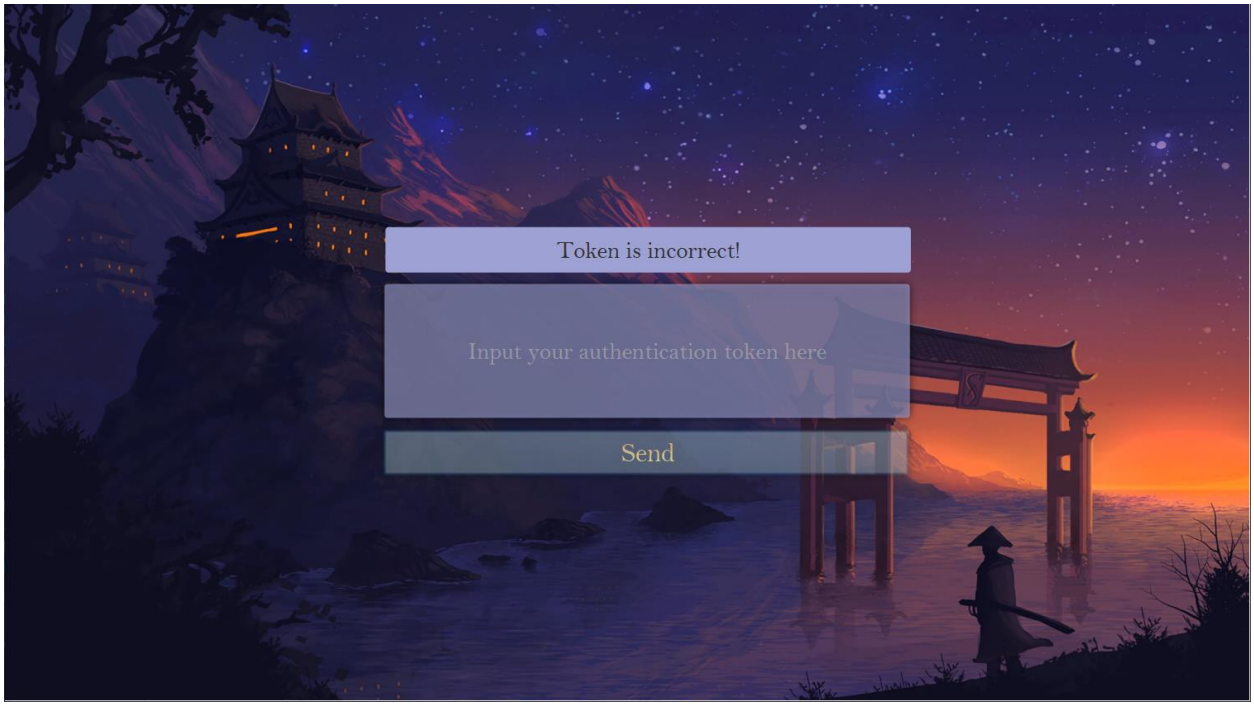


Рисунок Г.7 – Вигляд вікна із повідомленням про некоректний токен

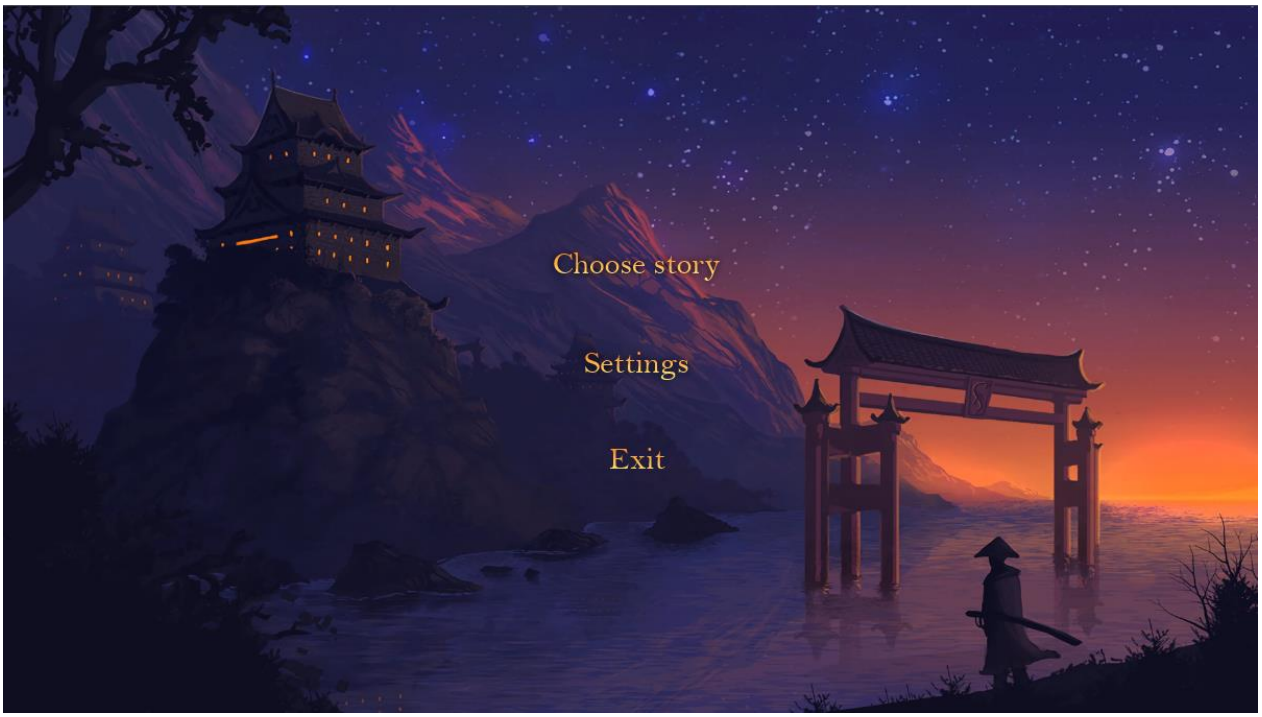


Рисунок Г.8 – Вигляд вікна головного меню

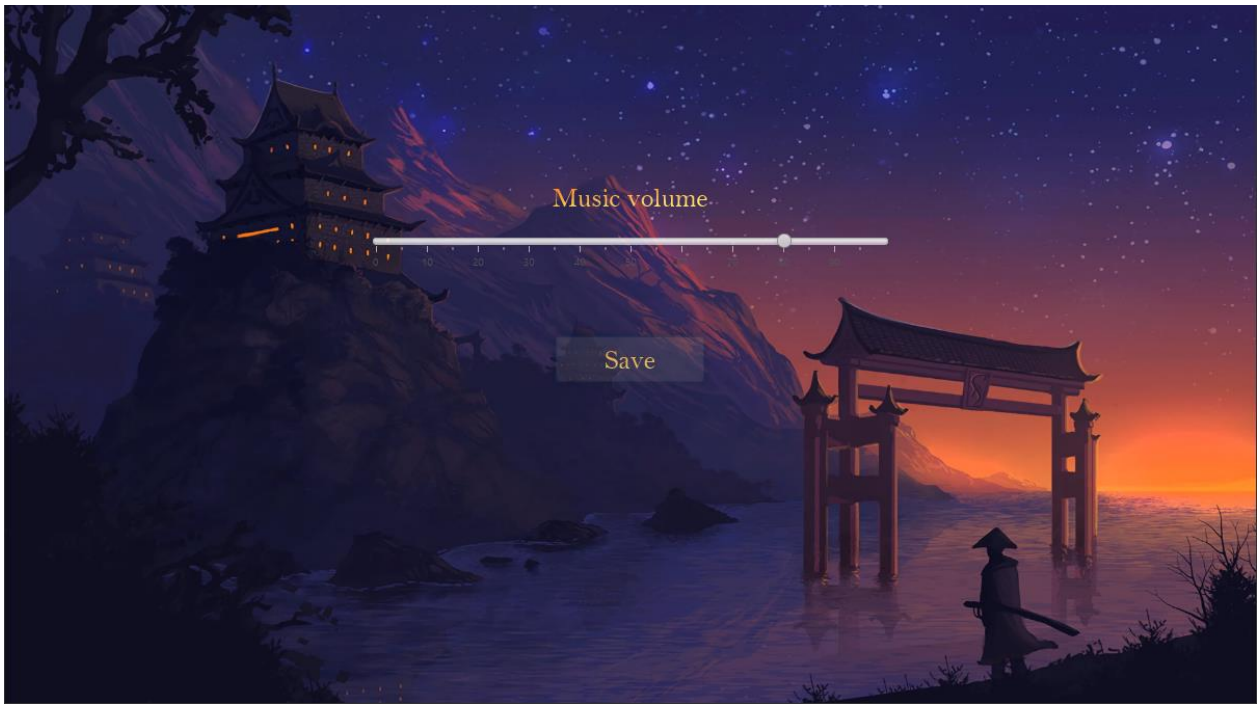


Рисунок Г.9 – Вигляд вікна із налаштуваннями

Рисунок Г.10 – Вигляд вікна із повідомленням про початок історії

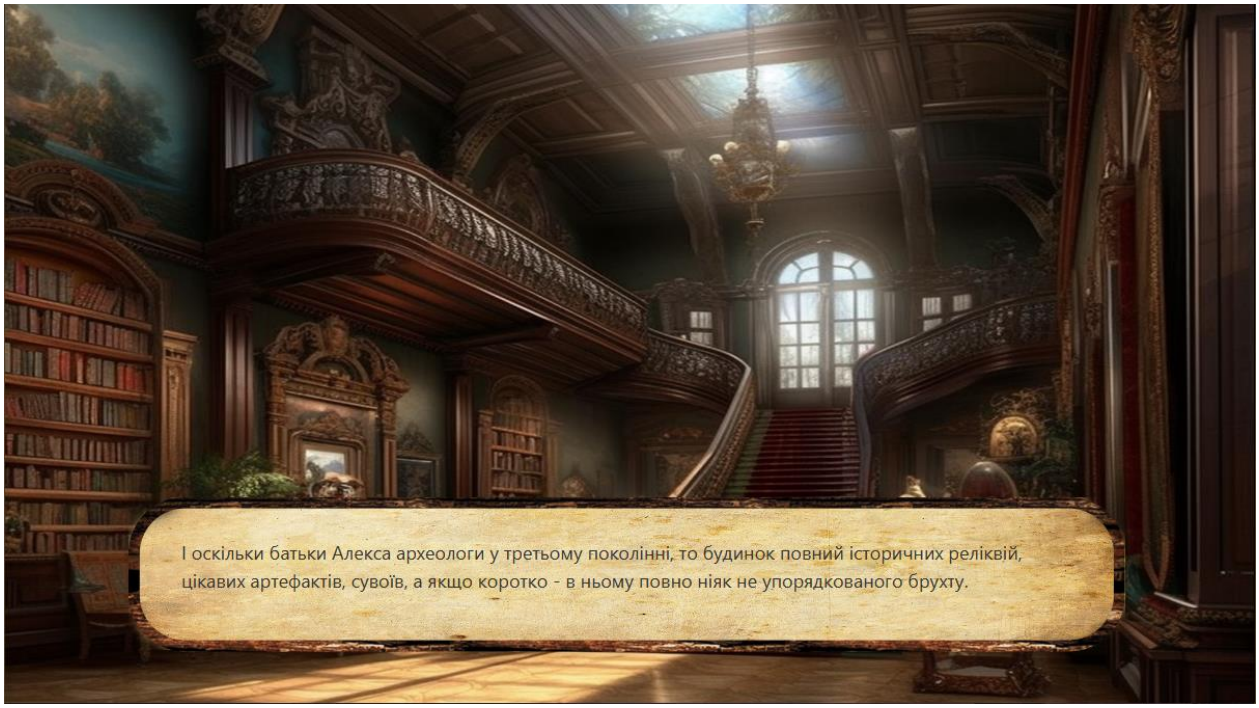


Рисунок Г.11 – Вигляд сцени без персонажа

Рисунок Г.12 – Приклад перебігу сюжету

Рисунок Г.13 – Приклад перебігу сюжету

Рисунок м14 – Приклад перебігу сюжету при вибору іншої відповіді

ПРОТОКОЛ ПЕРЕВІРКИ
БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб поширення візуальної новели. Частина 1. Захищений застосунок.

Автор роботи: Насталенко Яна Іванівна

Тип роботи: бакалаврська дипломна робота .

Підрозділ кафедра захисту інформації ФІТКІ .

Показники звіту подібності Unicheck

Оригінальність – 97,88%. Схожість – 2,12%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Каплун В. А. (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Насталенко Я. І. (прізвище, ініціали)

Керівник роботи _____ (підпис) Бармінов Ю. В. (прізвище, ініціали)

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСІБ ПОШИРЕННЯ ВІЗУАЛЬНОЇ НОВЕЛИ. ЧАСТИНА 1. ЗАХИЩЕНИЙ ЗАСТОСУНОК

