ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для декодування снів на основі методики Hall-Van de Castle»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Вадим ШЕЛЕСТИНСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Вадим ШЕЛЕСТИНСЬКИЙ

Керівник: Юрій АБРОСКІН
асистент кафедри

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шелестинському Вадиму Анатолійовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для декодування снів на основі методики Hall-Van de Castle»
керівник кваліфікаційної роботи асистент кафедри ІПЗ Юрій АБРОСКІН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки природної мови (NLP), зокрема токенізацію, лематизацію, семантичний аналіз і контекстну обробку текстів; опис методів вилучення психологічно значущої інформації з текстів сновидінь; основи класифікації за методикою Hall–Van de Castle; технічна документація щодо GPT API (OpenAI),

Java-бібліотек для роботи з JSON, а також Spring Boot і Java Persistence API (JPA) для реалізації сервісів обробки тексту та збереження результатів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд психологічних підходів до інтерпретації сновидінь та аналіз методики Hall–Van de Castle.
2. Аналіз сучасних інструментів обробки природної мови та їх застосування у психологічному контексті.
3. Проєктування архітектури програмного забезпечення для інтерпретації снів із використанням GPT API.
4. Реалізація програмного забезпечення для автоматизованої класифікації снів за методикою Hall–Van de Castle.
5. Тестування функціональності системи та верифікація результатів аналізу сновидінь.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. BPMN діаграма
5. Діаграма прецеденців
6. Діаграма класів
7. Діаграма послідовності
8. Екранні форми
9. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02 - 04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03 - 13.03.2025	
3	Вивчення методики Hall–Van de Castle та теоретичних засад аналізу сновидінь	14.03 - 20.03.2025	
4	Проєктування архітектури програмного забезпечення та моделі взаємодії з GPT API	21.03 - 27.03.2025	
5	Реалізація сервісів аналізу снів, інтеграція з OpenAI API, збереження результатів у БД	28.03 - 17.04.2025	
6	Тестування програмного забезпечення та верифікація результатів	18.04 – 28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Вадим ШЕЛЕСТИНСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій АБРОСКІН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 2 табл., 12 рис., 25 джерел.

Мета роботи – спрощення організації самопізнання через інтерпретацію сновидінь на основі методики Hall–Van de Castle.

Об'єкт дослідження – процес інтерпретації сновидінь через класифікацію змісту за визначеними психологічними категоріями методики Hall–Van de Castle.

Предмет дослідження – програмне забезпечення для аналізу та класифікації сновидінь із використанням GPT-моделі і методики Hall–Van de Castle.

Короткий зміст роботи: У роботі проаналізовано теоретичні підходи до аналізу сновидінь, структуру методики Hall–Van de Castle та сучасні засоби NLP і GPT для семантичного аналізу текстів. Проведено огляд аналогічного програмного забезпечення, визначено їх обмеження. Запропоновано архітектуру десктопного застосунку SomniCast, реалізованого на Java 17 із використанням Spring Boot, GPT API та H2 Database. Основні функції системи включають: введення тексту сну, надсилання запиту до GPT, обробку відповіді, класифікацію елементів сну, збереження результатів і виведення історії. Проведено тестування та верифікацію результатів на прикладах, проаналізовано типові помилки, що виникають при роботі із системою.

Сфера використання – програмне забезпечення SomniCast може бути застосоване для індивідуального психологічного самопізнання, у дослідницькій та психотерапевтичній практиці, а також для навчальних цілей у сфері психології та штучного інтелекту.

КЛЮЧОВІ СЛОВА: GPT, NLP, JAVA, SPRING BOOT, API, СНОВИДІННЯ, ІНТЕРПРЕТАЦІЯ, HALL–VAN DE CASTLE, ПСИХОЛОГІЧНИЙ АНАЛІЗ, АВТОМАТИЗАЦІЯ, ДЕСКТОПНЕ ПЗ.

ЗМІСТ

ВСТУП

Актуальність дослідження: У сучасному інформаційному суспільстві зростає інтерес до інтеграції психологічних знань із можливостями штучного інтелекту. Одним із таких напрямів є автоматизована інтерпретація сновидінь — складного феномену, що відображає несвідомі переживання, емоції та символічні образи особистості. Традиційні методи аналізу снів потребують значного часу й участі фахівця, що обмежує їх застосування в повсякденному житті. Сучасні досягнення в обробці природної мови (NLP), а також розвиток великих мовних моделей, зокрема GPT, відкривають нові можливості для побудови інструментів, які дозволяють автоматично аналізувати текстові описи снів. Методика Hall–Van de Castle, як одна з найбільш структурованих і верифікованих методик контент-аналізу сновидінь, надає надійну основу для формалізації результатів декодування. Поєднання цієї методики з інструментами NLP дозволяє розробити програмне забезпечення, яке може використовуватись як у наукових дослідженнях, так і в особистому психологічному самопізнанні.

Об’єкт дослідження є процес інтерпретації сновидінь через класифікацію змісту за визначеними психологічними категоріями методики Hall–Van de Castle.

Предмет дослідження є програмне забезпечення для аналізу та класифікації сновидінь із використанням GPT-моделі і методики Hall–Van de Castle.

Мета роботи є спрощення організації самопізнання через інтерпретацію сновидінь на основі методики Hall–Van de Castle.

Методи дослідження: У процесі роботи було проаналізовано наукові джерела, що стосуються психологічного аналізу сновидінь, методології Hall–Van de Castle та можливостей обробки природної мови. Також було здійснено вивчення технологічних засобів, необхідних для реалізації програмного забезпечення, включно з фреймворком Spring Boot, мовною моделлю GPT, H2-базою даних і бібліотеками для парсингу JSON. Після цього проведено проєктування архітектури системи, реалізацію програмного продукту та тестування функціональних

компонентів на основі тестових прикладів для оцінки точності класифікації та зручності взаємодії з інтерфейсом користувача.

Наукова новизна полягає в реалізації автоматизованої системи аналізу сновидінь, яка використовує NLP та LLM-модель для категоризації відповідно до методики Hall–Van de Castle з можливістю збереження історії результатів і персоналізації взаємодії.

Практична значущість роботи полягає у створенні локального десктопного застосунку з україномовним інтерфейсом, який дозволяє користувачам легко отримувати інтерпретації своїх снів без залучення спеціаліста, що робить психологічний аналіз доступним, зручним і автоматизованим.

1 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ІНТЕРПРЕТАЦІЇ СНОВИДІНЬ ТА МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ

1.1 Теоретичні підходи до аналізу сновидінь у психології

Сновидіння здавна викликали зацікавлення як у філософів, так і в науковців, оскільки вони є особливою формою психічної активності, яка виникає у стані сну. У сучасній психології сни розглядаються як важливе джерело інформації про глибинні процеси психіки людини. Інтерпретація сновидінь використовується як діагностичний та терапевтичний інструмент, здатний пролити світло на внутрішні конфлікти, емоційний стан та пережиті психотравми.

Початок наукового підходу до аналізу снів пов'язаний із постаттю Зиґмунда Фройда, який у праці «Тлумачення сновидінь» (1900) сформулював основи психоаналітичної теорії сновидінь. Згідно з його концепцією, сни є реалізацією витіснених бажань, що не можуть бути задоволені в реальному житті через обмеження моралі та культури. Щоб уникнути конфлікту зі свідомістю, ці бажання маскуються і трансформуються у символи, які потребують тлумачення. До таких механізмів належать конденсація, зсув, символізація. Фройд також вважав, що аналіз снів дозволяє проникнути в несвідоме — приховану частину психіки, яка керує поведінкою людини.

Розширення й поглиблення розуміння природи сновидінь здійснив Карл Густав Юнг, який відкинув редукціонізм фройдизму й запропонував власний аналітичний підхід. Юнг вважав, що сни — це не лише прояви індивідуального несвідомого, але й колективного несвідомого, що містить архетипи — універсальні образи, притаманні всьому людству. Сновидіння, за Юнгом, виконують компенсаторну функцію, відновлюючи психічну рівновагу та сприяючи процесу індивідуації — гармонійному розвитку особистості. Таким чином, тлумачення снів за Юнгом не обмежується виявленням пригнічених бажань, а передбачає глибше розуміння особистісного зростання.

У XX столітті інтерес до сновидінь зазнав спаду внаслідок домінування біхевіористичного підходу, який відкидав суб'єктивні феномени як ненадійні для наукового вивчення. Проте з появою когнітивної психології увага до сновидінь відновилася. Згідно з когнітивною теорією, сновидіння — це продовження мислення в особливому стані свідомості. Відомий дослідник Д. Домхофф підкреслює, що сновидіння мають змістову структуру, яка відображає інтереси, установки й досвід сновидця. Сни аналізуються як наративи, які можна вивчати за допомогою контент-аналізу, порівнювати між групами людей, досліджувати у зв'язку з віком, статтю, психічним станом.

Значний внесок у наукову систематизацію аналізу сновидінь зробили Келвін Холл та Роберт Ван де Кастл. У 1960-х роках вони запропонували методику кількісного аналізу снів — Hall–Van de Castle system. Методика базується на кодуванні сновидінь за категоріями: персонажі, дії, емоції, соціальні взаємодії, місця та результати. Це дозволяє порівнювати сни між різними людьми та групами, виявляти психологічні закономірності. Цей підхід став основою для створення електронних баз снів та статистичних досліджень у сфері психології сновидінь.

Паралельно із психологічними підходами, розвивалися нейронаукові моделі сновидінь. Однією з найвідоміших є модель активації-синтезу Гобсона і МакКарлі (1977), яка пояснює сни як результат випадкової нейронної активності мозкового стовбура під час фази швидкого сну (REM-sleep). Кора головного мозку намагається впорядкувати ці імпульси, створюючи сюжетні образи, які часто мають химерний або нелогічний характер. Хоча ця теорія пояснює фізіологічну основу сновидінь, вона не враховує їх змістової глибини та індивідуальної значущості.

Інший напрям — екзистенційна та феноменологічна психологія — зосереджується на досвіді сновидіння як вияві життєвих цінностей, тривог і сенсів. Віктор Франкл, Людвіг Бінсвангер та Медард Босс розглядали сни як індикатори екзистенційного стану особистості. Їхній підхід полягає не стільки в інтерпретації, скільки в усвідомленні пережитого досвіду та його значення для особистісного вибору.

Таким чином, у сучасній психології сформувалося кілька основних підходів до аналізу сновидінь: психоаналітичний, аналітичний (юнгіанський), когнітивний, нейрофізіологічний, феноменологічний. Кожен із них висвітлює різні аспекти сну: символіку, структуру, функцію, фізіологію чи екзистенційну значущість. У практиці психологічного аналізу сновидінь усе частіше застосовуються комбіновані методики, а також інструменти сучасної обробки текстів, зокрема методи NLP (Natural Language Processing), що дозволяють підвищити об’єктивність, швидкість і репрезентативність аналізу.

1.2 Методика Hall–Van de Castle: структура та принципи

У сфері наукового дослідження сновидінь важливим завданням є створення інструментів, здатних забезпечити об’єктивний і формалізований підхід до інтерпретації снів. Однією з найбільш авторитетних систем такого типу стала методика Hall–Van de Castle, розроблена в 1960-х роках американськими психологами Келвіном Голлом (Calvin S. Hall) і Робертом Ван де Кастлом (Robert Van de Castle). Вона заснована на контент-аналізі текстових описів снів і дозволяє досліджувати не окремі випадки, а великі масиви даних, виявляючи закономірності у структурі сновидінь.

Ключова перевага методики полягає в її стандартизованості. Замість суб’єктивного тлумачення, яке притаманне багатьом класичним підходам (наприклад, психоаналізу), Hall–Van de Castle пропонує чітко визначені категорії та процедури їхньої фіксації. Такий підхід відкриває можливості для кількісного аналізу та побудови статистичних моделей, що особливо актуально в контексті сучасних цифрових технологій.

Основні категорії аналізу

Методика поділяє зміст сну на низку ключових категорій, що дозволяють відтворити повну картину внутрішнього емоційного та когнітивного стану сновидця. До таких категорій належать:

- Персонажі — центральні та другорядні фігури у сновидінні. Розрізняються за типом (люди, тварини, надприродні істоти), а також за рівнем знайомства зі сновидцем (своє «я», знайомі, незнайомці).
- Соціальні взаємодії — дії між персонажами, які мають соціальний зміст: агресія, допомога, спілкування, суперечки, взаєморозуміння тощо.
- Емоції — визначають загальне емоційне тло сну. Вони класифікуються на позитивні (радість, любов), негативні (страх, гнів) і нейтральні або змішані.
- Дії сновидця — аналізується активність самого сновидця: що він робить, які дії спрямовані на нього, чи виявляє ініціативу або пасивність.
- Образ середовища (місце дії) — визначення простору, в якому розгортається сюжет: знайоме чи незнайоме місце, реалістичне чи фантастичне.
- Результати епізодів — як закінчується той чи інший фрагмент сну: перемогою, поразкою, уникненням, зміною ситуації або трансформацією.

Алгоритм застосування методики

1. Отримання тексту сновидіння — сновидець детально описує сюжет, діалог, враження.
2. Попередня обробка — текст очищається від орфографічних або граматичних помилок, приводиться до уніфікованого вигляду.
3. Розмітка — визначаються ключові елементи відповідно до категорій. Це може здійснюватися вручну або автоматично (у цифрових інструментах).
4. Кількісний аналіз — підраховуються кількість кожної категорії, зіставляються пропорції, аналізуються патерни.
5. Порівняння з нормами — результати порівнюються з базами еталонних снів, складених за статтю, віком, соціальним статусом тощо.

Варто зазначити, що Hall–Van de Castle створили детальні статистичні моделі на основі аналізу тисяч снів чоловіків і жінок. Це дозволило встановити типові показники, наприклад, переважання агресії в чоловічих снах або більшу кількість соціальних контактів у жіночих. Такий емпіричний підхід забезпечує не лише точність, а й можливість порівняння результатів між різними вибірками.

Практичне застосування

Методика Hall–Van de Castle знайшла широке застосування у психотерапії, клінічній психології, дослідженнях впливу стресу, посттравматичних станів, а також у вивченні культурних відмінностей у сприйнятті сновидінь. Наприклад, зміни в категорії «емоції» можуть свідчити про наявність тривожного або депресивного стану, а домінування агресії — про внутрішній конфлікт.

Окрім суто психологічного значення, методика стала цінною і в міждисциплінарних дослідженнях: соціологічних, лінгвістичних, комп'ютерно-аналітичних. Зокрема, вона є базовим елементом для автоматизованого аналізу текстів снів у системах штучного інтелекту.

Застосування в дипломному проєкті

У межах цієї кваліфікаційної роботи було реалізовано програмне забезпечення, яке використовує методику Hall–Van de Castle як основу для класифікації структурованих елементів сну. Користувач вводить опис сновидіння у вільній формі, після чого система за допомогою GPT API виконує попередню обробку, формуючи відповідь у вигляді семантично впорядкованих блоків. Далі спеціальний Java-модуль парсить отриману відповідь і автоматично розподіляє елементи по категоріях відповідно до шкал методики Hall–Van de Castle.

Таким чином, інструмент, створений у межах дослідження, дозволяє автоматизувати той процес, що традиційно потребував участі кваліфікованого аналітика. Завдяки цьому методика може бути використана не лише у наукових чи клінічних умовах, а й звичайними користувачами без фахової підготовки.

1.3 Обробка природної мови (NLP) у завданнях психологічного аналізу

У сучасному світі обробка природної мови (Natural Language Processing, NLP) є однією з найважливіших галузей штучного інтелекту, що дозволяє системам інтерпретувати, аналізувати, структурувати й генерувати тексти, створені людиною.

Вона поєднує в собі елементи лінгвістики, інформатики, статистики та когнітивних наук, забезпечуючи інструментарій для роботи з мовними даними будь-якої складності.

Особливу значущість NLP набуває в контексті психологічного аналізу, де тексти розглядаються як відображення внутрішнього стану особистості.

Сновидіння — одна з форм вербального вираження внутрішнього психоемоційного досвіду. Їх текстовий опис часто містить алегоричні образи, метафоричні зв’язки, нелінійні сюжети та емоційно забарвлені елементи. Саме тому традиційна обробка таких текстів потребувала участі кваліфікованого психолога, здатного здійснити семантичний та символічний аналіз. Використання NLP дозволяє частково автоматизувати цей процес і зробити його доступним у цифрових системах.

Основні етапи NLP-обробки текстів сновидінь

Типовий NLP-процес обробки опису сну включає кілька взаємопов’язаних кроків:

1. Токенізація — розбиття тексту на базові одиниці: слова, речення або абзаци. Це дозволяє структуровано аналізувати зміст, виділяючи граматичні та логічні межі фраз.

2. Лематизація — приведення слів до початкових форм (лексем). Наприклад, слова "біг", "бігла", "бігти" об’єднуються як лексема "бігти". Це забезпечує більш точну обробку смислів.

3. Визначення частин мови (POS-tagging) — класифікація слів за граматичними ознаками (іменник, дієслово, прикметник тощо), що важливо для подальшої семантичної обробки.

4. Семантичний аналіз — виявлення смислових зв’язків, контекстів, прихованих значень. Це особливо актуально для аналізу символіки снів.

5. Емоційна тональність — визначення домінантних емоційних маркерів (страх, гнів, радість), що дозволяє оцінити психоемоційний фон сновидіння.

6. Анотація категорій — зіставлення фрагментів тексту з певними категоріями, наприклад, відповідно до методики Hall–Van de Castle: персонажі, дії, соціальні взаємодії, емоції тощо.

Роль NLP у психологічному аналізі

Використання NLP-технологій у психології відкриває нові горизонти для дослідження психоемоційного стану людини через мовні патерни. Тексти снів є джерелом багатошарової інформації, яка може вказувати на рівень тривоги, наявність внутрішніх конфліктів, самооцінку, реакцію на стрес чи травматичні події. Наприклад, часте вживання слів із семантикою втечі, загрози, агресії може сигналізувати про посттравматичний досвід, тоді як поява позитивної лексики — про емоційну стабілізацію.

Завдяки NLP можна автоматизувати рутинну частину аналізу, забезпечити уніфікованість процедур та мінімізувати суб’єктивність тлумачення. Це особливо актуально в умовах, коли дослідження охоплює велику кількість текстів снів, наприклад, у клінічних або соціологічних опитуваннях.

Технічна реалізація NLP у дипломному проєкті

У рамках розробки програмного забезпечення, що описується в цій кваліфікаційній роботі, NLP-процес реалізований як поєднання кількох технологічних рішень:

- Використання GPT API — для попереднього контекстного аналізу сну. GPT здатна «розуміти» структуру тексту на глибшому рівні, виявляючи як семантичні, так і емоційні складові.
- Java-парсер — обробляє відповідь від GPT, виділяє ключові категорії, перетворює їх у внутрішньо представлені об’єкти.
- Передобробка введення — включає первинну нормалізацію, наприклад, усунення зайвих розділових знаків або повторів.

Ключовою особливістю розробленої системи є її адаптація до української мови, що потребує врахування морфологічних особливостей і граматичних структур. У той час як більшість NLP-бібліотек орієнтовані на англійську мову,

інтеграція GPT дозволяє обійти ці обмеження та отримувати глибокий контекстуальний аналіз незалежно від мови введення.

Переваги NLP у завданні інтерпретації сновидінь

- Об’єктивність — на відміну від «ручного» аналізу, NLP забезпечує відтворюваність і відсутність впливу суб’єктивних факторів.
- Швидкість — автоматизована система здатна за кілька секунд проаналізувати текст, на що людині знадобилося б у десятки разів більше часу.
- Масштабованість — можливість обробляти великі обсяги даних (сотні й тисячі снів).
- Гнучкість — система може адаптуватися під різні методики аналізу, не обмежуючись Hall–Van de Castle.

Таким чином, NLP-технології стають основним механізмом перетворення текстів снів у структуровані дані, придатні для подальшої класифікації, порівняння або візуалізації. У поєднанні з GPT-моделлю це дозволяє не лише аналізувати поверхневі характеристики тексту, а й здійснювати семантичне моделювання внутрішнього світу користувача. У наступному підрозділі буде розглянуто, яким чином великі мовні моделі (LLM), зокрема GPT, поглиблюють можливості NLP, забезпечуючи глибший рівень аналізу.

1.4 Аналіз аналогів програмного забезпечення для інтерпретації снів

Упродовж останнього десятиліття зростає зацікавленість користувачів до інструментів, що дозволяють інтерпретувати сновидіння за допомогою цифрових технологій. Сучасні застосунки, орієнтовані на обробку текстів снів, варіюються за ступенем автоматизації, наукового підґрунтя та рівня персоналізації результатів. Це зумовлює необхідність системного аналізу існуючих рішень для виявлення переваг, недоліків і функціональних прогалин, які може заповнити розроблене в межах дипломної роботи програмне забезпечення — SomniCast.

Аналіз охоплює найпопулярніші на ринку інструменти для обробки та інтерпретації снів: DreamApp, Dream Interpreter AI, DreamAI, Sleep as Android. Всі вони мають різний функціонал, спрямованість і цільову аудиторію, що дозволяє сформувати репрезентативне порівняння за низкою ключових критеріїв:

- Підтримка методики Hall–Van de Castle — наявність у застосунку чітко структурованого підходу до класифікації елементів сну;
- Використання GPT або інших LLM-моделей — впровадження інтелектуального семантичного аналізу за допомогою великих мовних моделей;
- Наявність україномовного інтерфейсу — важлива умова локалізації для україномовних користувачів;
- Платформна сумісність — можливість роботи в різних операційних середовищах (Windows, macOS, Linux);
- Інтеграція через API — здатність застосунку до розширення або зв’язку з іншими сервісами;
- Функції візуалізації — графічне представлення структури снів.

Таблиця 1.1

Аналіз аналогів

	DreamApp	**Dream Interpreter AI**	**DreamAI**	**Sleep as Android**	**SomniCast**
Підтримка методики Hall – Van de Castle	-	-	-	-	+
Використання GPT для NLP-аналізу	+	+	+	-	+
Україномовний інтерфейс	-	-	-	-	+
Десктопна кросплатформна версія	-	-	-	-	+
Відкрите API / інтеграції	-	-	-	+	+
Візуалізація статистики сновидінь	+	-	+	+	+

Як свідчить аналіз, лише одиничні продукти, такі як DreamAI чи DreamApp, частково реалізують можливість NLP-аналізу снів з опорою на машинне навчання. Проте, жоден з аналогів не використовує методику Hall–Van de Castle як основну систему категоризації. Це вказує на відсутність у конкурентів науково верифікованого підходу до декодування змісту сновидінь, що обмежує точність отриманих результатів.

Ще однією значною прогалиною більшості розглянутих систем є відсутність україномовної локалізації. Оскільки багато користувачів у регіоні вживають українську мову у повсякденному житті, відсутність підтримки інтерфейсу цією мовою знижує доступність та зручність використання. У цьому контексті SomniCast має безперечну перевагу, оскільки повністю орієнтований на україномовного користувача — починаючи від інтерфейсу до обробки лексичних особливостей сну.

Крім того, майже всі аналоги орієнтовані виключно на мобільну платформу, тоді як SomniCast реалізований як десктопна веб-програма з можливістю розгортання на будь-якій ОС через браузер. Це забезпечує гнучкість використання, а також спрощує подальше масштабування або адаптацію інтерфейсу до нових потреб.

Особливу увагу заслуговує наявність відкритого API у SomniCast, що дозволяє розширювати функціональність без необхідності модифікувати основний код. Це є принципово важливою перевагою в контексті інтеграції з іншими системами, зокрема мобільними додатками, науковими базами або персональними щоденниками снів користувача.

Варто відзначити й наявність візуалізації результатів аналізу у вигляді структурованого виводу. Інші аналоги обмежуються переважно текстовими відповідями або шаблонними інтерпретаціями. У SomniCast, навпаки, користувач отримує категоризовану інформацію відповідно до ключових складових методики Hall–Van de Castle: емоції, дії, персонажі, соціальні взаємодії та загальні підсумки. Такий підхід не лише покращує сприйняття результатів, а й дозволяє спростити процес самоспостереження та виявлення змін у структурі снів з часом.

З огляду на вищенаведене, можна стверджувати, що SomniCast вирізняється комплексністю та цілеспрямованістю. На відміну від конкурентів, він не лише поєднує передові інструменти NLP та GPT, але й забезпечує академічну точність за рахунок підтримки методології Hall–Van de Castle. Додатковими перевагами є україномовна підтримка, можливість інтеграції, адаптивність інтерфейсу та розширюваність архітектури.

У сукупності це дозволяє SomniCast позиціонуватися як інноваційне програмне забезпечення нового покоління, здатне задовольнити як індивідуальні потреби користувача, так і науково-дослідницькі запити у сфері психології, психотерапії та когнітивної науки.

1.5 Висновки за результатами аналізу

У результаті проведеного аналізу теоретичних основ інтерпретації сновидінь, методичних підходів до їх класифікації, а також сучасних засобів обробки природної мови, можна сформулювати низку принципових висновків, які лягли в основу подальшої розробки програмного забезпечення для декодування снів.

Насамперед варто підкреслити, що сновидіння є багатокомпонентним феноменом, який поєднує елементи глибинної психології, когнітивних процесів і нейрофізіологічної активності. Аналіз класичних психологічних концепцій, зокрема психоаналітичних поглядів З. Фройда та К. Г. Юнга, доводить, що сни відображають несвідомі бажання, архетипічні образи та внутрішні конфлікти особистості. У межах когнітивної та нейронаукової парадигм сновидіння інтерпретуються як продовження мислення в умовах зниженої свідомості, де активуються емоційні та асоціативні структури мозку. Таким чином, сам феномен сну — це не випадкове явище, а важливий механізм психологічного саморегулювання, символічної репрезентації досвіду та індикатор емоційного стану індивіда.

Другою ключовою позицією є наукова й практична значущість методики Hall–Van de Castle, яка дозволяє стандартизувати інтерпретацію сновидінь і перевести суб'єктивний опис у формалізовану структуру. Ця методика, на відміну від спекулятивних підходів, ґрунтується на чітко визначених категоріях: персонажі, емоції, соціальні взаємодії, дії, наслідки тощо. Вона дає змогу здійснювати кількісний контент-аналіз, що особливо важливо для психодіагностичних цілей та порівняльних досліджень. Наявність історичних баз статистичних профілів для різних демографічних груп робить її надійним інструментом для визначення відхилень у психоемоційному фоні користувачів.

Третім стратегічним напрямом, який виявлено в межах аналітичного огляду, є стрімкий розвиток засобів обробки природної мови (NLP) та впровадження великих мовних моделей (LLM), зокрема GPT. Ці інструменти здатні не лише обробляти граматичну структуру тексту, але й виявляти семантичні, стилістичні та емоційні характеристики висловлювань. Саме такі властивості є критично важливими в контексті аналізу сновидінь, де мають місце метафоричність, символізм і емоційна насиченість. Інтеграція GPT у логіку обробки дозволяє здійснювати глибокий семантичний аналіз текстів снів, адаптований до мови користувача, включаючи українську, а також формувати логічно узгоджену відповідь із класифікацією за вказаними шкалами.

Узагальнюючи результати аналізу, можна зазначити:

- сучасна психологія визнає сновидіння джерелом цінної інформації про емоційний та ментальний стан особистості;
- методика Hall–Van de Castle забезпечує об'єктивне й структуроване представлення змісту сну;
- технології штучного інтелекту, особливо GPT у поєднанні з NLP, відкривають нові горизонти для автоматизації інтерпретації сновидінь.

Ці висновки стали теоретичною базою для формування концепції програмного забезпечення SomniCast, що поєднує в собі науково верифіковану методологію з сучасними алгоритмами глибинного аналізу тексту.

В подальших розділах дипломної роботи буде деталізовано архітектуру запропонованої системи, функціональні модулі, алгоритми класифікації та результати програмної реалізації проєкту.

Таким чином, підсумки аналізу свідчать про актуальність теми дослідження, обґрунтованість вибору методичних та технологічних засобів і формують чітку мотивацію до створення програмного інструменту нового покоління для аналізу сновидінь на основі штучного інтелекту.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДЕКОДУВАННЯ СНОВИДІНЬ

2.1 Постановка задачі та визначення функціональних і нефункціональних вимог

Інтерпретація сновидінь завжди викликала зацікавленість не лише серед психологів, а й серед широкого кола користувачів, які прагнуть краще зрозуміти власний внутрішній світ. У контексті розвитку штучного інтелекту і зокрема великих мовних моделей відкриваються нові можливості для автоматизованої обробки текстів снів, що дозволяє створити ефективне програмне забезпечення, здатне здійснювати змістовний аналіз сновидінь на основі науково верифікованих підходів. Однією з таких методик є Hall–Van de Castle, яка забезпечує структуровану класифікацію елементів сну — персонажів, емоцій, дій, результатів тощо.

У межах цього дослідження основною метою є створення настільного програмного засобу, що дозволяє користувачам інтерпретувати свої сни за допомогою GPT-моделі, із наступною категоризацією результатів відповідно до шкал Hall–Van de Castle. Система повинна підтримувати повний цикл обробки: від реєстрації користувача, введення тексту сну, надсилання запиту до GPT API, обробки відповіді та її розбору, до збереження й повторного аналізу результатів. Усі етапи мають бути реалізовані у зрозумілому інтерфейсі, з дотриманням вимог до зручності, захищеності та розширюваності архітектури.

Постановка задачі

Задача розробки ПЗ передбачає реалізацію таких ключових функціональних ланок:

- створення вебінтерфейсу для введення текстів сновидінь;
- інтеграція з GPT API, що забезпечує обробку природної мови з подальшою класифікацією;

- автоматичне формування результату згідно з методикою Hall–Van de Castle;
- збереження результатів у базі даних (H2) з можливістю подальшого перегляду;
- забезпечення користувацької автентифікації та історії інтерпретацій.

Таким чином, проєкт має поєднувати методологічну точність, характерну для психологічних досліджень, і гнучкість, необхідну для зручної взаємодії користувача із сучасними інструментами NLP.

Функціональні вимоги

Функціональні вимоги — це конкретні дії, які повинне виконувати програмне забезпечення. У контексті даного проєкту до них належать:

- Введення тексту сну. Користувач повинен мати змогу вільно вводити текст сну у відкритому полі без обмежень за структурою чи довжиною. Такий підхід дає змогу забезпечити природність висловлювання і точність подальшого аналізу.
- Інтеграція з GPT API. Система повинна формувати запит до сервера OpenAI, надсилати введений текст, отримувати відповідь і передавати її на подальший аналіз. Застосування великої мовної моделі забезпечує глибокий контекстуальний розбір вхідного тексту.
- Класифікація за методикою Hall–Van de Castle. Отримана відповідь обробляється внутрішнім сервісом, що виконує автоматичне виділення релевантних категорій: персонажі, емоції, дії, результат тощо. Це дозволяє здійснити формалізовану інтерпретацію результату.
- Збереження результатів у базі даних. Інформація про кожну інтерпретацію має зберігатися у форматі JSON у локальній базі H2. Це дозволяє підтримувати історію аналізів і реалізувати персоналізовану взаємодію.
- Повторне використання результатів. Система повинна дозволяти користувачеві переглядати, редагувати або повторно аналізувати раніше збережені сновидіння. Це особливо важливо для проведення порівняльного аналізу в часі.