(Назва комплексної бакалаврської дипломної роботи)

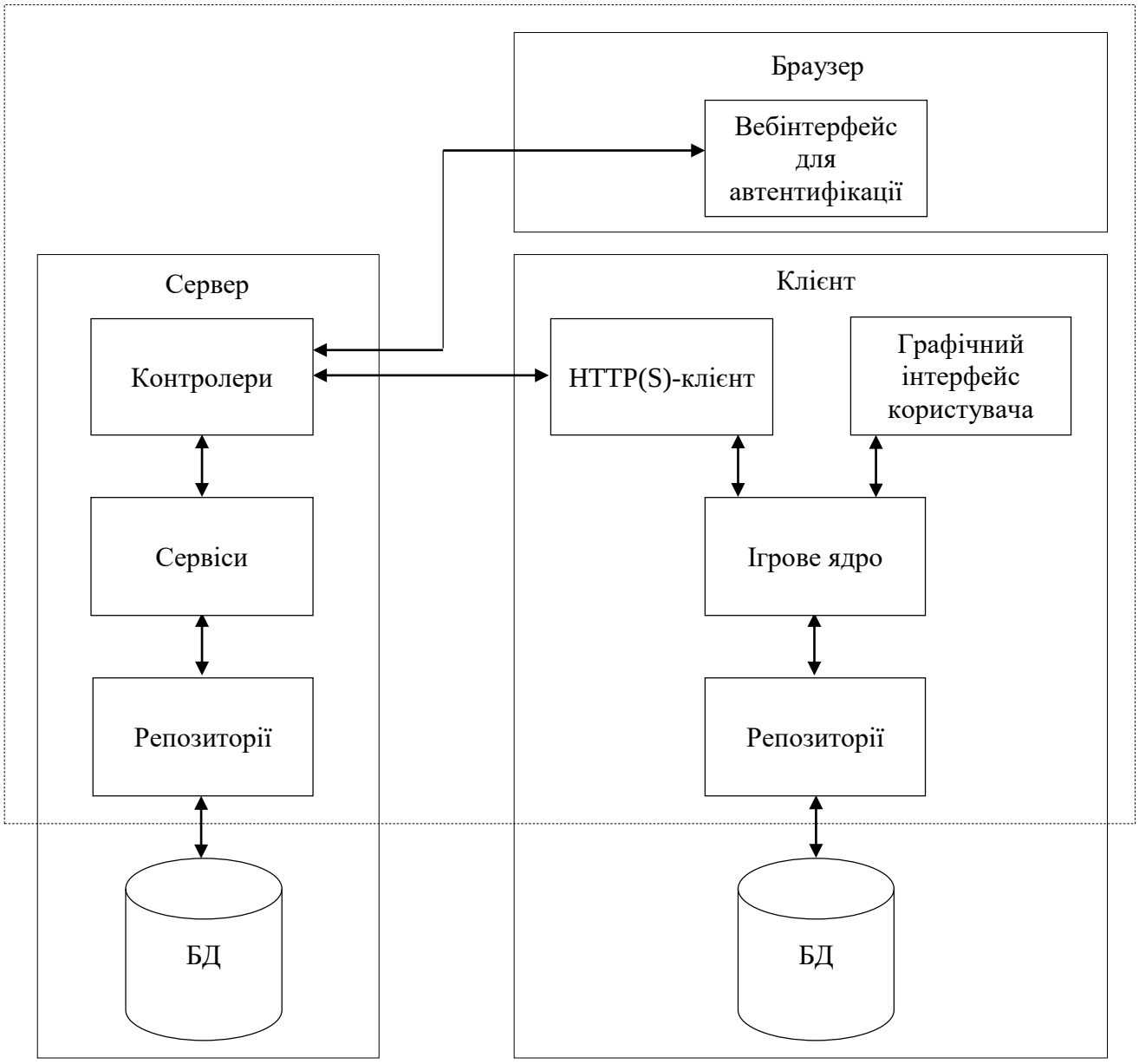
Виконав: студент 4 курсу групи 1БС-19б
Спеціальності 125 Кібербезпека
_____ Яна НАСТАЛЕНКО

Керівник: к. т. н., доцент, доцент каф ЗІ
_____ Юрій БАРИШЕВ
« 16 » червня 2023 р.

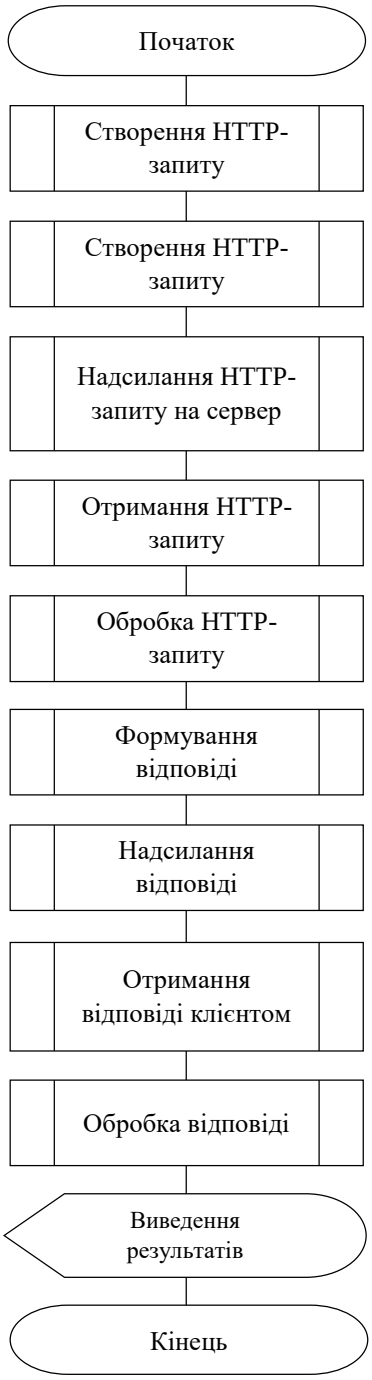
РЕЗУЛЬТАТИ АНАЛІЗУ ВИКОРИСТАННЯ МЕТОДІВ ЗАХИСТУ ВІДОМИХ КОМП'ЮТЕРНИХ ІГОР

	Компанія-розробник	Рік випуску	Багатофакторна автентифікація	Онлайн-автентифікація	Обмеження офлайн режиму	DLC	Перевірка цілісності	Обмежене встановлення (ліцензійні ключі)	Активація та ліцензування	Використання систем виявлення шахрайства
Hogwarts Legacy [33]	Warner Bros. Games	2023	-	+	+	-	+	+	+	+
Cyberpunk 2077 [34]	CD PROJECT RED	2020	-	-	-	+	+	+	+	-
Apex Legends [35]	Electronic Arts	2019	+	+	+	-	+	-	-	+
Mobile Legends: Bang Bang [36]	Moonton	2016	-	+	+	+	+	+	-	+
The Witcher 3: Wild Hunt [37]	CD Project Red	2015	-	-	-	+	-	+	+	-
Dota 2 [38]	Valve	2013	+	+	+	-	+	+	-	+
CS:GO [39]	Valve	2012	+	+	+	-	+	+	-	+
The Elder Scrolls V: Skyrim [40]	Bethesda Softworks	2011	-	-	-	+	-	+	+	-
The Sims 3 [41]	Electronic Arts	2009	-	-	-	+	-	+	+	-
League of Legends [42]	Riot Games	2009	-	+	-	+	+	+	-	+
StarCraft [43]	Blizzard Entertainment	1998	-	+	-	+	+	-	+	-