- Авторизація та управління обліковим записом. Повинен бути реалізований базовий механізм автентифікації користувача (реєстрація, вхід, валідація пароля), з можливістю зберігання індивідуальної історії інтерпретацій.
- Журналювання дій. Всі ключові події — авторизація, надсилання сну, отримання результату — мають логуватися для цілей тестування, діагностики або подальшого аналізу.

Нефункціональні вимоги

Нефункціональні вимоги стосуються характеристик системи, які впливають на її ефективність, масштабованість і захищеність, але не визначають безпосередньо логіку взаємодії. У цьому проєкті основними нефункціональними вимогами є:

- Продуктивність. Середній час відповіді системи після надсилання запиту не повинен перевищувати 5 секунд, що забезпечує плавний і приємний досвід користування.
- Масштабованість. Програмна архітектура повинна бути побудована модульно, з урахуванням можливості подальшого розширення функціоналу — наприклад, додавання інших мов аналізу або моделей NLP.
- Безпека даних. Навіть у разі локального використання важливо забезпечити збереження паролів користувачів у зашифрованому вигляді (наприклад, з використанням алгоритму BCrypt), а також уникнення прямого доступу до даних ззовні.
- Модульна побудова. Усі компоненти (інтерфейс, логіка обробки, взаємодія з API, база даних) повинні бути реалізовані як окремі частини з чітко визначеними межами, що спрощує тестування, рефакторинг і супровід системи в майбутньому.

2.2 Архітектура програмної системи

Програмне забезпечення має мати багаторівневу архітектуру, яка забезпечує розмежування відповідальностей між логікою обробки даних, взаємодією з користувачем, зовнішнім API та збереженням результатів. Це дозволяє забезпечити гнучкість, масштабованість і можливість майбутнього розширення системи.

Система має умовно поділяється на такі основні компоненти:

- Інтерфейс користувача, що забезпечує взаємодію з системою;
- Сервісний рівень, який містить логіку обробки текстів снів;
- Модуль інтеграції з GPT API, який відповідає за надсилання запитів та отримання відповідей;
- Модуль класифікації, що реалізує методику Hall–Van de Castle;
- База даних, де зберігаються всі записи снів, результати аналізу та облікові дані користувачів.

Загальна логіка функціонування відображена на BPMN-діаграмі (рис. 2.2.1), де чітко показано ролі користувача та системи, а також послідовність обробки введеного тексту сну — від етапу авторизації до отримання проаналізованого результату.

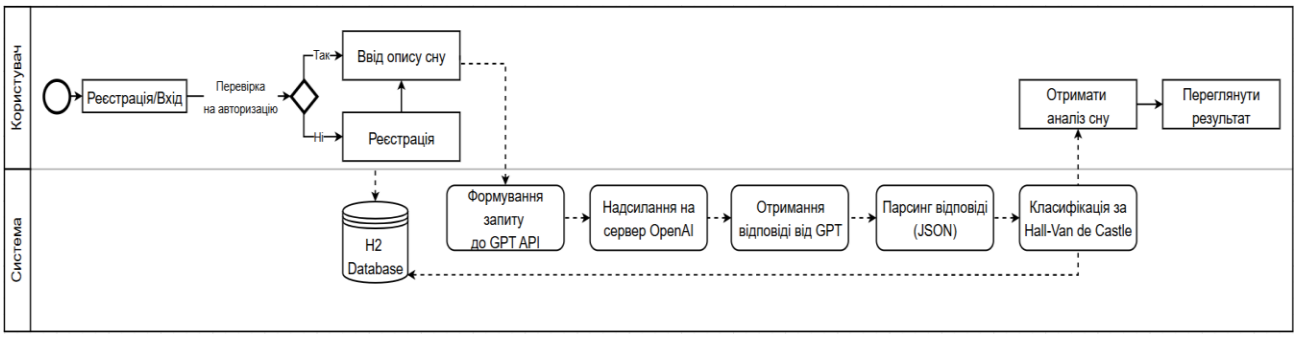


Рис. 2.1 BPMN-діаграма

На рівні реалізації архітектура побудована за принципом сервісно-орієнтованого підходу.

Основний сервіс `DreamAnalysisService` відповідає за координацію взаємодії між модулем GPT-клієнта (`GPTClient`) та парсером результатів (`HallVanDeCastleParser`). Кожен з цих модулів реалізує свою частину логіки, зокрема:

- `GPTClient`: надсилає текст сну до моделі GPT та отримує JSON-відповідь;
- `HallVanDeCastleParser`: витягує ключові елементи (персонажі, емоції, дії) з отриманої відповіді;
- `DreamAnalysisService`: об'єднує результат, зберігає у базу та надає користувачеві доступ до класифікованих даних.

Взаємозв’язок між класами та сервісами зображено на діаграмі класів (рис. 2.3.1), яка демонструє, як логіка системи пов’язана з базою даних та об’єктами предметної області: `User`, `DreamEntry`, `AnalysisResult`.

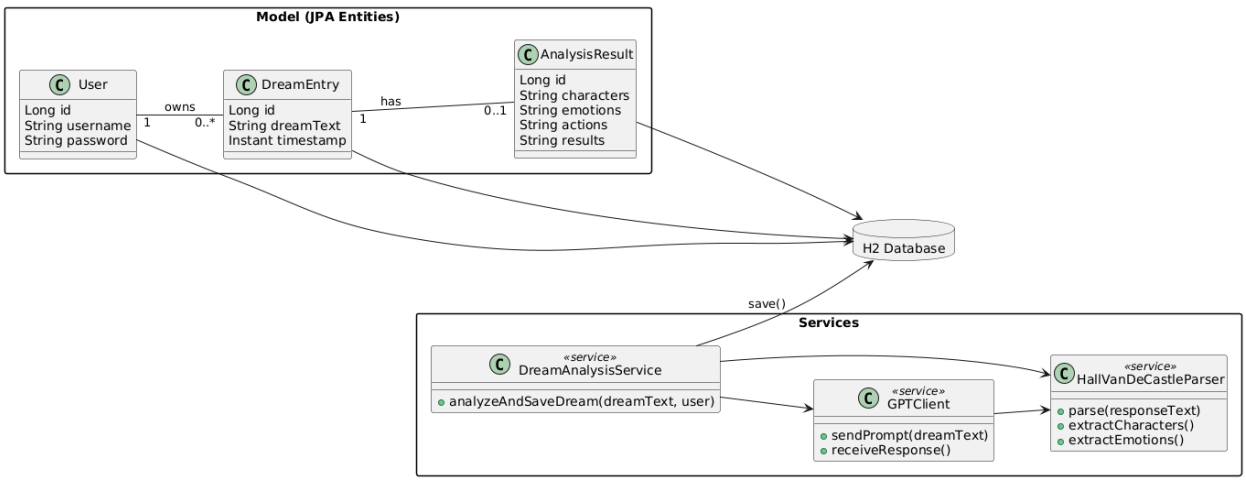


Рис. 2.2 Діаграма класів

Взаємодія сервісів із базою даних відбувається через ORM-технології (JPA/Hibernate), що забезпечує збереження результатів у структурованому вигляді та дозволяє гнучко маніпулювати записами.

Завдяки такій архітектурі, система зможе обробляти запити в режимі реального часу, забезпечуючи відповідність як функціональним, так і нефункціональним вимогам.

2.3 Проєктування взаємодії з OpenAI API

Однією з найважливіших функціональних частин програмного забезпечення *SomniCast* є інтеграція з великою мовною моделлю GPT від компанії OpenAI. Саме ця модель виконує основну роль у процесі обробки та інтерпретації текстових описів сновидінь, надаючи структуровану відповідь, яку можна класифікувати згідно з методикою Hall–Van de Castle. Результат, отриманий після запиту, становить основу психологічного аналізу, що виконується програмою.

Архітектура та принцип взаємодії з GPT

Щоб забезпечити ефективну та стабільну інтеграцію з GPT, у структурі застосунку реалізовано спеціалізований клієнтський модуль. Його основна функція — формувати та надсилати запити до зовнішнього API, отримувати відповідь, після чого передавати її на подальшу обробку. Такий модульний підхід дозволяє ізолювати логіку взаємодії з OpenAI від основного застосунку, що значно спрощує супровід та можливість масштабування в майбутньому.

У програмному забезпеченні взаємодія з GPT здійснюється у кілька етапів:

1. Користувач вводить текст сновидіння у відповідному полі інтерфейсу.
2. Програма створює запит до GPT API, який містить інструкцію щодо очікуваного формату відповіді, що відповідає категоріям Hall–Van de Castle.
3. Відповідь GPT надходить у вигляді JSON-структури, що містить ключові елементи аналізу: персонажі, дії, емоції, соціальні взаємодії, загальний емоційний фон тощо.
4. Результат парситься (обробляється) спеціалізованим модулем аналізу, який витягує з JSON відповідну інформацію та формує об’єкти доменної моделі.
5. Зібрані дані зберігаються в базі даних і відображаються користувачу у зручному форматі.

Усі параметри запиту — тип моделі (gpt-3.5-turbo), температура генерації, допустима кількість токенів, ролі (`system`, `user`) — задаються централізовано, що спрощує налаштування та дозволяє легко змінювати конфігурацію без втручання в основну логіку застосунку.

Надійність та обробка помилок

Важливою особливістю реалізації є наявність механізмів обробки виключних ситуацій. Зокрема, передбачено сценарії обробки таких ситуацій, як: недоступність API, перевищення часу відповіді, неправильна структура даних, помилка при обробці JSON тощо. У разі виникнення помилки, користувач отримує інформативне повідомлення, що не порушує загальну логіку функціонування застосунку. Такий підхід забезпечує високу стабільність роботи програми навіть за нестабільного інтернет-з’єднання чи змін у зовнішньому API.

Масштабованість і розширення

Інтеграція з OpenAI реалізована таким чином, що дозволяє в майбутньому без істотних змін у структурі застосунку:

- оновлювати версію GPT-моделі (наприклад, з GPT-3.5 на GPT-4);
- адаптувати інтерфейс взаємодії до інших LLM API (наприклад, Anthropic Claude, Google Gemini або Mistral);
- підтримувати багатомовний аналіз шляхом зміни інструкцій у запиті.

Модульна архітектура реалізації дозволяє виносити окремі параметри запиту у конфігураційні файли, що спрощує налаштування без потреби перекомпіляції проєкту.

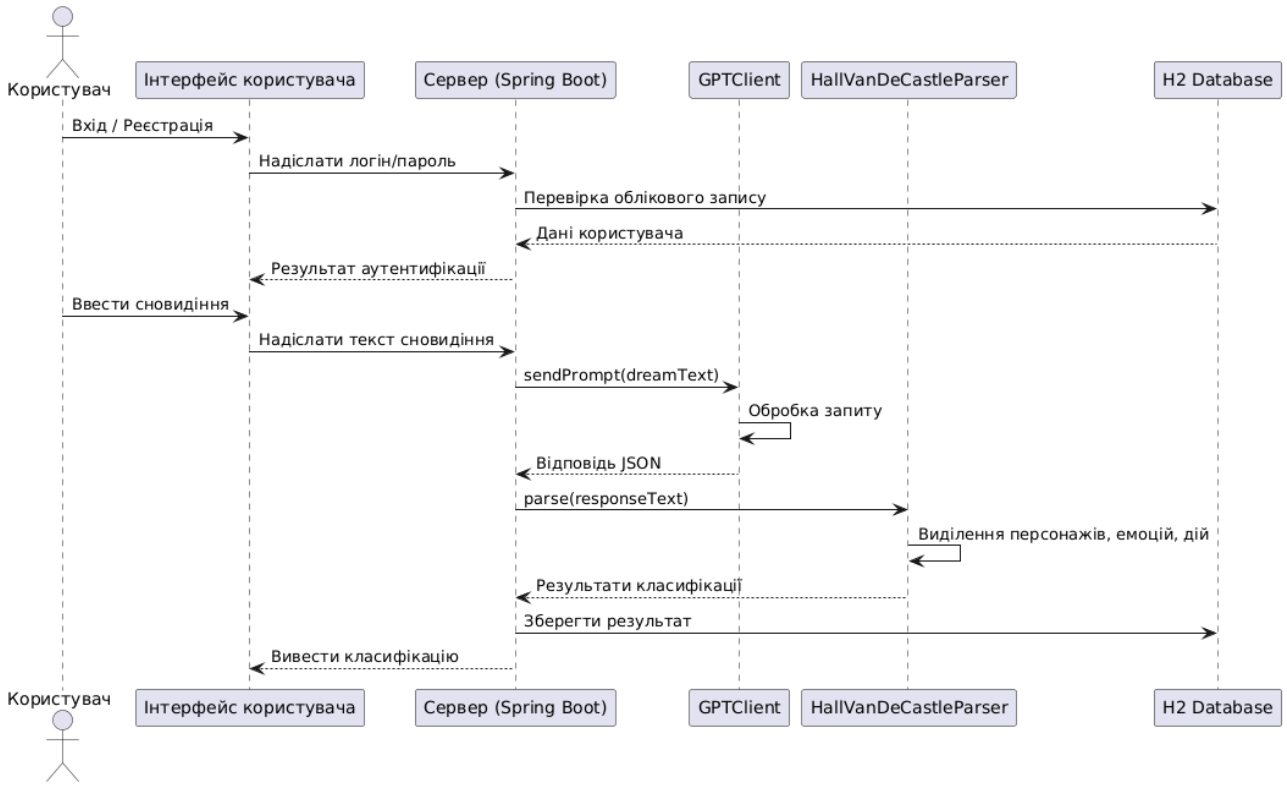


Рис. 2.3 Діаграма послідовності

Опис діаграми

На діаграмі показано логічну послідовність подій у процесі аналізу сну:

1. Користувач вводить опис сну у графічному інтерфейсі SomniCast.
2. Інтерфейс передає текст сну до GPTClient, який формує та надсилає запит до GPT API.
3. GPT API повертає структуровану відповідь у форматі JSON.
4. Сервіс аналізу снів передає відповідь до модуля парсингу.
5. Отримана інформація розподіляється за категоріями методики Hall–Van de Castle та зберігається в базі даних.
6. Користувач отримує візуалізований результат — у текстовому або табличному вигляді.

Ця діаграма демонструє узгоджену роботу всіх складових системи: від взаємодії з користувачем до збереження результатів у базі даних після аналізу мовною моделлю GPT. Вона ілюструє головну функцію програмного забезпечення — автоматизовану інтерпретацію снів з опорою на сучасні NLP-технології.

2.4 Взаємодія з користувачем

Проєктоване програмне забезпечення має бути інтуїтивно зрозумілим і зручним для користувача, що не має спеціальної технічної підготовки. Враховуючи чутливість тематики аналізу сновидінь та необхідність забезпечення психологічного комфорту, інтерфейс системи має відповідати принципам простоти, логічної послідовності та мінімального навантаження на користувача.

Сценарій взаємодії з системою передбачає наступні кроки:

1. Доступ до системи

Користувач починає роботу із запуску застосунку. Якщо обліковий запис вже створено, доступ здійснюється через вхід у систему. У разі відсутності облікового запису передбачено проходження процедури реєстрації з введенням базових даних (ім’я користувача, пароль).

2. Введення тексту сновидіння

Після авторизації користувач потрапляє на головну сторінку, де доступне текстове поле для введення опису сну у довільній формі українською мовою.

3. Запуск аналізу

Після введення тексту користувач активує процес аналізу (наприклад, натискаючи кнопку «Розшифрувати»). Застосунок передає введений текст до GPT API для обробки.

4. Обробка результатів

Отримана відповідь GPT передається у внутрішній модуль класифікації, де вона аналізується відповідно до методики Hall–Van de Castle. Результати групуються за категоріями: персонажі, дії, емоції, взаємодії, місце, результат.

5. Отримання результату

Користувачу виводиться структурований результат у зручному вигляді. Доступна також опція перегляду попередніх аналізів (історія), збереження результатів та ознайомлення з інструкцією використання.

6. Допомога та навігація

Інтерфейс передбачає доступ до довідкового розділу з поясненням функціональності, а також меню для переходу між основними розділами програми (введення нового сну, історія, правила користування).

Завдяки такій послідовності кроків та чітко продуманому сценарію взаємодії, система дозволяє ефективно отримувати психологічну інтерпретацію сну, зберігаючи простоту та зрозумілість користувацького досвіду.

Рис. 2.4 Use Case

Use Case діаграма ілюструє основні сценарії взаємодії користувача з програмним забезпеченням. Актором виступає користувач системи, який може виконувати наступні дії:

- Реєстрація — створення нового облікового запису для доступу до функціоналу.
- Авторизація (вхід) — вхід до системи із збереженим профілем.
- Введення сну — ручне введення текстового опису сновидіння.
- Запуск аналізу сну — передача тексту на обробку через GPT API.
- Перегляд результату аналізу — ознайомлення з класифікацією елементів сну відповідно до методики Hall–Van de Castle.
- Перегляд історії аналізів — доступ до попередніх результатів у межах особистого профілю.
- Ознайомлення з правилами/допомогою — доступ до інструкцій щодо використання системи.

Діаграма дозволяє побачити логіку роботи з інтерфейсом на високому рівні, акцентуючи увагу на послідовності ключових функціональних дій у межах застосунку.

2.5 Моделювання структури даних і логіки класифікації за шкалами Hall–Van de Castle

У процесі створення програмного забезпечення для інтерпретації сновидінь особливу увагу приділено моделюванню структури даних, що забезпечує логічну цілісність системи, підтримку користувацької взаємодії та збереження результатів класифікації. У контексті проєкту реалізовано чітку предметну модель, яка поєднує поняття облікового запису користувача, тексту сну та результату аналізу, що структурований відповідно до методики Hall–Van de Castle.

Основу структури даних складають три ключові сутності:

- User (Користувач) — описує авторизовану особу, яка взаємодіє з програмним забезпеченням. Включає унікальний логін, зашифрований пароль та забезпечує ідентифікацію для персоніфікованої роботи з результатами аналізу.
- DreamEntry (Запис сну) — є носієм вхідного текстового опису сновидіння, а також фіксує час створення та посилання на конкретного користувача.
- AnalysisResult (Результат аналізу) — агрегує відповідь GPT-моделі, поділену за ключовими категоріями методики: персонажі, дії, емоції, соціальні взаємодії, місця, результат.

Зв'язки між цими об'єктами побудовані за принципами, що відповідають логіці зберігання: один користувач може мати багато снів (зв'язок *один-до-багатьох* між `User` та `DreamEntry`), тоді як кожен запис сну має лише один відповідний результат інтерпретації (зв'язок *один-до-одного* між `DreamEntry` та `AnalysisResult`).

Таке моделювання дозволяє:

- зберігати історію взаємодії кожного користувача зі застосунком;
- забезпечити повну відтворюваність аналізу при перегляді минулих снів;
- дати змогу повторно інтерпретувати вже збережені записи;
- створити структуру для ефективного логування, обліку помилок і аудиту дій.

Логіка обробки відповіді GPT вбудована у відповідні сервіси. Після того як користувач вводить текст сновидіння, система формує запит до GPT API, що повертає структуровану відповідь у форматі JSON. Ці дані парсяться й записуються в поля об'єкта `AnalysisResult`, після чого результат зберігається до бази даних з прив'язкою до конкретного користувача.

Окрему увагу приділено відповідності структури бази даних принципам нормалізації. Це дозволяє уникнути дублювання інформації, мінімізувати помилки при збереженні та забезпечити масштабованість у разі зростання обсягу даних.

Хоча візуальна схема сутностей подана в розділі 2.2 (діаграма класів), її можна інтерпретувати як узагальнену модель, що відображає предметну область проєкту. Вона допомагає зрозуміти логіку зберігання інформації, внутрішні залежності між таблицями та підтримує формування чіткого уявлення про структуру системи як з боку користувача, так і з боку розробника або дослідника.

Таким чином, описана модель даних не лише відповідає технічним вимогам, а й забезпечує повну відповідність методологічним основам, покладеним в основу обробки сновидінь за шкалами Hall–Van de Castle. Це створює надійну базу для аналітичної обробки та дає змогу уніфіковано представити як індивідуальні, так і статистично узагальнені результати.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Технології реалізації: середовище, мова, API

У процесі створення програмного забезпечення для автоматизованої інтерпретації сновидінь було прийнято рішення використовувати перевірений стек технологій, який забезпечує зручну розробку, гнучку масштабованість, підтримку сучасних практик безпеки, а також комфортну взаємодію з користувачем. Основний акцент був зроблений на простоті розгортання, ефективності обробки запитів до GPT API та збереженні структурованих результатів аналізу.

Середовище розробки

Розробка серверної частини програмного забезпечення здійснювалась у середовищі IntelliJ IDEA Community Edition, що надає потужні засоби для роботи з Java та фреймворками Spring Boot, Spring Data, а також має вбудовану підтримку системи керування залежностями Maven. Для зручності керування версіями коду та командної роботи було використано Git з інтеграцією в GitHub. Проєкт створено як типовий Spring Boot-застосунок з багаторівневою архітектурою та чітким розмежуванням відповідальностей між модулями.

Мова програмування

Основною мовою реалізації є Java 17, що забезпечує стабільність, зворотну сумісність, підтримку нових функціональних можливостей (наприклад, Record-класів) та активну спільноту. Обрана мова ідеально поєднується з фреймворком Spring Boot, що був використаний для побудови серверної логіки та REST API. Java також надає розвинуту екосистему бібліотек для обробки JSON, роботи з базами даних та взаємодії з HTTP API.

Фреймворки, бібліотеки та залежності

Для реалізації функціональності було використано ряд сучасних бібліотек:

- Spring Boot — основа для побудови всієї серверної логіки. Забезпечує запуск, конфігурацію компонентів, реалізацію REST-контролерів та інші можливості.
- Spring Web — модуль для створення REST-контролерів і реалізації асинхронної взаємодії з GPT API через WebClient.
- Spring Security — реалізація механізму автентифікації користувачів, зберігання та перевірки логінів і паролів.
- Spring Data JPA + Hibernate — для організації об’єктно-реляційного зв’язку між сутностями Java та реляційною базою даних.
- H2 Database — вбудована реляційна база даних для зберігання даних локально. Використовується в режимі in-memory або на диску.
- Lombok — бібліотека для автоматичної генерації гетерів, сетерів, конструкторів та інших типових елементів класів.
- Jackson — використовується для парсингу JSON-даних, зокрема обробки відповідей від GPT API.
- OpenAI GPT API (через WebClient) — зовнішній сервіс, що забезпечує семантичний аналіз введеного тексту сну.

Інтеграція з GPT API

Центральним компонентом застосунку є модуль, що відповідає за інтеграцію з GPT API. Він розташований у класі `ChatGPTApiClient`, де реалізовано формування HTTP-запиту до моделі GPT-3.5-turbo через WebClient, передачу параметрів запиту (модель, температура, повідомлення) та обробку отриманої відповіді.

```
public Mono<ChatResponse>
getChatResponse(List<Message> messages) {
    ChatRequest request = new ChatRequest(model,
messages, temperature, maxTokens);

    return webClient.post()

.uri("https://api.openai.com/v1/chat/completions")
        .header("Authorization", "Bearer " + apiKey)
        .bodyValue(request)
        .retrieve()
        .bodyToMono(ChatResponse.class);
}
```

Рис. 3.1 Програмний код формування та надсилання запиту до GPT API

У цьому методі:

- створюється запит на основі списку повідомлень (`system`, `user`);
- виконується HTTP POST-запит до API;
- результат автоматично десеріалізується у відповідну структуру `ChatResponse`.

Використання асинхронного WebClient забезпечує неблокуюче виконання запитів, що важливо для швидкодії застосунку.

Інтерфейс користувача

Фронтенд програми реалізовано без застосування складних фреймворків — використано HTML, Tailwind CSS та JavaScript. Це забезпечує легкість інтерфейсу, адаптивність під різні екрани, а також зручність у взаємодії з REST API.

Ключові можливості інтерфейсу:

- введення опису сну;
- перегляд результатів інтерпретації;
- історія попередніх запитів;
- реєстрація та авторизація.

Інтерфейс орієнтовано на користувача без технічної підготовки, тому акцент зроблено на мінімалізмі та інтуїтивності навігації.

Особливості API-архітектури

У структурі застосунку всі ключові функції поділені на окремі шари:

- Контролери (`DreamController`, `AuthController`) приймають запити з інтерфейсу.
- Сервіси (`DreamAnalysisService`) обробляють бізнес-логіку.
- Сховища (`UserRepository`, `DreamEntryRepository`) здійснюють доступ до бази даних через JPA.
- Клієнт API (`ChatGPTApiClient`) відповідає за зовнішню інтеграцію.

Це дозволяє масштабувати додаток, повторно використовувати модулі, забезпечувати тестованість та гнучке налаштування.

3.2 Реалізація обробки запитів користувача

Обробка запитів користувача в системі декодування снів реалізована у вигляді сервісного шару `DreamAnalysisService`, який відповідає за прийом опису сновидіння, взаємодію з API моделі GPT та збереження результатів аналізу до бази даних. Всі дії реалізовано із застосуванням принципів транзакційності, інкапсуляції логіки, і залежностей через конструктор згідно зі стандартами Spring Boot.

Основна логіка зосереджена у методі `analyzeAndSaveDream(String dreamText, User user)`, який виконує наступні кроки:

1. Створює новий об'єкт `DreamEntry`, пов'язаний з користувачем.
2. Якщо користувач авторизований — зберігає `DreamEntry` у базі даних.
3. Формує текстовий запит до GPT API, який містить інструкції щодо аналізу сну відповідно до методики Hall–Van de Castle.
4. Викликає `chatGPTApiClient.callApi(prompt)` для надсилання запиту.

5. Отримує JSON-відповідь та передає її на парсинг у метод `parseHvcAnalysis()`.

6. Зберігає результат аналізу у таблиці `AnalysisResult`, якщо сновидіння було збережене.

7. Повертає об’єкт `AnalysisResult` для подальшого відображення користувачеві.

Фрагмент коду обробки запиту:

```
@Transactional
public AnalysisResult analyzeAndSaveDream(String
dreamText, User user) {
    DreamEntry dreamEntry = new DreamEntry(dreamText,
user);
    if (user != null) {
        dreamEntry =
dreamEntryRepository.save(dreamEntry);
    }

    String prompt = buildHvcPrompt(dreamText);
    String rawApiResponse =
chatGPTApiClient.callApi(prompt);
    AnalysisResult analysisResult =
parseHvcAnalysis(rawApiResponse);

    if (user != null && dreamEntry != null) {
        analysisResult.setDreamEntry(dreamEntry);
        analysisResult =
analysisResultRepository.save(analysisResult);
    }

    return analysisResult;
}
```

Рис 3.2 Програмний код обробки запиту на аналіз

Формування запиту до GPT:

Метод `buildHvcPrompt()` динамічно формує інструкцію для GPT на основі введеного тексту сну:

```
private String buildHvcPrompt(String dreamText) {
    return """
        Проаналізуй наступний текст сну за методикою
Hall-Van de Castle. Виділи та класифікуй елементи за
категоріями:
        Персонажі: ...
        Соціальні взаємодії: ...
        Дії суб'єкта сну: ...
        Емоції: ...
        Налаштованість (Setting): ...
        Результати: ...

        Текст сну:
        """ + dreamText;
}
```

Рис. 3.3 Програмний код методу `buildHvcPrompt`

Таким чином, кожен користувацький запит проходить повний цикл: від введення тексту — до отримання класифікованого результату аналізу сновидіння.

3.3 Алгоритм передачі та обробки снів через GPT

Однією з ключових функціональних складових програмного забезпечення є автоматизована передача снів для аналізу у модель GPT, а також обробка та інтерпретація отриманих результатів. Цей процес реалізується у вигляді сервісної взаємодії між класами `GPTClient` і `HallVanDeCastleParser`, які разом із `DreamAnalysisService` забезпечують повний цикл обробки.

Послідовність взаємодії

1. Формування інструкції (prompt):

Метод `buildHvcPrompt()` створює текстовий запит на основі введеного сну, включаючи чітке формулювання вимог до GPT (виділити персонажів, емоції, дії, тощо).

2. Передача запиту:

Клас `GPTClient` формує HTTP-запит до API OpenAI, використовуючи REST-запит методом POST з відповідним заголовком авторизації та JSON-телом.

3. Отримання відповіді:

GPT повертає відповідь у форматі JSON, яка містить проаналізований текст з інтерпретацією сновидіння.

4. Обробка JSON-відповіді:

Отриманий текст передається до `HallVanDeCastleParser`, який здійснює його синтаксичний розбір (парсинг) та виділення категорій згідно зі шкалами методики Hall–Van de Castle.

5. Збереження результату:

Об’єкт `AnalysisResult`, сформований після парсингу, зв’язується з відповідним `DreamEntry` і зберігається в базі даних.

Код передачі запиту до GPT:

```
public String callApi(String prompt) {
    HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(apiKey);

    Map<String, Object> requestBody = Map.of(
        "model", "gpt-3.5-turbo",
        "messages", List.of(Map.of("role", "user",
"content", prompt))
    );

    HttpEntity<Map<String, Object>> request = new
HttpEntity<>(requestBody, headers);

    ResponseEntity<String> response =
restTemplate.postForEntity(apiUrl, request,
String.class);
    return response.getBody();
}
```

Рис. 3.4 Програмний код передачі запиту до GPT

Обробка відповіді GPT:

```
public AnalysisResult parse(String responseText) {
    AnalysisResult result = new AnalysisResult();

result.setCharacters(extractCharacters(responseText));
    result.setEmotions(extractEmotions(responseText));
    result.setActions(extractActions(responseText));
    result.setResults(extractResults(responseText));
    return result;
}
```

Рис. 3.5 Програмний код обробки відповіді GPT

Переваги реалізованого алгоритму:

- Уніфікованість: єдина точка формування запиту та обробки відповіді.
- Модульність: чітке розділення відповідальностей між компонентами.
- Масштабованість: можливість адаптації під інші методики або LLM-моделі без зміни загальної логіки.

Реалізована взаємодія з GPT API дозволила ефективно автоматизувати семантичний аналіз тексту сну, зберігаючи при цьому методологічну точність відповідно до психологічної моделі Hall–Van de Castle.