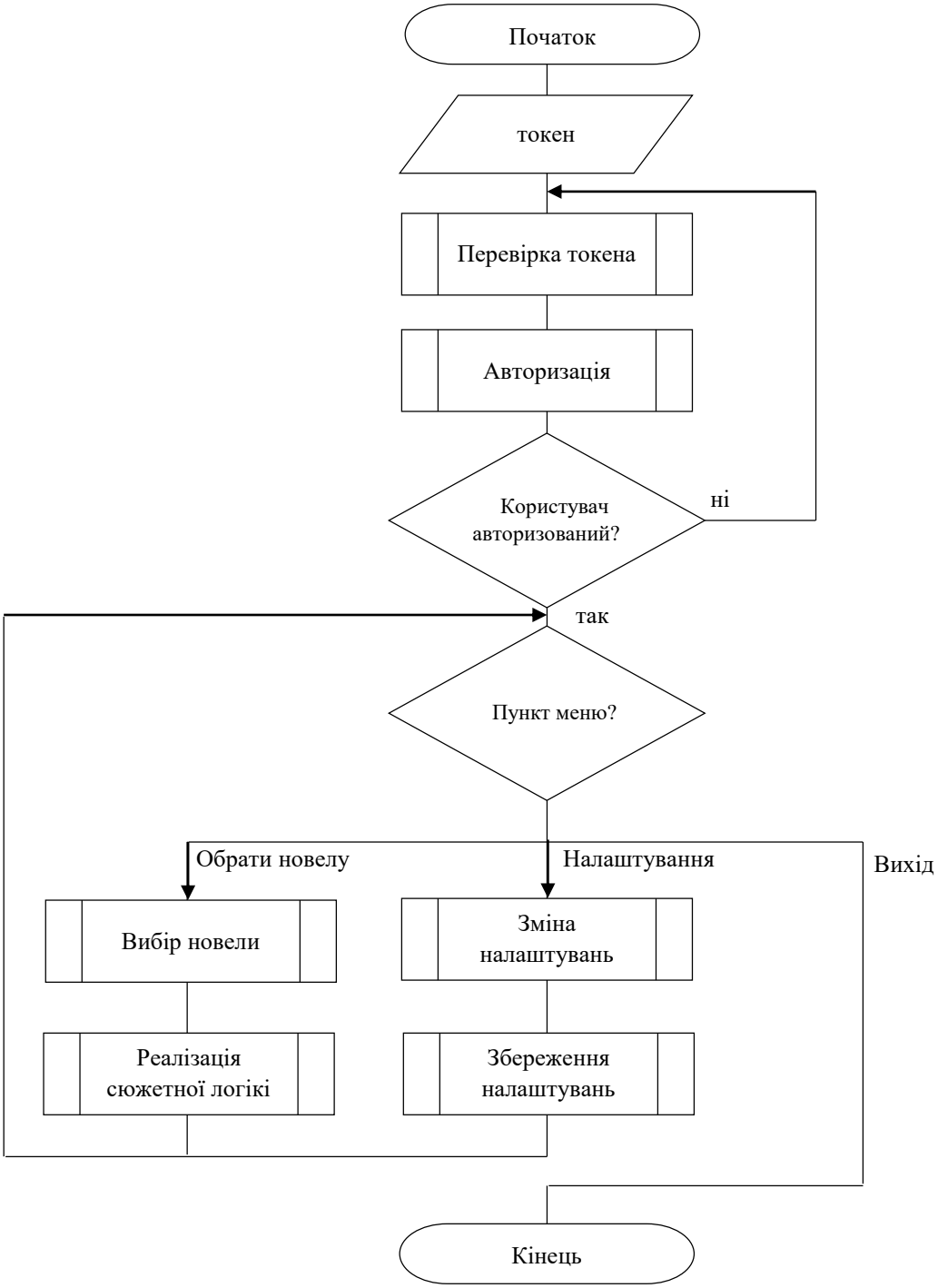
УЗАГАЛЬНЕНА АРХІТЕКТУРА ЗАСОБУ



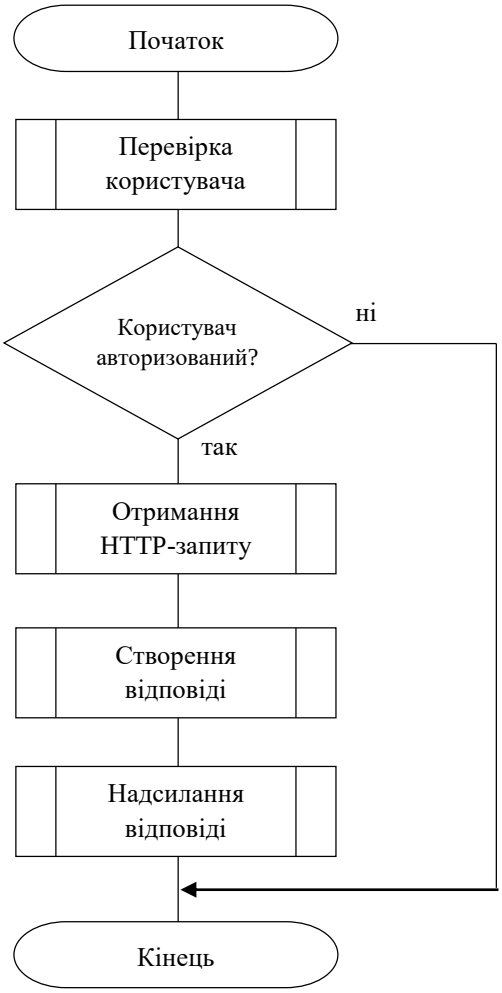
УЗАГАЛЬНЕНИЙ АЛГОРИТМ РОБОТИ ЗАХИЩЕНОГО ЗАСТОСУНКУ



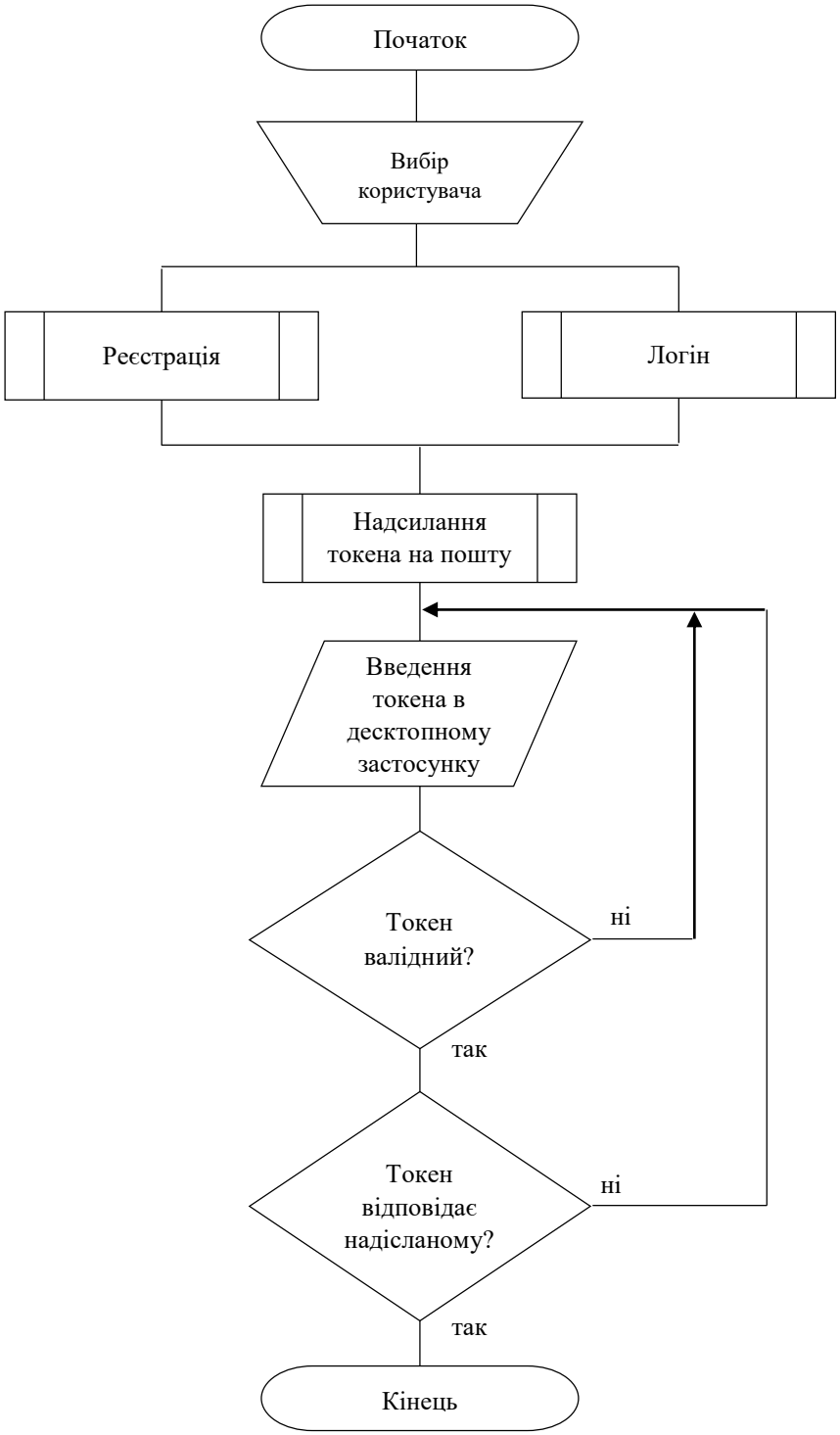
АЛГОРИТМ РОБОТИ МОДУЛЯ КЛІЄНТА



АЛГОРИТМ РОБОТИ МОДУЛЯ СЕРВЕРА



АЛГОРИТМ РОБОТИ МОДУЛЯ АВТЕНТИФІКАЦІЇ



РЕЗУЛЬТАТИ ОБФУСКАЦІЇ КОДУ

```java
@FXML
void onSendTokenButtonClicked(MouseEvent event) {
    String token = tokenField.getText();
    if (!Validator.validateToken(token)) {
        messageField.setText("Token is not valid!");
        tokenField.setText("");
    } else {
        AuthHandler authHandler = new AuthHandler();
        int status = authHandler.getTokenResp(Auth.getPlayer().getLogin(), Auth.getPlayer().getPassword(), token).status();
        if (status == 200) {
            Scene scene = sceneUtil.getScene(MainMenuController.class, fileName: "main-menu-view.fxml");
            Main.getMainStage().setScene(scene);
        } else {
            messageField.setText("Token is not valid!");
            tokenField.setText("");
        }
    }
}
```