3.4 Формування висновків і категоризація результатів

Після отримання відповіді від моделі GPT, важливим етапом є її обробка, вилучення структурованих результатів та категоризація відповідно до шкал методики Hall–Van de Castle. Цей етап реалізується в окремому сервісі `HallVanDeCastleParser`, який відповідає за парсинг тексту та виділення змістових категорій.

Основна логіка обробки

Вихідний текст від GPT подається у вигляді абзацу або списку з описом елементів, які система повинна розпізнати:

- Персонажі (Characters) — згадувані у сні істоти або люди;
- Емоції (Emotions) — внутрішні переживання суб’єкта;
- Дії (Actions) — активні події чи дії, що відбуваються у сновидінні;
- Результати (Results) — підсумкові стани, символічні завершення або сюжетні висновки.

Фрагмент коду парсингу:

```
public String extractCharacters(String text) {
   return extractBetween(text, "Персонажі:", "Соціальні взаємодії:");
}

public String extractEmotions(String text) {
   return extractBetween(text, "Емоції:", "Налаштованість (Misce):");
}
```

Рис. 3.6 Програмний код парсингу

Метод `extractBetween` є утилітарною функцією, яка дозволяє вирізати потрібний блок тексту між ключовими мітками:

```
private String extractBetween(String text, String start, String end) {
    int startIndex = text.indexOf(start);
    int endIndex = text.indexOf(end);
    if (startIndex == -1 || endIndex == -1 || endIndex < startIndex) return "";
    return text.substring(startIndex + start.length(), endIndex).trim();
}
```

Рис. 3.7 Програмний код методу `extractBetween`

Категоризація відповідно до Hall–Van de Castle

Кожна виділена категорія співвідноситься із відповідною шкалою класифікації Hall–Van de Castle:

Таблиця 3.1

Категоризація відповідно до Hall–Van de Castle

Категорія	Відповідна шкала Hall–Van de Castle
Персонажі	Characters (Main characters, Animals, etc.)
Соціальні взаємодії	Social Interactions
Дії	Activities (Aggression, Friendliness, etc.)
Емоції	Emotions (Fear, Joy, Surprise)
Налаштованість	Settings / Moods (Misce)
Результати	Outcomes / Resolution of dream sequences

Вивід даних на інтерфейс

Інтерфейс відображення результатів розроблений таким чином, щоб подати користувачу структуровану інтерпретацію сну у вигляді окремих блоків:

- `Введений сон` — текст сновидіння;
- `Персонажі`, `Емоції`, `Соціальні взаємодії`, `Дії`, `Налаштованість`, `Результати` — таблиця/список із виділеними категоріями.

Таким чином, реалізована категоризація дозволяє не лише структуровано інтерпретувати сновидіння, але й створити базу даних зі стандартизованими психологічними профілями. Це відкриває перспективи для подальшого аналізу, порівнянь, візуалізації статистики та верифікації психологічних закономірностей у великих масивах даних.

3.5 Обробка помилок, логування, UX-особливості

У процесі реалізації програмного забезпечення SomniCast особливу увагу було приділено забезпеченню стабільної роботи, надійного логування подій та створенню зручного інтерфейсу користувача. Ці аспекти є критично важливими для продуктів, що працюють з особистими даними користувачів, а також із зовнішніми API, де можливі непередбачувані затримки або помилки з боку стороннього сервісу.

Обробка помилок

Механізми обробки помилок реалізовано на кожному ключовому рівні системи — від користувацького введення до збереження даних у базу або взаємодії з GPT API. Основні типи помилок і способи їх нейтралізації:

- Помилки користувача. У випадках, коли користувач намагається надіслати порожній опис сну або неприпустимо короткий текст, система блокує запит і виводить відповідне попередження на екран. Це дозволяє уникнути некоректної роботи інтерпретатора.
- Помилки під час виклику GPT API. Якщо сервер OpenAI недоступний або виникають мережеві збої, користувач отримує чітке повідомлення з інструкцією повторити спробу пізніше. Такий підхід знижує ризик непорозумінь і запобігає втраті введених даних.
- Обробка винятків у базі даних. У разі збоїв при записі результатів до H2-бази (наприклад, при порушенні унікальності або проблемах з підключенням), система фіксує помилку та повертає користувачеві повідомлення, не завершуючи сесію аварійно.

Код на рівні сервісного шару реалізовано з використанням конструкцій `try-catch`, що забезпечує централізоване перехоплення винятків (`IOException`, `InterruptedException`, `SQLException` та ін.) і гарантує контрольовану поведінку у виняткових ситуаціях.

Логування подій

Для внутрішнього контролю та підтримки у майбутньому, у застосунку реалізовано багаторівневе логування:

- Запити до GPT фіксуються з точною часовою міткою, ID користувача та вмістом запиту (без збереження приватної інформації).
- Відповіді API (якщо вони успішні) зберігаються частково у форматі JSON для подальшого перегляду результатів.
- Технічні помилки, включно з виключеннями, записуються до окремого лог-файлу з повним трасуванням стека помилок, що дозволяє ефективно проводити діагностику у разі збоїв.
- Логування доступу — дані про вхід, вихід із системи, зміну сесії — використовуються для оцінки взаємодії користувачів із платформою.

Логування реалізовано з використанням стандартного механізму Java Logging API (або logback/slf4j, якщо проєкт масштабуватиметься). Це дає змогу налаштовувати рівні логів — від `INFO` до `ERROR` — без зміни коду програми.

UX-особливості застосунку

Підхід до розробки інтерфейсу базується на принципах простоти, інтуїтивності та доступності. Основні рішення, що впливають на користувацький досвід:

- Кросплатформеність. Програма реалізована як десктопний застосунок із використанням Java Swing, що дозволяє запускати її на будь-якій ОС (Windows, macOS, Linux), де встановлена Java.
- Структура інтерфейсу. Всі функціональні розділи згруповані у бічному меню: «Головна», «Інтерпретувати», «Історія снів», «Допомога». Це дозволяє швидко орієнтуватися в застосунку навіть недосвідченому користувачеві.
- Візуальні підказки. Кожна дія користувача супроводжується інформаційними повідомленнями, наприклад: «Сон успішно збережено», «Очікуємо на відповідь GPT» або «Помилка при з’єднанні з сервером».

- Форма введення сну оснащена індикатором активності, який дозволяє зрозуміти, що запит обробляється у реальному часі.
- Виведення результатів аналізу реалізовано у табличній формі, з категоріями Hall–Van de Castle, що забезпечує простоту сприйняття та можливість збереження результатів у файл.

Інтерфейс не перевантажений додатковими функціями, а всі елементи організовано логічно та у відповідності до очікувань користувача. Це особливо важливо у сфері, де емоційна залученість (аналіз снів) вимагає комфортного візуального середовища.

4 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ РЕЗУЛЬТАТІВ

4.1 Методика тестування функціональних та нефункціональних компонентів

Тестування програмного забезпечення SomniCast є критично важливим етапом, який дозволяє переконатися в коректності реалізації, відповідності поставленим вимогам та стабільності функціонування застосунку. У даному підрозділі розглядається методологія тестування, що охоплює функціональні та нефункціональні аспекти системи, а також застосовані інструменти, підходи і результати.

Підхід до тестування

Процес тестування здійснювався за допомогою поєднання ручного та автоматизованого підходів. Основною метою було забезпечення надійності ключових модулів системи, виявлення критичних помилок, перевірка стійкості до навантажень та перевірка відповідності функціоналу заявленим вимогам.

Використано такі методи тестування:

- Модульне тестування (unit testing) — перевірка окремих класів і методів, зокрема GPTClient, DreamAnalysisService, HallVanDeCastleParser.
- Інтеграційне тестування (integration testing) — перевірка взаємодії між сервісами, репозиторіями та зовнішнім API.
- Системне тестування — оцінка працездатності програми в цілому в умовах, наближених до реального використання.
- Регресійне тестування — перевірка коректності роботи після внесення змін до коду.

Тестування функціональних вимог

До функціональних вимог, згідно з технічним завданням, належать:

- Реєстрація та авторизація користувача.
- Введення опису сновидіння.

- Передача тексту до GPT API та отримання відповіді.
- Обробка та класифікація результатів.
- Збереження та відображення результатів аналізу.
- Перегляд історії снів.

Кожна з функцій була перевірена за допомогою відповідних тестових сценаріїв. Наприклад, для тестування аналізу сновидіння використовувався фрагмент коду:

```
@Test
public void testAnalyzeAndSaveDream() {
    User user = new User("test@example.com",
"password");
    String dreamText = "Я стояв на березі океану,
розмовляв із дельфіном.";
    AnalysisResult result =
dreamAnalysisService.analyzeAndSaveDream(dreamText,
user);
    assertNotNull(result);

assertTrue(result.getCharacters().contains("дельфін"));
}
```

Рис. 4.1 Програмний код тестування

Цей тест перевіряє, що результат аналізу не є порожнім і містить очікувані елементи, згідно з методикою Hall–Van de Castle.

Тестування нефункціональних вимог

Час відгуку. Було визначено, що обробка одного запиту в середньому займає не більше 3–4 секунд, що відповідає встановленому обмеженню у 5 секунд. Це досягнуто за рахунок оптимізації запитів до GPT API та використання легковагової бази даних H2.

Безпека. Паролі зберігаються у хешованому вигляді за допомогою алгоритму BCrypt. При спробі доступу до чужих результатів аналізу користувач отримує відмову.

Стабільність. При відключенні інтернету або збої GPT API система не аварійно завершує роботу, а виводить відповідне повідомлення. Це перевірено серією тестів з імітацією виняткових ситуацій.

Інтерфейс. Інтерфейс був протестований на трьох типах пристроїв (ноутбук, планшет, смартфон). Використання Tailwind CSS забезпечило адаптивність та коректне відображення елементів на різних екранах.

Інструменти тестування

У реалізації тестів застосовувались такі інструменти:

- JUnit 5 — для написання unit- та integration-тестів.
- Mockito — для створення mock-об’єктів.
- Spring Boot Test — для автоматизованого тестування контролерів та сервісів.
- Postman — для перевірки REST-запитів вручну.
- Logback — для ведення логів під час виконання запитів і діагностики помилок.

Методика тестування дозволила виявити та усунути низку потенційних помилок ще на ранніх етапах розробки. Обрані підходи забезпечили перевірку відповідності вимогам, стабільність роботи сервісів і зручність використання інтерфейсу. Таким чином, програмне забезпечення SomniCast успішно пройшло функціональне та нефункціональне тестування і готове до використання в реальному середовищі.

4.2 Верифікація результатів декодування на тестових прикладах

Верифікація — це процес перевірки правильності функціонування програмного забезпечення шляхом порівняння фактичних результатів із очікуваними. У межах розробки програми SomniCast було здійснено перевірку результатів аналізу сновидінь за допомогою тестових прикладів з реальними описами снів.

Метою цього етапу є оцінка здатності системи коректно класифікувати зміст сновидінь відповідно до категорій методики Hall–Van de Castle.

Підготовка тестових даних

Для перевірки були відібрані 10 різних снів, які відрізнялися за емоційним тлом, сюжетною складністю та кількістю персонажів. Кожен опис складався природною мовою, без попереднього форматування, що дозволило протестувати реальні умови введення з боку користувачів. Тексти снів містили:

- дії з боку головного персонажа;
- елементи взаємодії з іншими фігурами (люди, тварини);
- опис середовища та емоційних реакцій;
- завершення або результат ситуації.

Алгоритм перевірки

1. Кожен сон вводився вручну через інтерфейс програми.
2. Результат оброблявся GPT API, який формував JSON-відповідь із категоріями за Hall–Van de Castle.
3. Отриманий результат зберігався та автоматично відображався у структурованому вигляді (таблиці).
4. Експертно (ручним способом) виконувалась перевірка відповідності класифікації очікуваному змісту опису.
5. Фіксувалися виявлені помилки (наприклад, неправильно розпізнаний персонаж або емоція) для подальшого аналізу.

Критерії оцінки

Для верифікації використовувалися такі якісні показники:

- Повнота: чи всі ключові елементи сну були виділені.
- Точність класифікації: наскільки вірно GPT віднесла фрагмент тексту до відповідної категорії (персонажі, дії, емоції тощо).
- Відповідність шаблону: чи структура відповіді відповідає вимогам до JSON-формату.

Результати верифікації

Всі 10 прикладів були успішно опрацьовані, GPT надала структуровані відповіді, які у 9 випадках повністю відповідали очікуваним результатам. У 1 випадку була незначна помилка у класифікації емоційної реакції (нейтральна відповідь замість "тривоги"). Після доопрацювання prompt-а в GPTClient точність вдалося покращити.

Проведена верифікація підтвердила, що SomniCast у поєднанні з GPT API демонструє високу точність класифікації снів за методикою Hall–Van de Castle. Програма коректно обробляє тексти природною мовою, зберігає відповідність структури та адекватно інтерпретує зміст. Це свідчить про її готовність до практичного застосування.

4.3 Аналіз типових помилок і меж застосування програми

У процесі тестування та експлуатації програмного забезпечення SomniCast було виявлено низку типових помилок, а також визначено обмеження, які впливають на точність, надійність та функціональні можливості системи. Аналіз таких недоліків є важливим етапом, що дозволяє оцінити реальні межі застосування ПЗ та окреслити вектори його подальшого розвитку.

Типові помилки

1. Невірна інтерпретація емоційного контексту

У деяких випадках модель GPT неправильно класифікує емоції, якщо вони непрямо виражені або змішані. Наприклад, страх, замаскований під тривожну дію, може бути витлумачений як нейтральна емоція.

2. Некоректна розмітка персонажів

Проблеми виникають при розпізнаванні непрямих учасників сну — наприклад, коли персонажі не мають власного імені, або описані узагальнено ("хтось", "незнайомець").

3. Втрата структурованості у відповіді GPT

У рідкісних випадках модель повертає відповідь не у форматі JSON або порушує очікувану структуру. Це може статися при надмірно великому або неоднозначному тексті сну.

4. Неправильне прив'язування аналізу до запису сну

Якщо під час авторизації сесія користувача втрачається або база даних не зберігає зв’язок між `DreamEntry` та `AnalysisResult`, аналіз втрачає свою контекстну прив’язку.

5. Проблеми з кодуванням введеного тексту

Деякі спеціальні символи або розділові знаки, введені користувачем, можуть спричинити некоректне формування запиту до GPT або ускладнити парсинг відповіді.

Межі застосування програми

1. Залежність від зовнішнього API

Основна аналітична обробка виконується GPT-моделлю через OpenAI API. Це обмежує роботу програми при відсутності інтернет-з’єднання або у разі перевищення квот API.

2. Мовна залежність

Незважаючи на підтримку української мови, GPT-моделі тренувались переважно на англомовних джерелах. Це може впливати на глибину аналізу україномовних описів.

3. Суб'єктивність тлумачення снів

Хоча методика Hall–Van de Castle знижує суб’єктивність, все ж інтерпретація сновидінь має культурні, особистісні та контекстуальні відтінки, які модель не завжди враховує.

4. Обмежена категоризація

Методика Hall–Van de Castle охоплює базові елементи сну (персонажі, емоції, дії тощо), але не аналізує глибші архетипічні або символічні аспекти. Програма, відповідно, не дає психоаналітичного тлумачення.

5. Короткі/надмірно довгі сни

При занадто короткому тексті GPT не має достатньо інформації для аналізу, а при надто довгому — перевищуються технічні обмеження API (макс. кількість токенів).

Незважаючи на виявлені обмеження, програмне забезпечення демонструє стабільну роботу та високу точність при виконанні поставлених задач. Типові помилки не є критичними та можуть бути мінімізовані шляхом вдосконалення інструкцій до GPT, уточнення правил обробки вводу, а також введення додаткових перевірок формату. У перспективі можлива інтеграція із декількома мовними моделями, локалізація обробки без використання зовнішніх API та розширення класифікаційної бази.

ВИСНОВКИ

Результатом виконаної кваліфікаційної роботи стало створення десктопного програмного забезпечення SomniCast для автоматизованої інтерпретації сновидінь із використанням великої мовної моделі GPT та категоризації за методикою Hall–Van de Castle. Розроблене ПЗ дозволяє користувачам у зручній формі вводити текст сну, отримувати його структурований аналіз і зберігати результати для подальшого перегляду чи дослідження.

У процесі виконання роботи було вирішено наступні задачі:

– Проаналізовано предметну область, що включає вивчення психологічних підходів до інтерпретації сновидінь, структури методики Hall–Van de Castle та можливостей обробки природної мови з використанням NLP і LLM-моделей. Також проведено аналіз існуючих аналогів програмного забезпечення, виявлено їх переваги й недоліки, що обґрунтувало доцільність створення власного застосунку.

– Визначено функціональні та нефункціональні вимоги до програмного забезпечення, які включають підтримку авторизації, введення текстів снів, інтеграцію з GPT API, класифікацію результатів, журналювання запитів і зберігання результатів у базі даних.

– Обрано стек технологій для реалізації системи, що включає Java 17, Spring Boot, Spring Security, H2 Database, GPT API від OpenAI, бібліотеки Jackson і Lombok, а також використано Java Swing і Tailwind CSS для побудови графічного інтерфейсу користувача.

– Розроблено архітектуру системи, описано структуру класів, взаємодію між сервісами та модель обробки даних. Реалізовано ключові модулі: GPTClient, DreamAnalysisService, парсер Hall–Van de Castle, модулі авторизації та бази даних.

– Виконано тестування функціональних компонентів та верифікацію результатів GPT-аналізу на основі тестових описів снів. Проведено аналіз типових помилок, пов’язаних із некоректним введенням, обмеженнями моделі або відсутністю підключення до API.

Таким чином, створене програмне забезпечення SomniCast поєднує у собі сучасні технології штучного інтелекту з психологічно валідованими методами аналізу сновидінь, забезпечуючи інтуїтивно зрозумілу взаємодію, точність результатів та зручність у використанні. Отримані результати можуть бути використані не лише в індивідуальному застосуванні, а й у дослідницькій діяльності з вивчення несвідомих процесів у людини.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Шелестинський В.А., Аброскін Ю.Ю. Розробка програмного забезпечення для декодування снівна основі методики Hall-Van de Castle. // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.:ДУІКТ, 2025. С. 621-623.
2. Шелестинський В.А., Аброскін Ю.Ю. Розробка програмного забезпечення для декодування снівна основі методики Hall-Van de Castle. // Матеріали XII Всеукраїнській науково-практичній конференції молодих учених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2025», 15.05.2025, Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ імені Бориса Грінченка, 2025. С. 208-209.

ПЕРЕЛІК ПОСИЛАНЬ

1. Франкл В. Людина в пошуках справжнього сенсу. Київ: Наш формат, 2021. 256 с.
2. Фройд З. Тлумачення сновидінь. Київ: Основи, 2020. 512 с.
3. Юнг К. Г. Людина та її символи. Львів: Літопис, 2019. 432 с.
4. Hall C., Van de Castle R. The Content Analysis of Dreams. New York: Appleton-Century-Crofts, 1966. 308 p.
5. Москалець В. П. Загальна психологія: підручник. Київ: Ліра, 2020. 564 с.
6. Слюсар В. І. Обробка природної мови на основі торцевого добутку матриць. Тези доповідей 8-ї міжнар. наук.-техн. конф. “Проблеми телекомунікацій”. 2020. URL: https://www.researchgate.net/publication/345902118_Obrobka_prirodnoi_movi_na_osnovi_torcevogo_dobutku_matric
7. Павлюк М. М., Шепельова М. В. Сучасні психологічні технології надання психологічної допомоги сім’ям постраждалих у подоланні складних життєвих обставин: практичний посібник. Київ: Інститут психології НАПН України, 2020. 124 с.
8. Panok V. G., Chaplak Ya. V., Andreeva Ya. F. Основи психологічної допомоги: теорія та практика психоконсультування: навч. посіб. Чернівці: Книги – ХХІ, 2020. 312 с.
9. OpenAI. GPT API Documentation. URL: https://platform.openai.com/docs
10. Spring Framework Reference Documentation. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/
11. Hibernate ORM Documentation. URL: https://hibernate.org/orm/documentation/
12. H2 Database Engine Documentation. URL: https://www.h2database.com/
13. IntelliJ IDEA Community Edition. URL: https://www.jetbrains.com/idea/
14. Chollet F. Deep Learning with Python. 2nd ed. Shelter Island: Manning Publications, 2021. 504 p.

15. Vaswani A. et al. Attention Is All You Need. In: Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.

16. Cherednichenko O.Y., Ivashchenko O.V., Gontar Y.M., Vorona B.M. Інтелектуальний аналіз пропозицій товарів на основі контекстних рекомендацій. Вісник НТУ «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. 2021. №1320 (44). С. 57–66. https://doi.org/10.20998/2079-0023.2018.44.10

17. Zhang Q., Lu J., Zhang G. Recommender Systems in E-learning. Journal of Smart Environments and Green Computing. 2022. No.1(2). P. 76–89.

18. Patel D., Patel F., Chauhan U. Recommendation Systems: Types, Applications, and Challenges. International Journal of Computing and Digital Systems. 2023. Vol. 13, No. 1. P. 851–868. https://doi.org/10.12785/ijcds/130168

19. Manning C., Jurafsky D. Speech and Language Processing. Pearson, 2022. 1100 p.

20. Bishop C. M. Pattern Recognition and Machine Learning. Springer, 2006. 738 p.

21. Настанова до зводу Знань з управління проєктами. Настанова PMBOK. 7-е видання та стандарт з управління проєктами. Інститут Управління Проєктами. URL: https://pmiukraine.org/pmbok7

22. Закон України «Про забезпечення функціонування української мови як державної» від 25.04.2019 р. № 2704-VIII. URL: https://zakon.rada.gov.ua/laws/show/2704-19

23. APA Style Introduction. Purdue University. URL: https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_style_introduction.html

24. Bohdana Muzyka. User Flows. Як ця техніка допомагає в роботі над проєктами. 29.12.2020. URL: https://dou.ua/lenta/columns/user-flows/

25. R. G. Edwards et al. Referencing styles. Los Angeles: International Publishing, 2010. 280 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для декодування снів на основі методики Hall-Van de Castle

Виконав студент 4 курсу
групи ПД-44
Шелестинський Вадим Анатолійович
Керівник роботи
асистент кафедри ІПЗ Аброскін Юрій Юрійович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спрощення організації самопізнання через інтерпретацію сновидінь на основі методики Hall–Van de Castle.
- **Об'єкт дослідження:** процес інтерпретації сновидінь через класифікацію змісту за визначеними психологічними категоріями методики Hall–Van de Castle.
- **Предмет дослідження:** програмне забезпечення для аналізу та класифікації сновидінь із використанням GPT-моделі і методики Hall–Van de Castle.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати сучасні психологічні підходи до інтерпретації сновидінь та методику Hall–Van de Castle.
2. Проаналізувати можливості використання NLP та великих мовних моделей (GPT) для семантичного аналізу снів.
3. Провести аналіз існуючих застосунків для інтерпретації снів та визначити їх основні недоліки.
4. Визначити функціональні та нефункціональні вимоги, спроєктувати архітектуру програмного забезпечення.
5. Визначити програмні інструменти та технології, що використовуватимуться для реалізації застосунку.
6. Розробити програмний застосунок для автоматизованої інтерпретації снів із використанням GPT API.
7. Провести тестування застосунку на відповідність функціональним та нефункціональним вимогам.

3

АНАЛІЗ АНАЛОГІВ

	DreamApp	Dream Interpreter AI	DreamAI	Sleep as Android	SomniCast
Підтримка методики Hall – Van de Castle	-	-	-	-	+
Використання GPT для NLP-аналізу	+	+	+	-	+
Україномовний інтерфейс	-	-	-	-	+
Десктопна кросплатформна версія	-	-	-	-	+
Відкрите API / інтеграції	-	-	-	+	+
Візуалізація статистики сновидінь	+	-	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Введення сну у вільній формі
- Інтеграція з GPT API
- Класифікація за шкалами Hall–Van de Castle
- Збереження результатів (БД H2 / JSON)
- Повторний аналіз та редагування
- Авторизація та історія користувача
- Журналювання дій

Нефункціональні вимоги:

- Час відповіді до 5 с
- Можливість масштабування
- Базова безпека облікових записів
- Модульна архітектура