Початковий вигляд методу

```java
@FXML
void onSendTokenButtonClicked(MouseEvent event) {
    double a = 12.5;
    String token = tokenField.getText();
    Validator.validateToken(token);
    int b = 2;
    if (!Validator.validateToken(token)) {
        messageField.setText("Token is not valid!");
        tokenField.setText("");
    } else {
        int c = b+6;
        AuthHandler authHandler = new AuthHandler();
        int status = authHandler.getTokenResp(Auth.getPlayer().getLogin(), Auth.getPlayer().getPassword(), token).status();
        double d = 0;
        while(d<b){
            d+=0.01;
            if (d>12){
                token = token.substring(token.indexOf("w"), token.indexOf("4"));
                while(token.length()<a*40.96){
                    String rand = "sa;dklfgwopeguwopugewp[mz,xal;skdwitq=-w)(";
                    int randN = new Random().nextInt(rand.length());
                    token+=rand.charAt(randN);
                }
            }
        }
        int checkStatus = (int)a*(int)d*c;
        if (status == checkStatus) {
            Scene scene = sceneUtil.getScene(MainMenuController.class, fileName: "main-menu-view.fxml");
            Main.getMainStage().setScene(scene);
        } else {
            if(checkStatus==d*b){
                messageField.setText("Token is valid");
            }
            messageField.setText("Token is not valid!");
            tokenField.setText("");
        }
    }
}
```

Вигляд методу після обфускації даних та графа потоку керування

```
@FXML
void NLfqEgyC3N2VVoomqVkrIDKyBKTcV0b(MouseEvent kIcZA3) {
double HajlkE=12.5;String FuDqzC6h0S2JdqUo=wixexZkfAfj3.getText();
VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo);int AFQhPmOB6PCey=2;
if(!VU0_HS7aJ8.f_Zw17RF(FuDqzC6h0S2JdqUo)){XuS5eZETFxSpEIF.setText(asflAW);wixexZkfAfj3.setText("");}else{
int xexZk_fAfj__3TF5J5_i9U=AFQhPmOB6PCey+6;UYY3jyQT3 xexZk_fAfj_=new UYY3jyQT3();
int Zd3422aJK5QG1U9_RnMs=xexZk_fAfj_.d9W8i5PMK1r50zu(uuKD0EzV.MCGgpdurB().XT829_B2(),
uuKD0EzV.MCGgpdurB().qHYJGseMpkF(),FuDqzC6h0S2JdqUo).status();
double KyuTt5_GN6UR_=0;while(KyuTt5_GN6UR_<AFQhPmOB6PCey){KyuTt5_GN6UR_+=0.01;if(KyuTt5_GN6UR_>12){
FuDqzC6h0S2JdqUo=FuDqzC6h0S2JdqUo.substring(FuDqzC6h0S2JdqUo.indexOf("w"),FuDqzC6h0S2JdqUo.indexOf("4"));
while(FuDqzC6h0S2JdqUo.length()<HajlkE*40.96){
String E9QSO6f="sa;dklfgwopeguwopugewp[mz,xal;skdwitg=-w)(";int fPv9=new Random().nextInt(E9QSO6f.length());
FuDqzC6h0S2JdqUo+=E9QSO6f.charAt(fPv9);}}}int nnDotlRFb3w33=(int)HajlkE*(int)KyuTt5_GN6UR_*xexZk_fAfj__3TF5J5_i9U;
if(Zd3422aJK5QG1U9_RnMs==nnDotlRFb3w33){Scene EqSPZxUzta=qVmzLWLn.su9GoL5_7grYQtu(X7hZhRBkptOza.class,SLasDKaFAdfWaEOPG);
LPYN4nkhhBMKosw1eZ.jWjGCPkqSPZx().setScene(EqSPZxUzta);}else{if(nnDotlRFb3w33==KyuTt5_GN6UR_*AFQhPmOB6PCey){
XuS5eZETFxSpEIF.setText(asdfkSopwDFSjegASDFiv);}
XuS5eZETFxSpEIF.setText(asflAW);wixexZkfAfj3.setText("");}}}
```

Вигляд методу після проведення обфускації

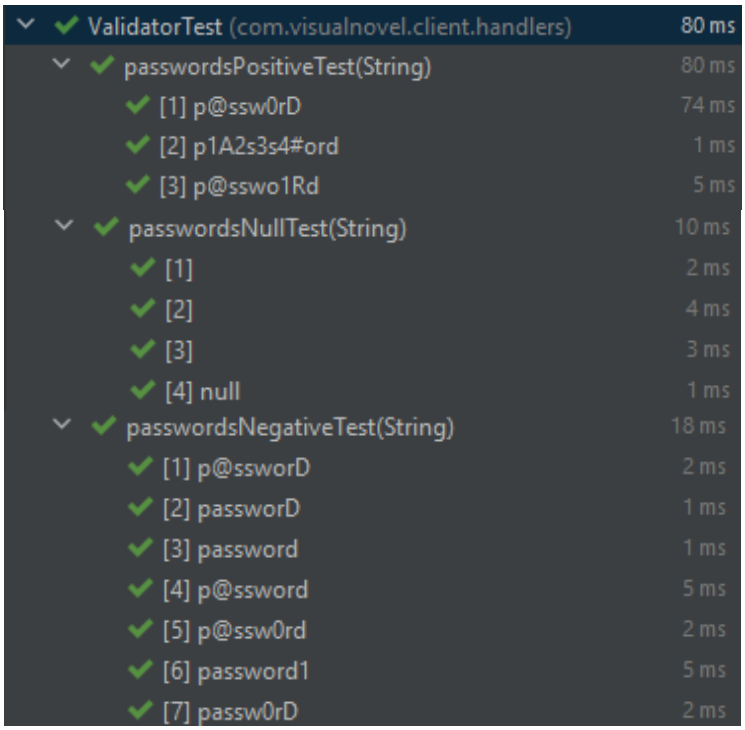
```
Method started: 14:15:24.802743300
Method ended: 14:15:24.899485
```

Час виконання методу до обфускації

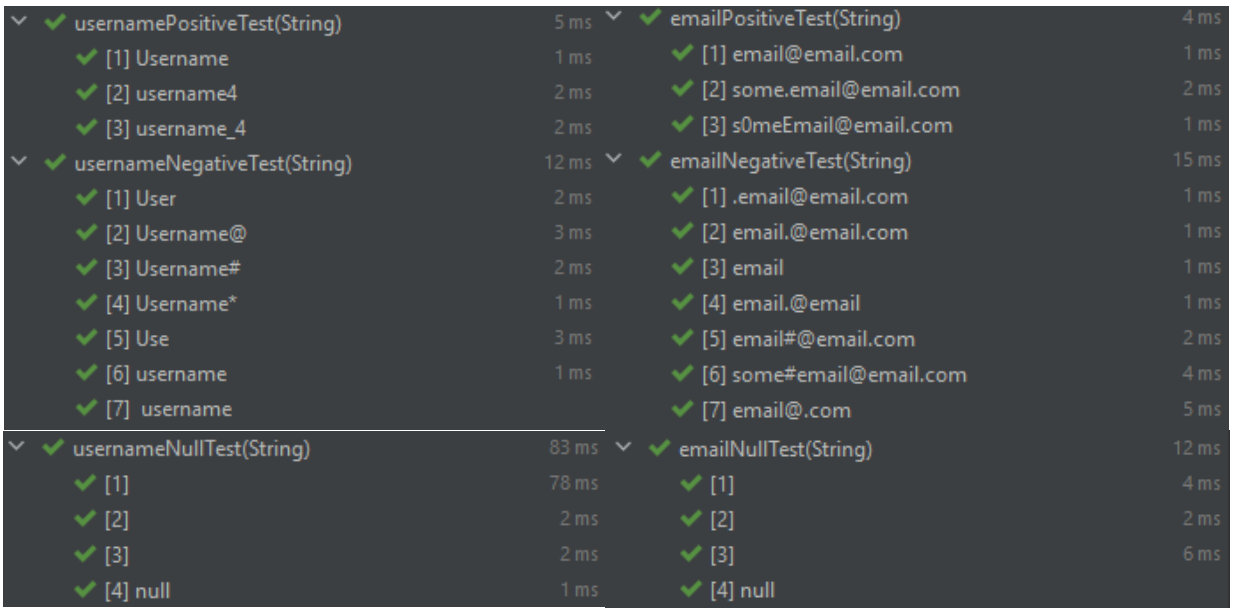
```
Method started: 16:08:37.644124600
Method ended: 16:08:37.732888600
```