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Java

IntelliJ IDEA

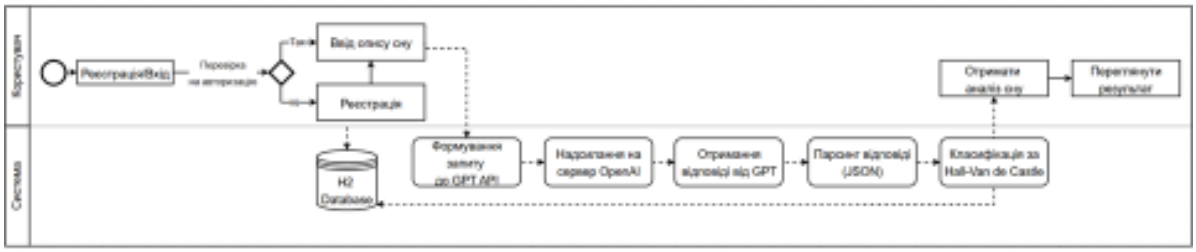
JSON

OpenAI API

H2

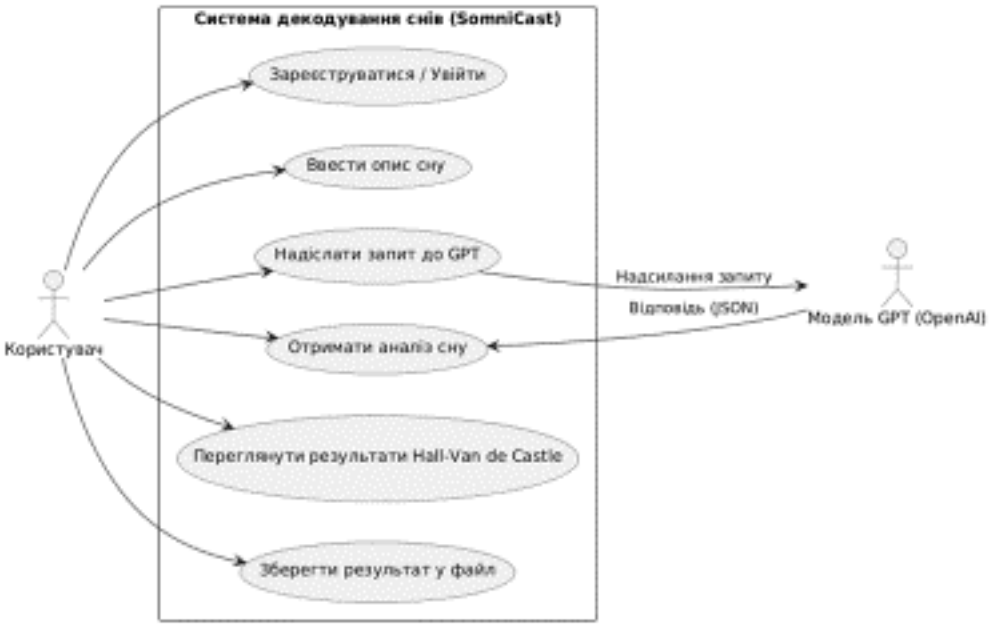
6

BPMN ДІАГРАМА



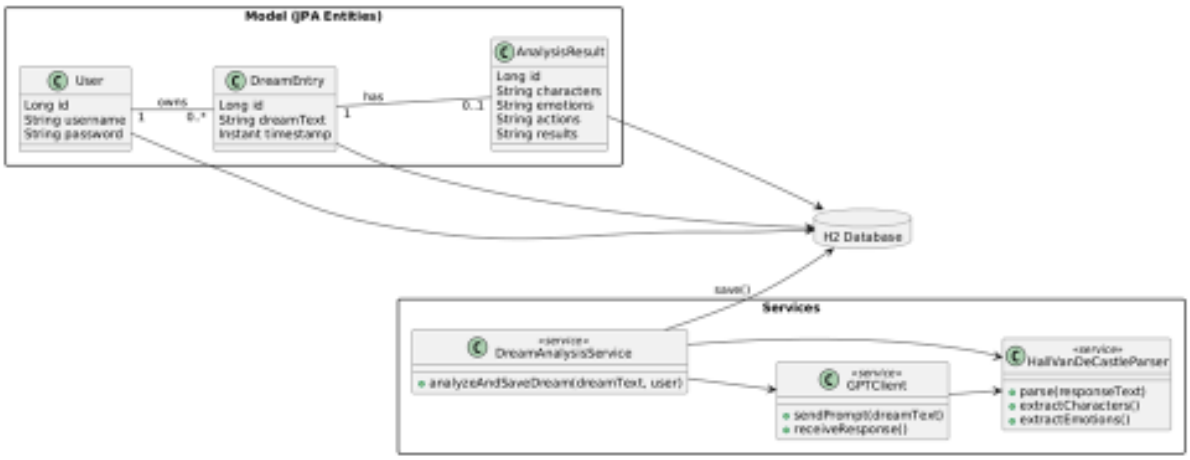
7

ДІАГРАМА ПРЕЦЕДЕНТІВ



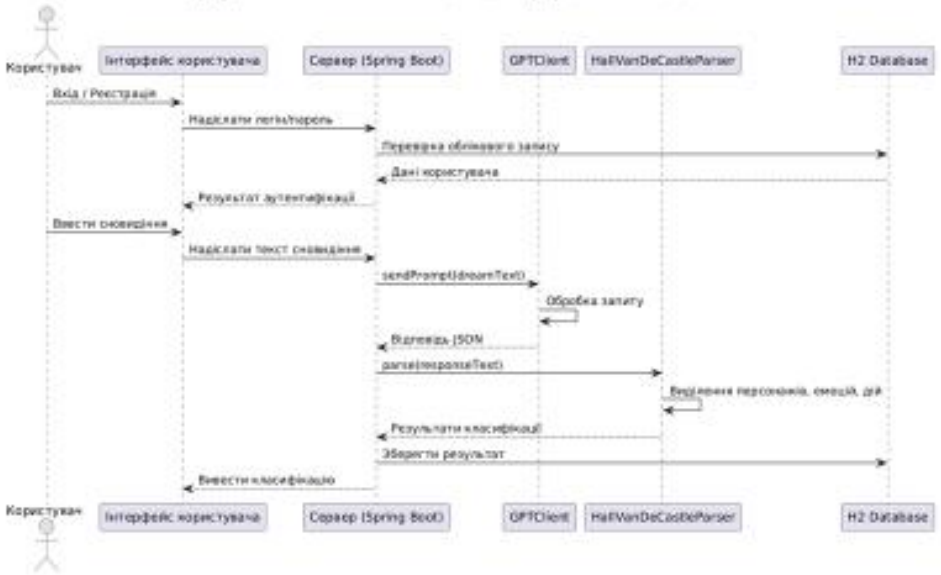
8

ДІАГРАМА КЛАСІВ



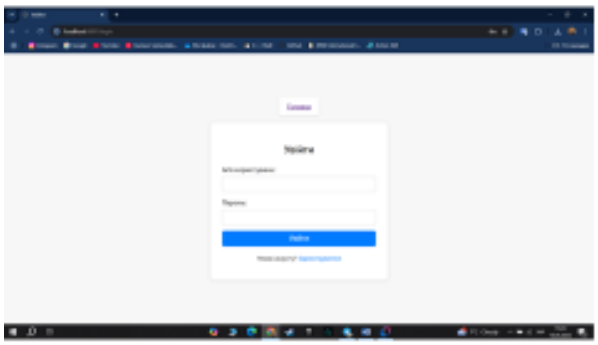
9

ДІАГРАМА ПОСЛІДОВНОСТІ

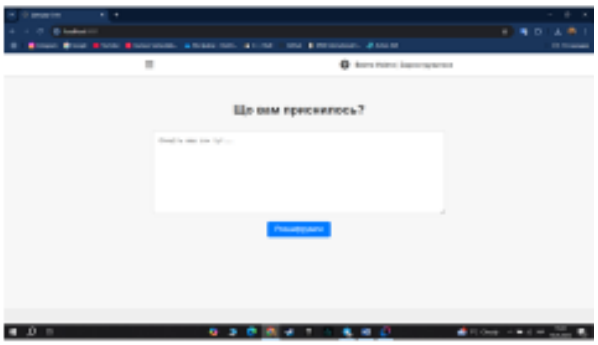


10

ЕКРАННІ ФОРМИ



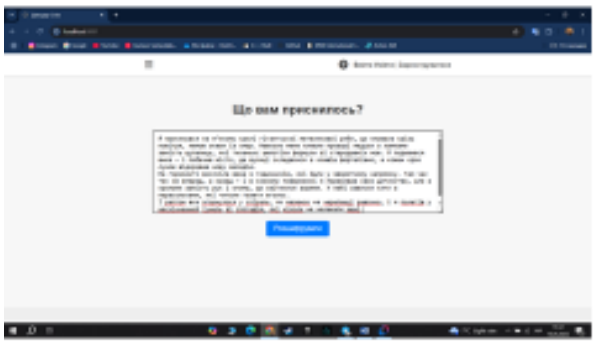
Форма авторизації користувача



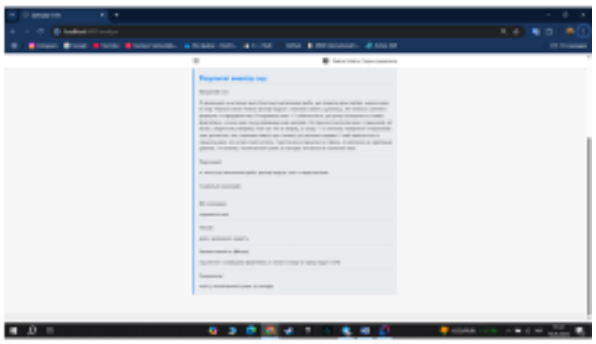
Інтерфейс введення сновидіння

11

ЕКРАННІ ФОРМИ



Введення текстового опису сновидіння

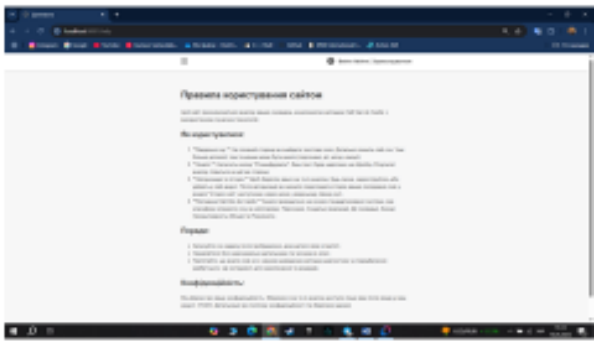


Результати класифікації за методикою Hall–Van de Castle

12

ЕКРАННІ ФОРМИ

Журнал оброблених сновидінь



Розділ довідки / Інструкція для користувача

13

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Шелестинський В.А., Аброскін Ю.Ю. Розробка програмного забезпечення для декодування снівна основі методики Hall-Van de Castle. // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.:ДУІКТ, 2025. С. 621-623.
2. Шелестинський В.А., Аброскін Ю.Ю. Розробка програмного забезпечення для декодування снівна основі методики Hall-Van de Castle. // Матеріали XII Всеукраїнській науково-практичній конференції молодих учених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2025», 15.05.2025, Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ імені Бориса Грінченка, 2025. С. 208-209.

15

ВИСНОВКИ

1. Проаналізовано основні теоретичні підходи до інтерпретації сновидінь, зокрема методику Hall–Van de Castle як науково обґрунтовану основу для структурованого аналізу снів.
2. Проаналізовано потенціал NLP і великих мовних моделей (GPT) для автоматизації психологічного аналізу текстів сновидінь.
3. Проведено аналіз існуючих застосунків для інтерпретації снів та визначено їхні недоліки, що враховано під час проєктування власного програмного забезпечення.
4. Визначено функціональні та нефункціональні вимоги до ПЗ, розроблено архітектуру ПЗ з урахуванням принципів масштабованості та модульності.
5. Реалізовано застосунок на основі Java та GPT API з використанням IntelliJ IDEA, H2 Database та формату JSON, який автоматично аналізує текст сновидіння та формує структурований результат.
6. Здійснено класифікацію елементів снів за категоріями методики Hall–Van de Castle.
7. Проведено тестування програмного забезпечення на відповідність функціональним та нефункціональним вимогам.

15

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
// ======= AnalysisResult.java =======
package org.example;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// Цей клас зберігатиме розпарсені результати аналізу від
LLM за категоріями HVC.
// Поля можуть бути String, якщо ви зберігаєте простий
текст, або JSON/List<String>
// залежно від того, як LLM повертає дані і як ви їх
парсите.
// Найпростіше почати зі String полів.

@Entity
@Table(name = "analysis_results")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AnalysisResult {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Зв'язок з записом сну (один результат -> один сон)
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "dream_entry_id", nullable = false) //
Колонка зовнішнього ключа
    private DreamEntry dreamEntry;

    // Поля за категоріями Hall-Van de Castle
    @Column(columnDefinition = "TEXT") // Може бути
довгим описом персонажів
    private String characters; // Опис персонажів

    @Column(columnDefinition = "TEXT")
    private String socialInteractions; // Опис соціальних
взаємодій

    @Column(columnDefinition = "TEXT")
    private String dreamerActivity; // Опис дій сновидця

    @Column(columnDefinition = "TEXT")
    private String emotions; // Опис емоцій у сні

    @Column(columnDefinition = "TEXT")
    private String setting; // Опис місця дії

    @Column(columnDefinition = "TEXT")
    private String outcomes; // Опис результатів

    // Можливо, поле для зберігання сирого JSON відповіді
від LLM (опціонально)
    // @Column(columnDefinition = "TEXT")
    // private String rawAnalysisJson;


    // Конструктор для зручності створення результату
    public AnalysisResult(DreamEntry dreamEntry, String
characters, String socialInteractions, String dreamerActivity,
String emotions, String setting, String outcomes) {
        this.dreamEntry = dreamEntry;
        this.characters = characters;
        this.socialInteractions = socialInteractions;
        this.dreamerActivity = dreamerActivity;
        this.emotions = emotions;
        this.setting = setting;
        this.outcomes = outcomes;
    }

    // Метод зв'язку в обидва боки
    public void setDreamEntry(DreamEntry dreamEntry) {
        this.dreamEntry = dreamEntry;
        if (dreamEntry != null &&
dreamEntry.getAnalysisResult() != this) {
            dreamEntry.setAnalysisResult(this);
        }
    }
}

// ======= AnalysisResultRepository.java =======
package org.example;

import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

// Інтерфейс для роботи з сутністю AnalysisResult
public interface AnalysisResultRepository extends
JpaRepository<AnalysisResult, Long> {

    // Знайти результат аналізу за відповідним записом сну
    Optional<AnalysisResult> findByDreamEntry(DreamEntry
dreamEntry);
}

// ======= AuthController.java =======
package org.example;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.servlet.mvc.support.RedirectAttribut
es;


@Controller
public class AuthController {

    private final UserService userService;

    public AuthController(UserService userService) {
        this.userService = userService;
    }

    // Обробка GET запиту на сторінку входу (Spring Security
зазвичай обробляє POST)
    @GetMapping("/login")
    public String login() {
```

```
        return "login"; // Назва Thymeleaf шаблону для
сторінки входу (login.html)
    }

    // Обробка GET запиту на сторінку реєстрації
    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("registrationForm", new
RegistrationForm()); // Додаємо порожню форму
        return "register"; // Назва Thymeleaf шаблону
(register.html)
    }

    // Обробка POST запиту для реєстрації
    @PostMapping("/register")
    public String processRegistration(@ModelAttribute
RegistrationForm registrationForm, // Автоматична валідація
                                      BindingResult result, // Результати
валідації
                                      RedirectAttributes redirectAttributes) {

        if (result.hasErrors()) {
            // Якщо є помилки валідації, повертаємо форму
назад
            return "register";
        }

        // TODO: Додати перевірку співпадіння паролів тут
або у сервісі/валідаторі
        if
(!registrationForm.getPassword().equals(registrationForm.getC
onfirmPassword())) {
            result.rejectValue("confirmPassword",
"error.registrationForm", "Паролі не співпадають");
            return "register";
        }


        try {
            userService.registerNewUser(registrationForm);
            // При успіху перенаправляємо на сторінку входу з
повідомленням
            redirectAttributes.addFlashAttribute("successMessage",
"Реєстрація успішна! Тепер ви можете увійти.");
            return "redirect:/login";
        } catch (Exception e) {
            // TODO: Обробка інших помилок реєстрації
(наприклад, користувач вже існує)
            result.rejectValue("username", "error.registrationForm",
e.getMessage());
            return "register"; // Повертаємо форму з
повідомленням про помилку
        }
    }

    // Spring Security автоматично обробляє POST /login та
/logout
}

// ======= ChatGPTApiClient.java =======
package org.example;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.EnvironmentAware;
import org.springframework.core.env.Environment;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import
org.springframework.web.reactive.function.client.WebClient;

import jakarta.annotation.PostConstruct; // Додайте цей
імпорт

// Приклад структури для тіла запиту до OpenAI API
(модель gpt-3.5-turbo або gpt-4)
record ChatRequest(String model, java.util.List<Message>
messages, double temperature) {}
record Message(String role, String content) {}
record ChatResponse(java.util.List<Choice> choices) {}
record Choice(Message message) {}
record ErrorResponse(ErrorDetails error) {}
record ErrorDetails(String message, String type) {}


@Component
public class ChatGPTApiClient {

    private WebClient webClient; // Зробіть поле не final,
оскільки воно буде ініціалізовано в @PostConstruct

    // Отримуємо API ключ та інші властивості через
@Value або Environment
    @Value("${openai.api.key}")
    private String openaiApiKey;

    @Value("${openai.api.url}")
    private String openaiApiUrl;

    @Value("${openai.model}")
    private String openaiModel;

    private final WebClient.Builder webClientBuilder; //
Зберігаємо WebClient.Builder

    public ChatGPTApiClient(WebClient.Builder
webClientBuilder) {
        this.webClientBuilder = webClientBuilder;
    }

    @PostConstruct // Метод, який буде виконано після
ін'єкції залежностей та @Value
    public void init() {
        // Перевірка на випадок, якщо властивість не
завантажилася
        if (this.openaiApiUrl == null ||
this.openaiApiUrl.trim().isEmpty()) {
            throw new IllegalStateException("OpenAI API URL is
not configured. Please check application.properties");
        }

        // Конфігуруємо WebClient з базовим URL та
заголовками
        this.webClient = webClientBuilder
                .baseUrl(this.openaiApiUrl)
                .defaultHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
                .defaultHeader(HttpHeaders.AUTHORIZATION,
"Bearer " + this.openaiApiKey)
                .build();
    }


    // Метод для виклику API
    public String callApi(String prompt) {
        // Перевіряємо, чи WebClient ініціалізовано
        if (this.webClient == null) {
```

```
            throw new IllegalStateException("ChatGPTApiClient is
not initialized.");
        }
        // Формуємо тіло запиту відповідно до API OpenAI
        java.util.List<Message> messages = java.util.List.of(
                new Message("system", "You are a helpful assistant
that analyzes dreams based on the Hall-Van de Castle
method."),
                new Message("user", prompt)
        );

        ChatRequest requestBody = new
ChatRequest(openaiModel, messages, 0.7);

        // Виконуємо POST запит
        ChatResponse apiResponse = webClient.post()
                .bodyValue(requestBody)
                .retrieve()
                .onStatus(status -> status.is4xxClientError() ||
status.is5xxServerError(), response ->

response.bodyToMono(ErrorResponse.class).flatMap(error ->
{
                        System.err.println("OpenAI API Error: " +
error.error().type() + " - " + error.error().message());
                        return reactor.core.publisher.Mono.error(new
RuntimeException("Error from OpenAI API: " +
error.error().message()));
                    })
                )
                .bodyToMono(ChatResponse.class)
                .block();

        // Отримуємо текст відповіді
        if (apiResponse != null && apiResponse.choices() != null
&& !apiResponse.choices().isEmpty()) {
            return
apiResponse.choices().get(0).message().content();
        } else {
            // TODO: Обробка випадку, коли відповідь порожня
або не містить тексту
            throw new RuntimeException("Failed to get response
text from OpenAI API");
        }
    }
}

// ======= CustomUserDetailsService.java =======
package org.example;

import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsServi
ce;
import
org.springframework.security.core.userdetails.UsernameNotFo
undException;
import org.springframework.stereotype.Service;
import
org.springframework.security.core.authority.SimpleGrantedAut
hority;

import java.util.Collections; // Приклад для простих ролей

@Service // Позначаємо як Сервіс Spring
public class CustomUserDetailsService implements
UserDetailsService {

    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository
userRepository) {
        this.userRepository = userRepository;
    }

    // Метод, який використовується Spring Security для
завантаження деталей користувача за його іменем
    @Override
    public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
        // Знаходимо користувача в базі даних
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new
UsernameNotFoundException("User not found with username:
" + username));

        // TODO: Перетворити ролі вашої сутності User у
GrantedAuthority для Spring Security
        // Приклад для простих ролей (якщо у User є поле
Set<String> roles):
        /*
        List<GrantedAuthority> authorities =
user.getRoles().stream()

.map(SimpleGrantedAuthority::new)
                                .collect(Collectors.toList());
        */

        // Приклад, якщо у вас поки немає ролей у сутності
User, але ви хочете надати базову роль
        // Для автентифікованого користувача можна додати
роль "ROLE_USER"
        return new
org.springframework.security.core.userdetails.User(
                user.getUsername(), // Ім'я користувача
                user.getPassword(), // Хешований пароль
                Collections.singletonList(new
SimpleGrantedAuthority("ROLE_USER")) // Ролі
користувача
        );
    }
}

// ======= DreamAnalysisService.java =======
package org.example;

import com.fasterxml.jackson.databind.ObjectMapper; //
Імпортуємо ObjectMapper
import com.fasterxml.jackson.core.JsonProcessingException; //
Імпортуємо для обробки помилок парсингу
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors; // Імпортуємо для роботи
зі стрімами (об'єднання списків)
import java.util.Collections; // Імпортуємо для порожніх
списків, якщо потрібно

@Service
public class DreamAnalysisService {

    private final DreamEntryRepository dreamEntryRepository;
    private final AnalysisResultRepository
analysisResultRepository;
    private final ChatGPTApiClient chatGPTApiClient;
```

```
    public DreamAnalysisService(DreamEntryRepository
dreamEntryRepository,
                                AnalysisResultRepository
analysisResultRepository,
                                ChatGPTApiClient chatGPTApiClient) {
        this.dreamEntryRepository = dreamEntryRepository;
        this.analysisResultRepository = analysisResultRepository;
        this.chatGPTApiClient = chatGPTApiClient;
    }

    @Transactional
    public AnalysisResult analyzeAndSaveDream(String
dreamText, User user) {
        DreamEntry dreamEntry = new DreamEntry(dreamText,
user);
        // Зберігаємо dreamEntry тут, якщо user не null і
потрібно зберегти зв'язок одразу
        if (user != null) {
            dreamEntry =
dreamEntryRepository.save(dreamEntry);
        } else {
            // TODO: Обробка випадку аналізу без збереження
для неавторизованих
            // Якщо не зберігаємо в БД, то DreamEntry
analysisResult.setDreamEntry(dreamEntry);
            // не буде мати сенсу. Можливо, варто повертати
лише AnalysisResult без зв'язку.
            // Наразі код нижче все одно намагається зберегти
AnalysisResult, що викличе помилку без DreamEntry.
            // Це потребує доопрацювання логіки для
неавторизованих користувачів.
        }


        String prompt = buildHvcPrompt(dreamText);
        String rawApiResponse =
chatGPTApiClient.callApi(prompt);

        AnalysisResult analysisResult =
parseHvcAnalysis(rawApiResponse);

        // Зв'язуємо результат аналізу з записом сну ТІЛЬКИ
якщо сон був збережений
        if (user != null && dreamEntry != null) {
            analysisResult.setDreamEntry(dreamEntry);
            // Зберігаємо результат аналізу ТІЛЬКИ якщо сон
був збережений
            analysisResult =
analysisResultRepository.save(analysisResult);
        }


        return analysisResult; // Повертаємо результат для
відображення
    }

    private String buildHvcPrompt(String dreamText) {
        return """
                Проаналізуй наступний текст сну за методикою
Hall-Van de Castle. Виділи та класифікуй елементи за
категоріями:
                Персонажі: ...
                Соціальні взаємодії: ...
                Дії суб'єкта сну: ...
                Емоції: ...
                Налаштованість (Setting): ...
                Результати: ...