Час виконання методу після обфускації

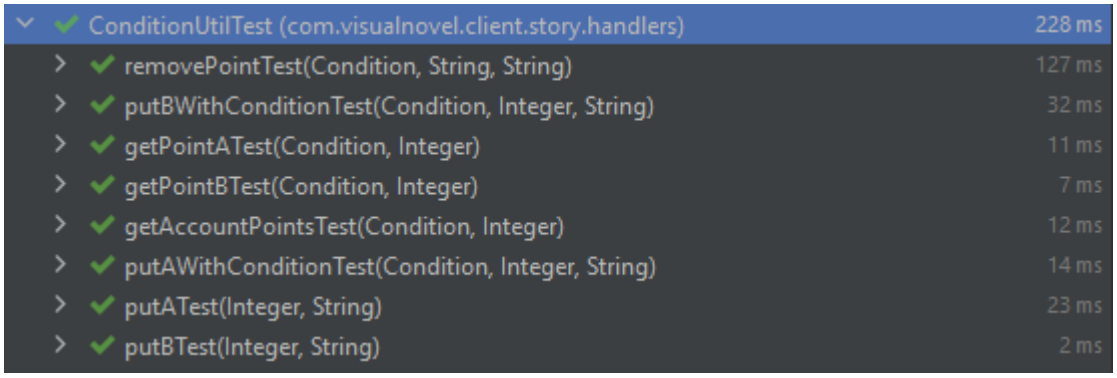
РЕЗУЛЬТАТИ БЛОКОВОГО ТЕСТУВАННЯ



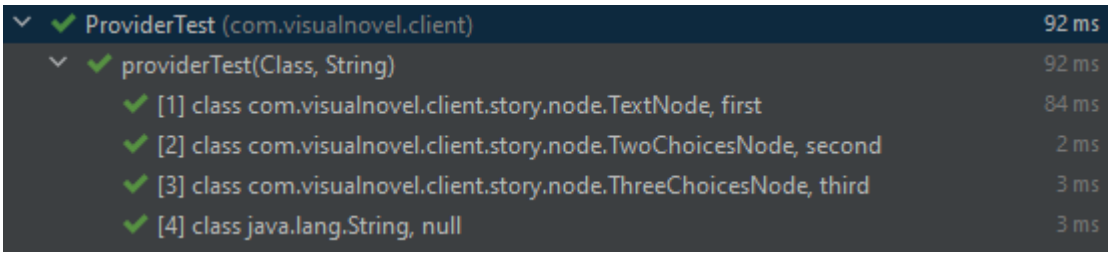
Результати виконання тестів для метода валідації паролів



Результати виконання тестів для методів валідації адрес електронної пошти та імен користувачів

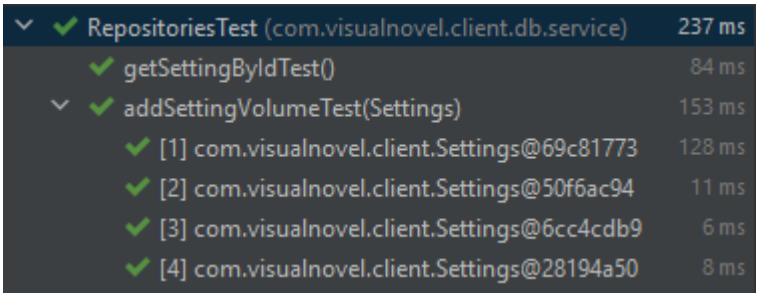


Результати виконання тестів для методів класу ConditionUtil



Результати виконання тестів для методів класу Provider

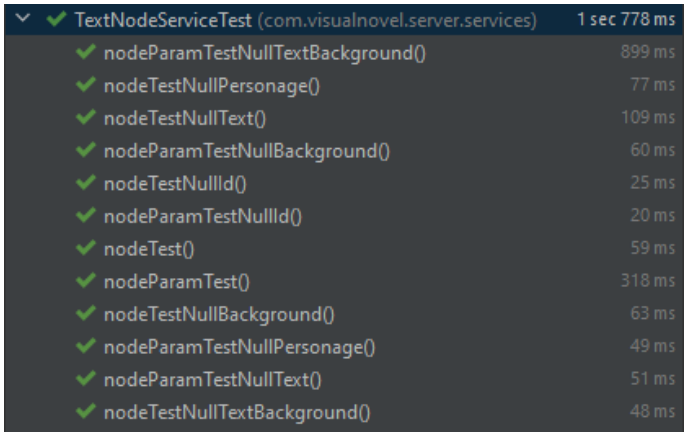
РЕЗУЛЬТАТИ ІНТЕГРАЦІЙНОГО ТЕСТУВАННЯ



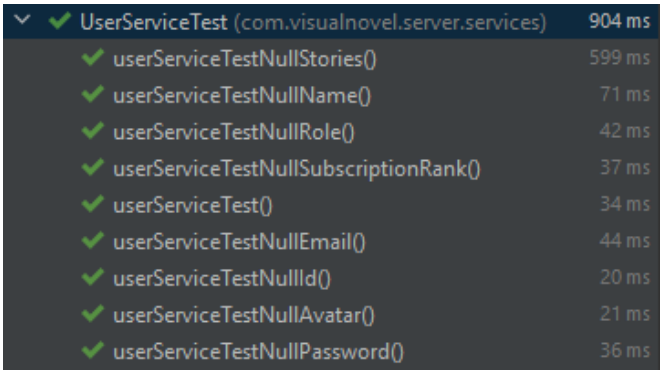
Результати тестування класу SettingsRepository

```
2023-06-17 02:10:45.666  INFO 13172 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 9.321 seconds (JVM running for 10.761)
2023-06-17 02:11:51.586  INFO 10696 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 7.516 seconds (JVM running for 8.88)
2023-06-17 02:12:19.640  INFO 15508 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 6.287 seconds (JVM running for 7.274)
2023-06-17 02:12:45.713  INFO 16768 --- [  restartedMain] c.visualnovel.server.ServerApplication
 : Started ServerApplication in 6.593 seconds (JVM running for 7.574)
```

Результати декількох запусків серверу

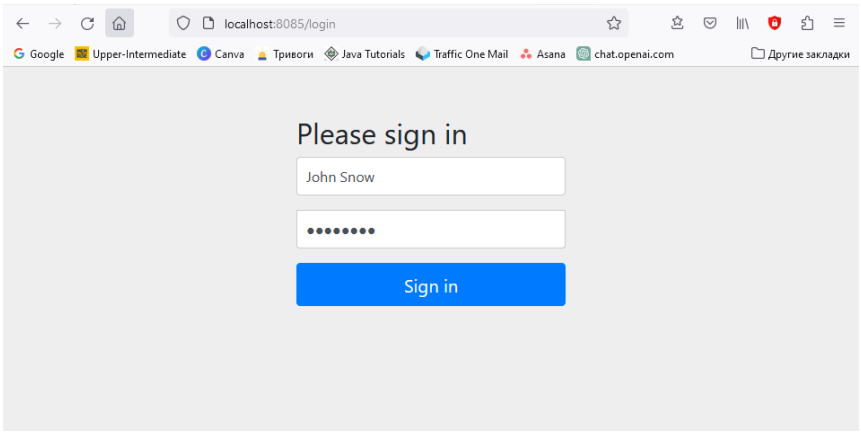


Результати тестування класу TextNodeService

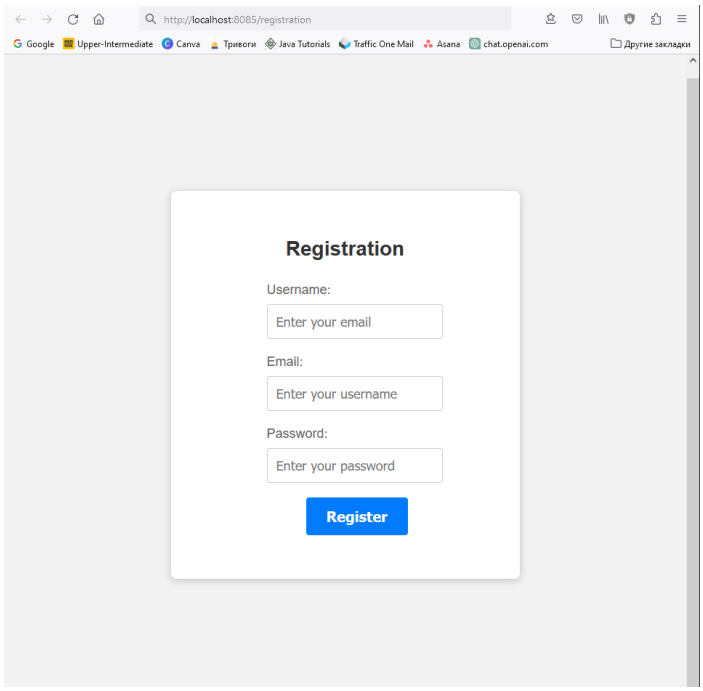


Результати тестування взаємодії із БД за допомогою класів

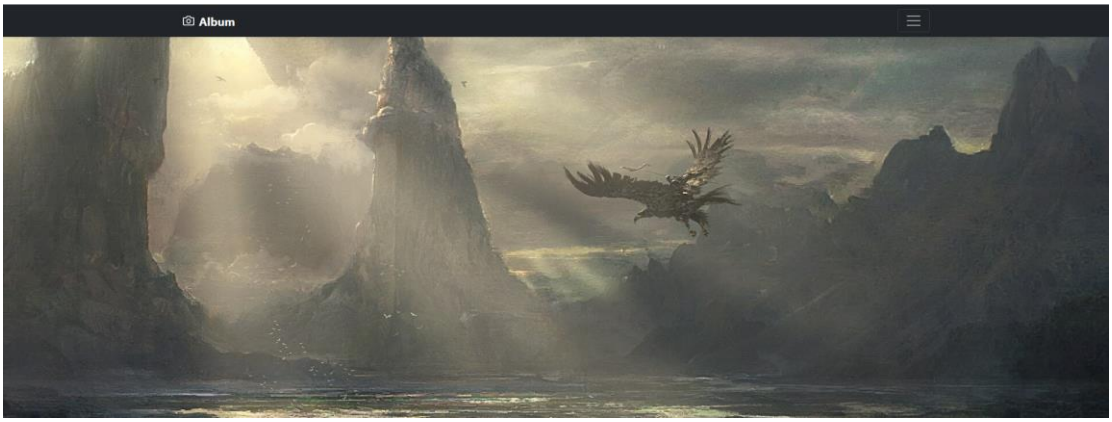
РЕЗУЛЬТАТИ ТЕСТУВАННЯ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ



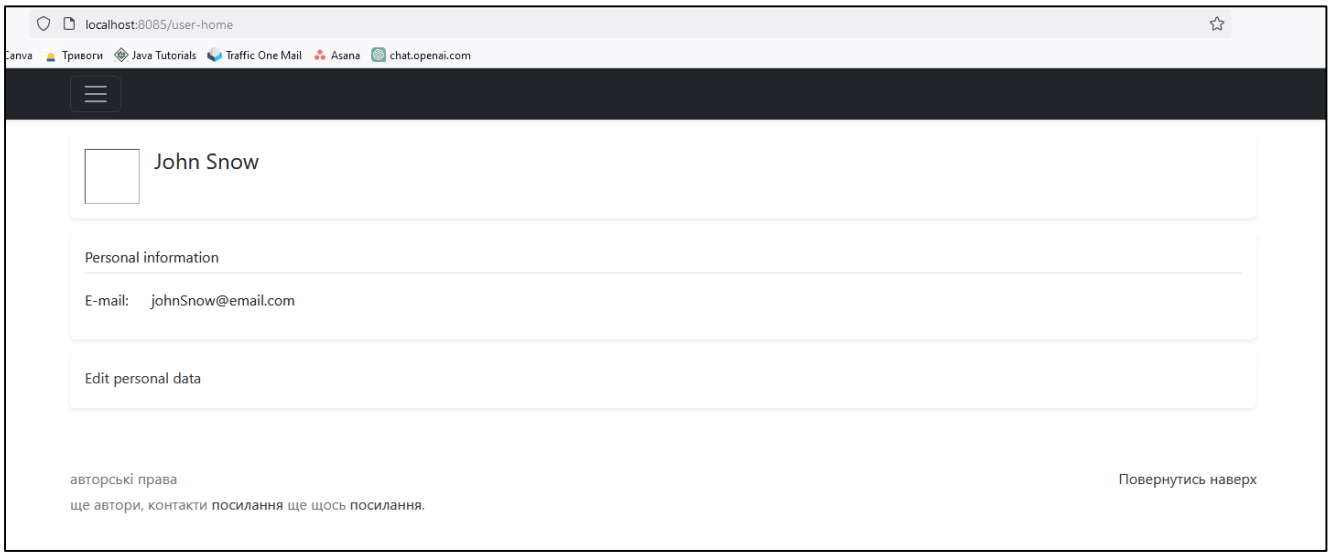
Вигляд форми для автентифікації



Вигляд форми для реєстрації

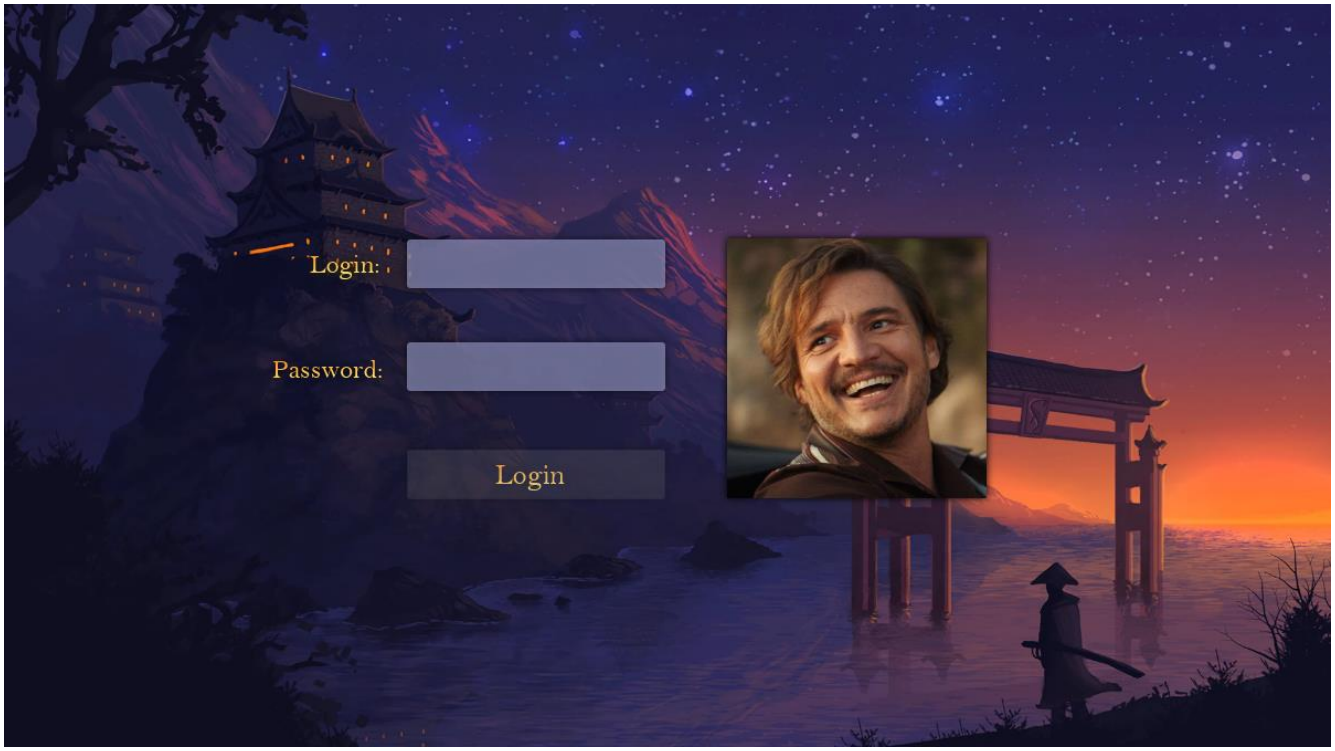


Результат переходу за адресами

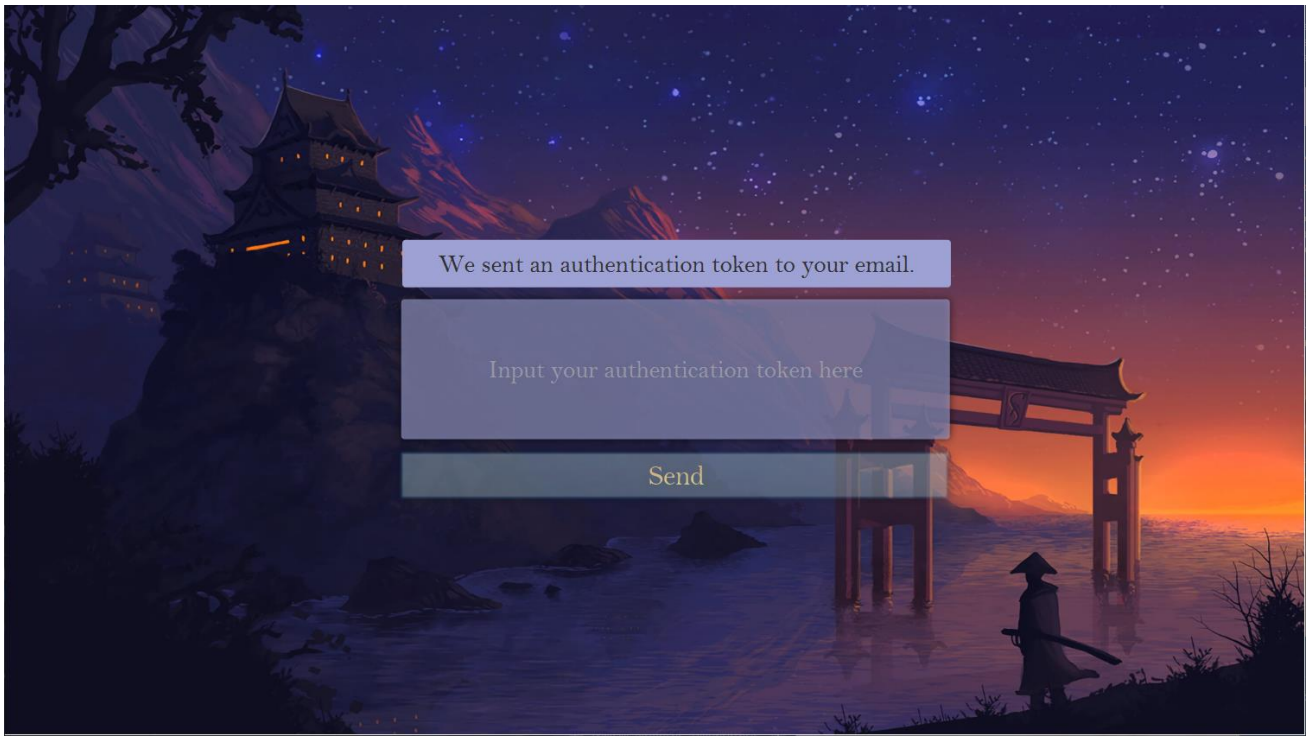


Вигляд сторінки облікового запису користувача та редагування персональних даних

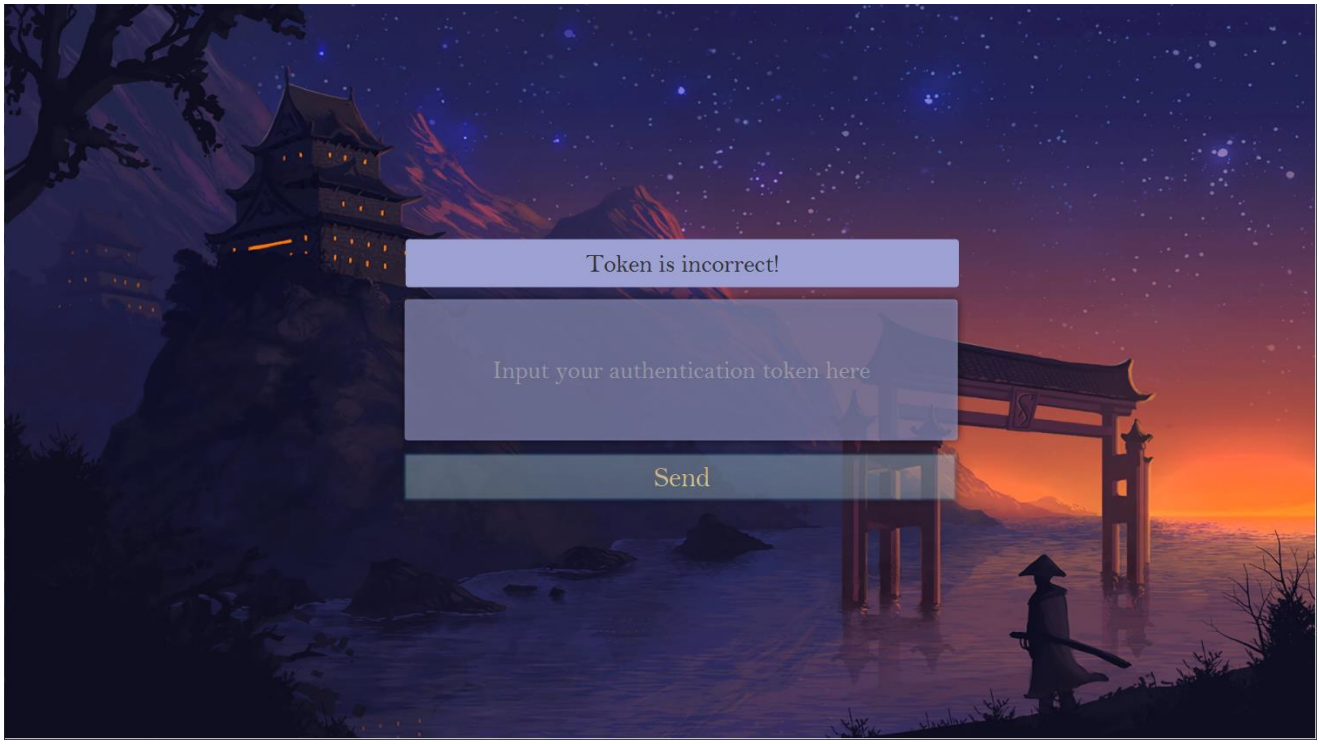
Вигляд отриманого електронного листа із токеном



Вікно автентифікації в застосунку

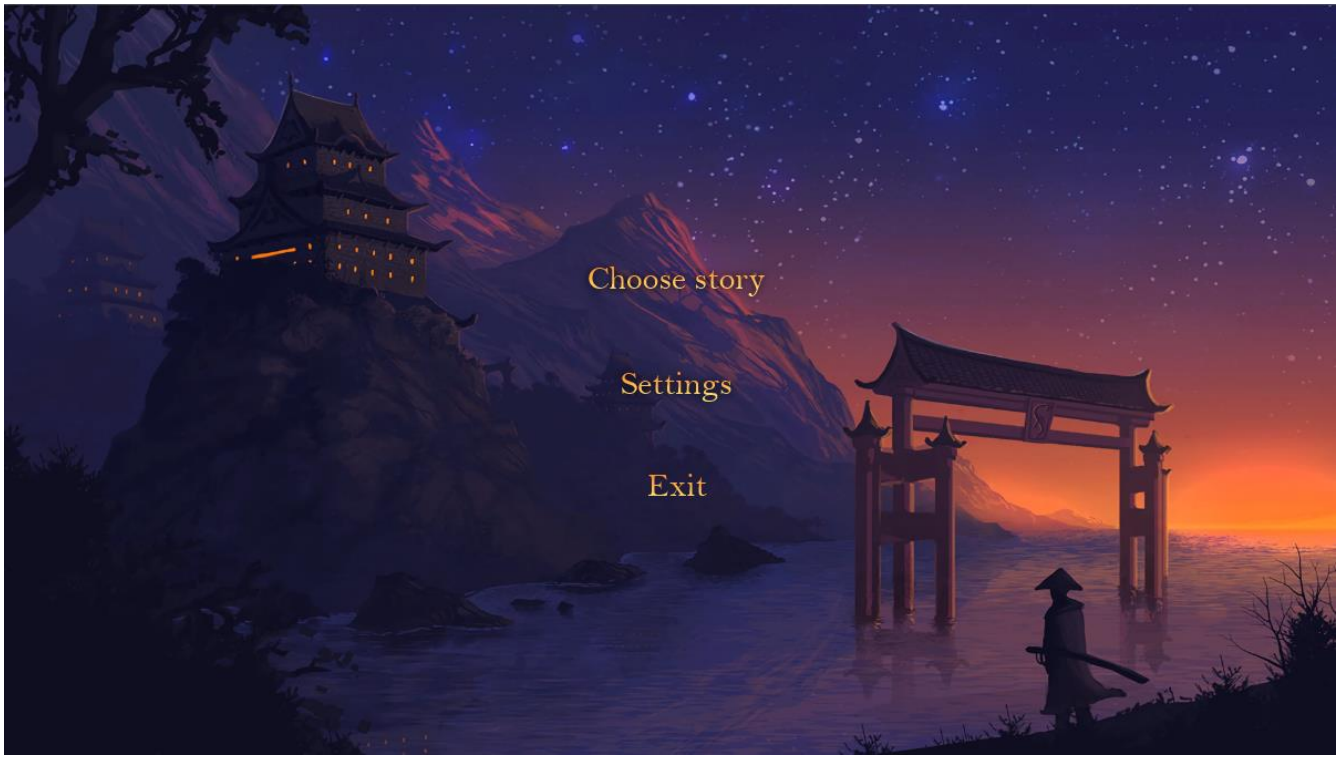


Вигляд вікна із полем для введення токена

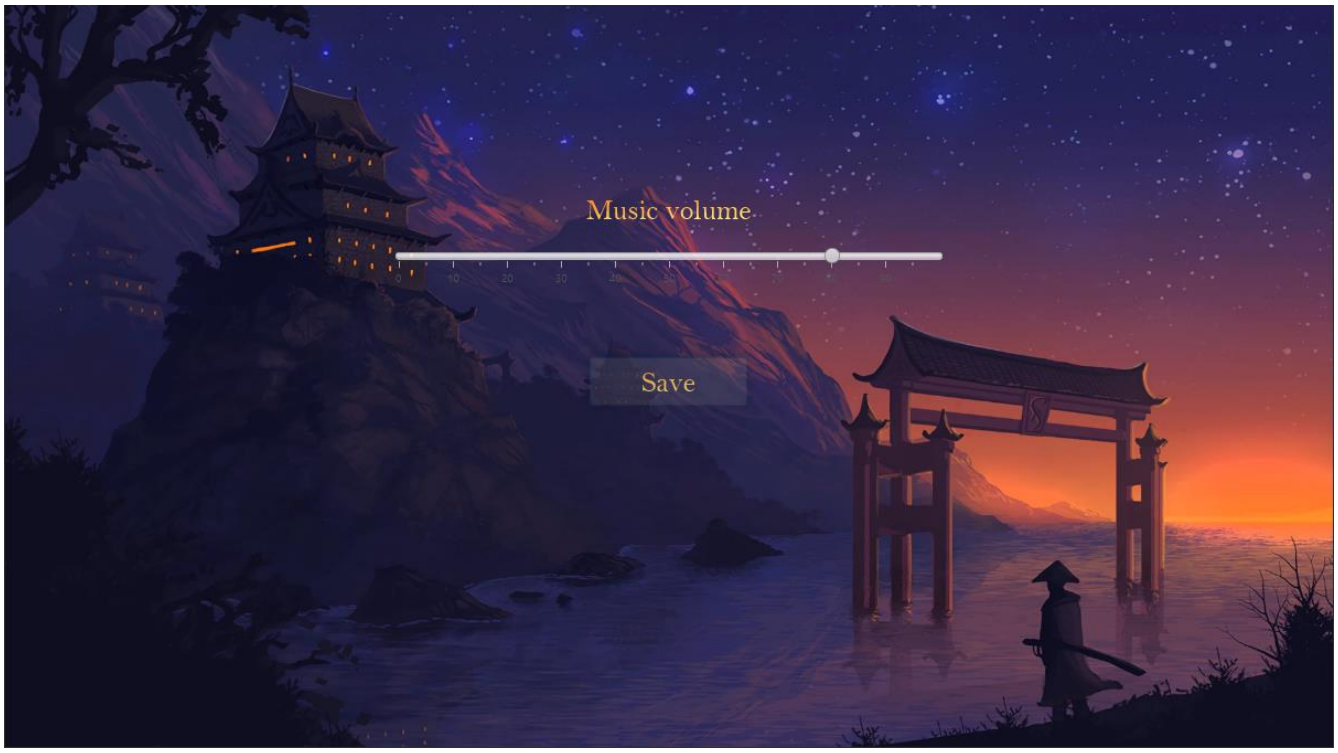


Вигляд вікна із повідомленням про некоректний токен

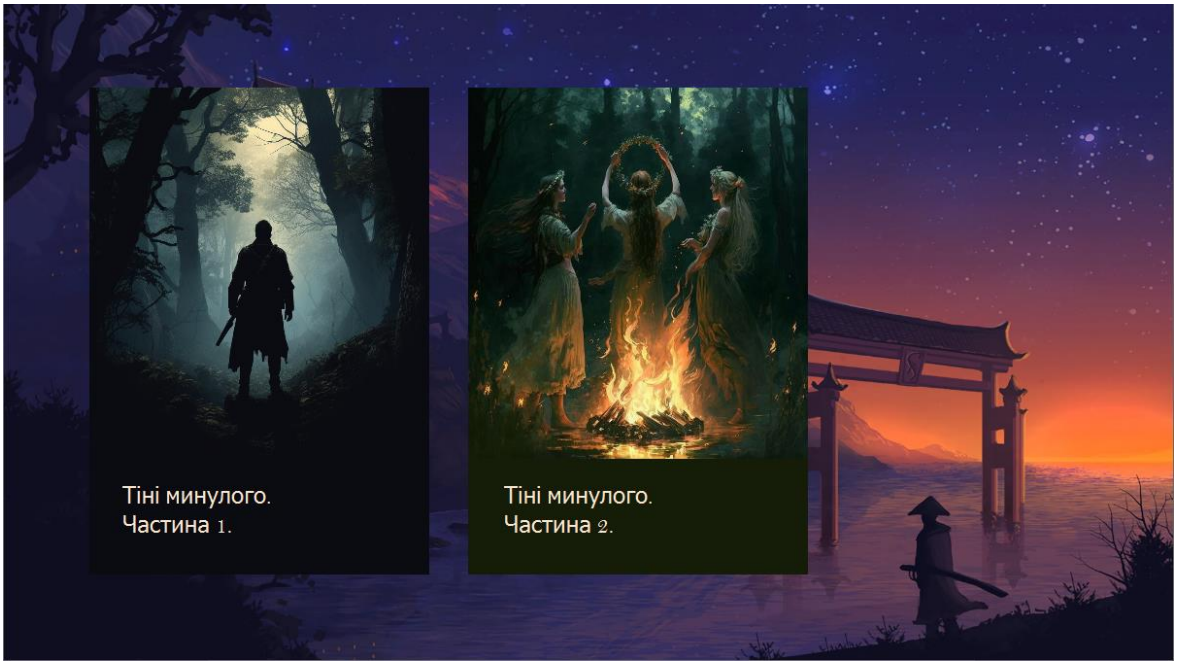
РЕЗУЛЬТАТИ ТЕСТУВАННЯ ЗАХИЩЕНОГО ЗАСТСОУНКУ



Вигляд вікна головного меню

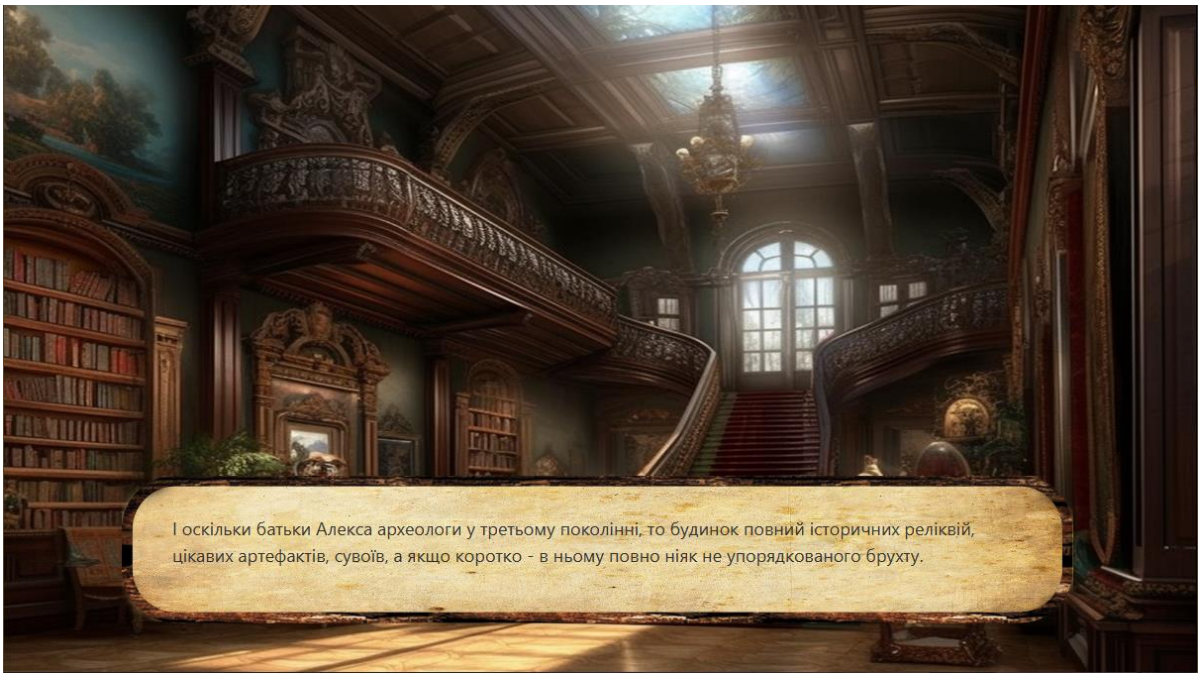


Вигляд вікна із налаштуваннями

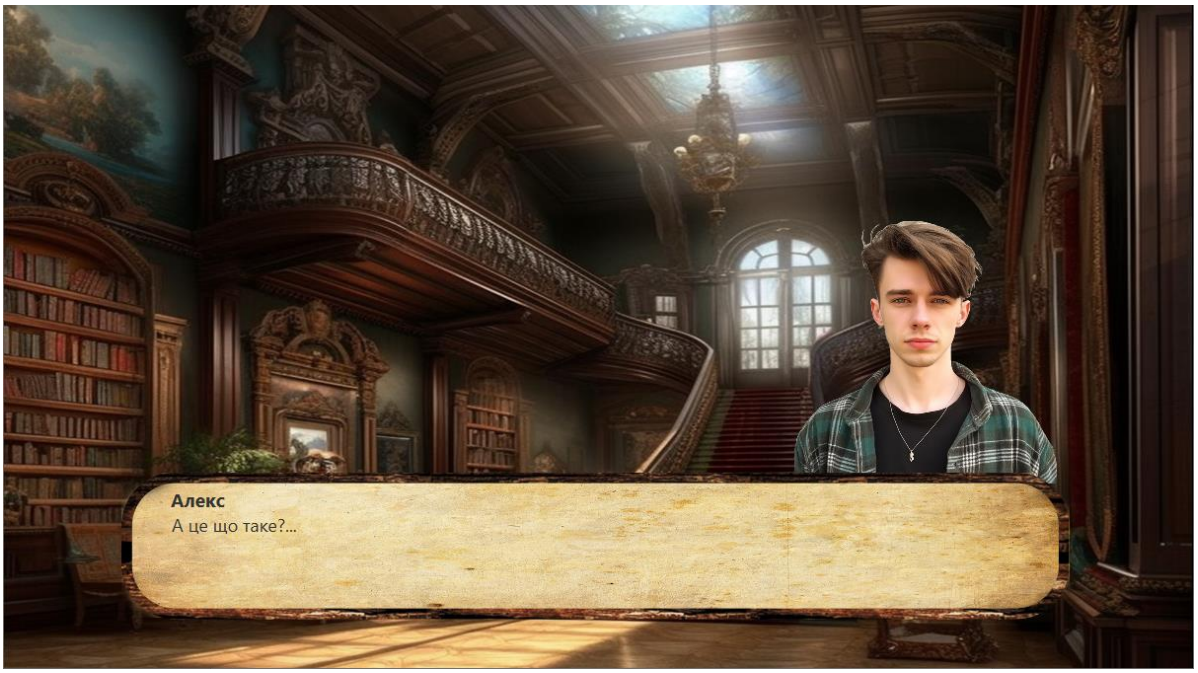


Вигляд вікна із вибором доступних історій

Вигляд вікна із повідомленням про початок історії



Вигляд сцени без персонажа

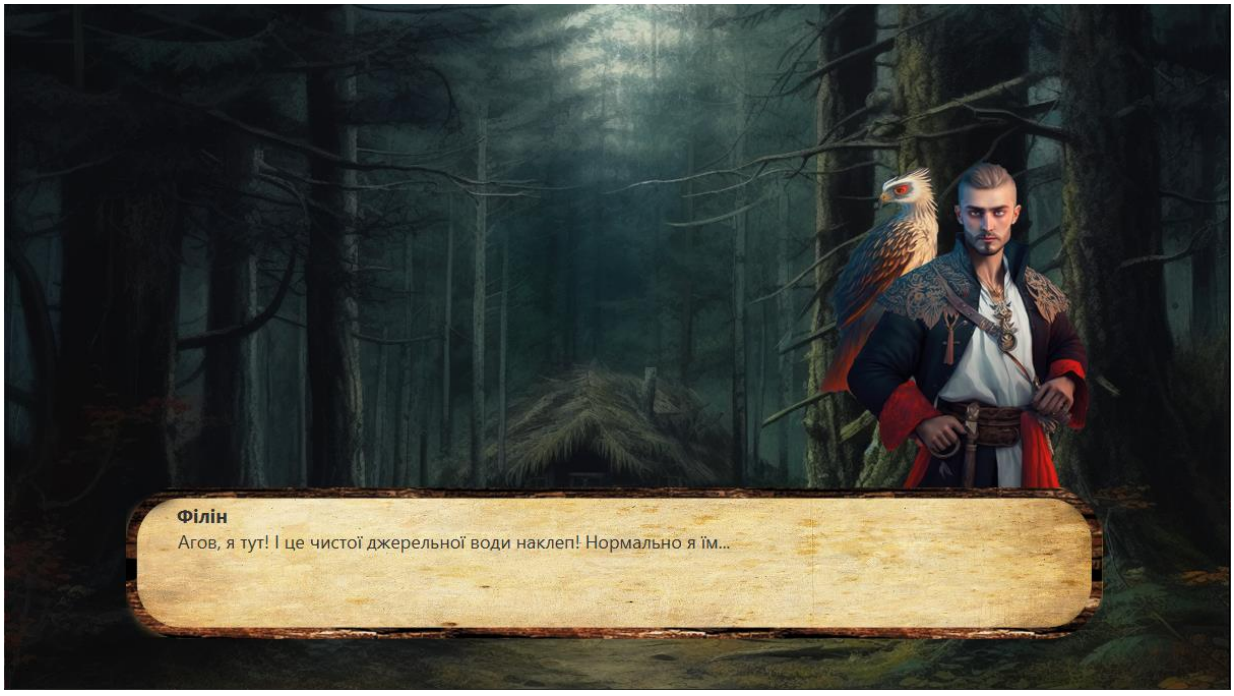


Вигляд сцени із персонажем

Приклад зміни емоції персонажа

Приклад перебігу сюжету

Приклад перебігу сюжету при виборі першої відповіді



Приклад перебігу сюжету при виборі другої відповіді