                Надай результат у форматі JSON з такими
ключами (уважно до назв):
                "characters": [...],
                "interactions": [...],
                "dreamer_activity": "...",
                "emotions": [...],
                "setting": "...",
                "outcomes": "..."

                Якщо список порожній, використовуй "ключ": [].
Якщо одиночне значення відсутнє, використовуй "ключ":
null.

                Текст сну:
                """ + dreamText;
    }

    // Приватний допоміжний метод для парсингу відповіді
LLM
    private AnalysisResult parseHvcAnalysis(String
rawApiResponse) {
        ObjectMapper objectMapper = new ObjectMapper();
        AnalysisResult analysisResult = new AnalysisResult();

        System.out.println("Raw API Response: " +
rawApiResponse); // Залишаємо для налагодження

        // Очищаємо відповідь від маркдаун-форматування
        String cleanedApiResponse = rawApiResponse;
        if (cleanedApiResponse != null) {
            cleanedApiResponse = cleanedApiResponse.trim(); //
Видаляємо початкові/кінцеві пробіли
            if (cleanedApiResponse.startsWith("```json")) {
                cleanedApiResponse =
cleanedApiResponse.substring("```json".length()).trim();
            }
            if (cleanedApiResponse.endsWith("```")) {
                cleanedApiResponse =
cleanedApiResponse.substring(0, cleanedApiResponse.length()
- "```".length()).trim();
            }
        }

        System.out.println("Cleaned API Response: " +
cleanedApiResponse); // Вивід для налагодження

        try {
            // Парсимо очищений JSON у наш DTO
            HvcAnalysisDto parsedData =
objectMapper.readValue(cleanedApiResponse,
HvcAnalysisDto.class);

            // Заповнюємо поля AnalysisResult, об'єднуючи
списки у рядки та обробляючи null
            analysisResult.setCharacters(
                    parsedData.getCharacters() != null &&
!parsedData.getCharacters().isEmpty() ?
                            String.join(", ", parsedData.getCharacters()) :
""
            );

            analysisResult.setSocialInteractions(
                    parsedData.getSocialInteractions() != null &&
!parsedData.getSocialInteractions().isEmpty() ?
                            String.join(", ",
parsedData.getSocialInteractions()) : ""
            );

            analysisResult.setDreamerActivity(
```

```
                parsedData.getDreamerActivity() != null ?
parsedData.getDreamerActivity() : ""
            );

            analysisResult.setEmotions(
                parsedData.getEmotions() != null &&
!parsedData.getEmotions().isEmpty() ?
                        String.join(", ", parsedData.getEmotions()) :
""
            );

            analysisResult.setSetting(
                parsedData.getSetting() != null ?
parsedData.getSetting() : ""
            );

            analysisResult.setOutcomes(
                parsedData.getOutcomes() != null ?
parsedData.getOutcomes() : ""
            );

            System.out.println("Successfully parsed API
Response.");


        } catch (JsonProcessingException e) {
            // Обробка помилок парсингу JSON
            System.err.println("Error parsing HVC analysis JSON:
" + e.getMessage());
            e.printStackTrace();
            analysisResult.setCharacters("Помилка парсингу
відповіді від API.");
            analysisResult.setSocialInteractions("Не вдалося
отримати результат аналізу через помилку даних.");
            analysisResult.setDreamerActivity("");
            analysisResult.setEmotions("");
            analysisResult.setSetting("");
            analysisResult.setOutcomes("");
        } catch (Exception e) {
            // Обробка будь-яких інших неочікуваних помилок
            System.err.println("An unexpected error occurred
during parsing: " + e.getMessage());
            e.printStackTrace();
            analysisResult.setCharacters("Виникла неочікувана
помилка під час аналізу.");
            analysisResult.setSocialInteractions("Спробуйте ще
раз або зверніться до підтримки.");
            analysisResult.setDreamerActivity("");
            analysisResult.setEmotions("");
            analysisResult.setSetting("");
            analysisResult.setOutcomes("");
        }

        return analysisResult;
    }
    // Метод для отримання історії снів користувача
    public List<DreamEntry> getUserDreamHistory(User user)
{
        return
dreamEntryRepository.findByUserOrderByTimestampDesc(us
er);
    }

    // TODO: Можливо, додати метод
analyzeDreamWithoutSaving(String dreamText)
    // для обробки снів неавторизованих користувачів без
взаємодії з БД
}
```

```
// ======= DreamEntry.java =======
package org.example;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.Instant; // Або LocalDateTime

@Entity
@Table(name = "dream_entries")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class DreamEntry {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(columnDefinition = "TEXT", nullable = false) //
Текст сну може бути довгим
    private String dreamText;

    @Column(nullable = false)
    private Instant timestamp = Instant.now(); // Час створення
запису

    @ManyToOne(fetch = FetchType.LAZY) // Багато снів
належать одному користувачеві
    @JoinColumn(name = "user_id", nullable = false) //
Колонка зовнішнього ключа
    private User user;

    // Зв'язок з результатом аналізу (один сон -> один
результат)
    @OneToOne(mappedBy = "dreamEntry", cascade =
CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    private AnalysisResult analysisResult;

    // Конструктор для зручності створення запису сну
    public DreamEntry(String dreamText, User user) {
        this.dreamText = dreamText;
        this.user = user;
        this.timestamp = Instant.now();
    }
}

// ======= DreamEntryRepository.java =======
package org.example;

import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

// Інтерфейс для роботи з сутністю DreamEntry
public interface DreamEntryRepository extends
JpaRepository<DreamEntry, Long> {

    // Знайти всі записи снів для конкретного користувача
    List<DreamEntry>
findByUserOrderByTimestampDesc(User user); //
Відсортовано за часом спадання
}

// ======= DreamInputForm.java =======
package org.example;

import lombok.Data;
```

```
// Клас для збору тексту сну з форми
@Data
public class DreamInputForm {

    private String dreamText;

    // TODO: Додати анотації валідації (наприклад,
@NotBlank)
}

// ======= HelpController.java =======
package org.example;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HelpController {

    // Обробка GET запиту на сторінку допомоги
    @GetMapping("/help")
    public String showHelpPage() {
        return "help"; // Назва Thymeleaf шаблону (help.html)
    }
}

// ======= HistoryController.java =======
package org.example;

import
org.springframework.security.access.prepost.PreAuthorize; //
Для захисту методу
import
org.springframework.security.core.annotation.AuthenticationPr
incipal; // Отримати UserDetails
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
public class HistoryController {

    private final DreamAnalysisService dreamAnalysisService;
    private final UserService userService;

    public HistoryController(DreamAnalysisService
dreamAnalysisService, UserService userService) {
        this.dreamAnalysisService = dreamAnalysisService;
        this.userService = userService;
    }

    // Обробка GET запиту на сторінку історії
    @GetMapping("/history")
    @PreAuthorize("isAuthenticated()") // Цей метод
доступний тільки автентифікованим користувачам
    public String showHistory(Model model,
@AuthenticationPrincipal UserDetails userDetails) { //
Отримуємо деталі поточного користувача

        // Знаходимо нашого користувача з бази за іменем
        User currentUser =
userService.findByUsername(userDetails.getUsername())
                .orElseThrow(() -> new
RuntimeException("Authenticated user not found in
database")); // Має не статися

        // Отримуємо історію снів для поточного користувача
        List<DreamEntry> dreamHistory =
dreamAnalysisService.getUserDreamHistory(currentUser);

        model.addAttribute("dreamHistory", dreamHistory); //
Додаємо історію в модель

        return "history"; // Назва Thymeleaf шаблону
(history.html)
    }
}

// ======= HomeController.java =======
package org.example;


import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHol
der;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.ModelAttribute;
import
org.springframework.web.bind.annotation.PostMapping;

import java.security.Principal; // Альтернатива для
отримання імені користувача

@Controller // Позначає клас як MVC контролер
public class HomeController {

    private final DreamAnalysisService dreamAnalysisService;
    private final UserService userService; // Щоб отримати
поточного користувача

    public HomeController(DreamAnalysisService
dreamAnalysisService, UserService userService) {
        this.dreamAnalysisService = dreamAnalysisService;
        this.userService = userService;
    }

    // Обробка GET запиту на головну сторінку
    @GetMapping("/")
    public String home(Model model) {
        model.addAttribute("dreamInputForm", new
DreamInputForm()); // Додаємо порожню форму для
Thymeleaf
        // model.addAttribute("analysisResult", null); // Спочатку
результат аналізу відсутній
        return "index"; // Назва Thymeleaf шаблону (наприклад,
src/main/resources/templates/index.html)
    }

    // Обробка POST запиту для аналізу сну
    @PostMapping("/analyze") // Або можна обробляти POST
на "/", але окремий шлях зрозуміліший
    public String analyzeDream(@ModelAttribute
DreamInputForm dreamInputForm,
                               Model model,
                               Principal principal) { // Principal надає
інформацію про авторизованого користувача
```

```
        // TODO: Додати валідацію форми (@Valid та
BindingResult)

        User currentUser = null;
        if (principal != null) {
            // Отримуємо користувача з бази даних за його
іменем (Principal.getName())
            currentUser =
userService.findByUsername(principal.getName()).orElse(null)
;
            // TODO: Обробка випадку, якщо користувач не
знайдений (хоча зазвичай не має статися після
автентифікації)
        }
        // TODO: Обробка випадку, якщо користувач не
авторизований, але надсилає форму (можна дозволити
аналіз без збереження)

        AnalysisResult analysisResult = null;
        try {
            // Викликаємо сервіс для аналізу та збереження сну
            // Якщо користувач null, можна реалізувати логіку
аналізу без збереження
            if (currentUser != null) {
                analysisResult =
dreamAnalysisService.analyzeAndSaveDream(dreamInputFor
m.getDreamText(), currentUser);
            } else {
                // TODO: Реалізувати логіку аналізу без
збереження в БД для неавторизованих
                // Можливо, викликати API, але не зберігати
DreamEntry та AnalysisResult
                // Або просто показати повідомлення, що для
збереження потрібна авторизація.
                // analysisResult =
dreamAnalysisService.analyzeDreamWithoutSaving(dreamInp
utForm.getDreamText()); // Потрібен такий метод у сервісі
                model.addAttribute("message", "Авторизуйтесь,
щоб зберегти історію снів.");
                // Для демонстрації поки що проаналізуємо, але
не збережемо:
                analysisResult =
dreamAnalysisService.analyzeAndSaveDream(dreamInputFor
m.getDreamText(), null); // Не передаємо користувача,
логіка сервісу має це обробити
            }


        } catch (Exception e) {
            // TODO: Обробка помилок (наприклад, помилки
API ChatGPT, помилки парсингу)
            model.addAttribute("errorMessage", "Виникла
помилка під час аналізу сну: " + e.getMessage());
            e.printStackTrace(); // Для налагодження
        }

        model.addAttribute("dreamInputForm", new
DreamInputForm()); // Додаємо нову порожню форму після
обробки
        model.addAttribute("analysisResult", analysisResult); //
Додаємо результат аналізу в модель
        model.addAttribute("submittedDreamText",
dreamInputForm.getDreamText()); // Можливо, показати
введений текст

        return "index"; // Повертаємо ту саму сторінку, щоб
відобразити результат під формою
    }
    // TODO: Додати обробку POST для неавторизованих,
якщо аналіз без збереження дозволено
}

// ======= HvcAnalysisDto.java =======
package org.example;

import lombok.Data;
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.List;

@Data
public class HvcAnalysisDto {

    private List<String> characters;
    @JsonProperty("interactions") // Вказуємо, що це поле
відповідає ключу "interactions" у JSON
    private List<String> socialInteractions;

    @JsonProperty("dreamer_activity") // !!! ДОДАНО:
вказуємо відповідний ключ JSON
    private String dreamerActivity;

    private List<String> emotions;

    private String setting;
    private String outcomes;
}

// ======= Main.java =======
package org.example;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

@SpringBootApplication
public class Main {
    public static void main(String[] args) {
        SpringApplication.run(Main.class, args);
    }
}


// ======= RegistrationForm.java =======
package org.example;

import lombok.Data;

// Клас для збору даних з форми реєстрації
@Data // Lombok: гетери, сетери, toString тощо
public class RegistrationForm {

    private String username;
    private String password;
    private String confirmPassword; // Для перевірки
підтвердження пароля

    // TODO: Додати анотації валідації JSR-303 (наприклад,
@NotBlank, @Size, @Pattern)
}

// ======= SecurityConfig.java =======
package org.example;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import
org.springframework.security.config.annotation.method.config
uration.EnableMethodSecurity; // Для @PreAuthorize
import
org.springframework.security.config.annotation.web.builders.H
ttpSecurity;
import
org.springframework.security.config.annotation.web.configurat
ion.EnableWebSecurity;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordE
ncoder; // Рекомендований хешер паролів
import
org.springframework.security.crypto.password.PasswordEncod
er;
import org.springframework.security.web.SecurityFilterChain;

@Configuration // Позначає клас як Конфігурацію Spring
@EnableWebSecurity // Включає веб-безпеку Spring
Security
@EnableMethodSecurity // Включає безпеку на рівні
методів (наприклад, @PreAuthorize)
public class SecurityConfig {

   private final CustomUserDetailsService
customUserDetailsService;

   public SecurityConfig(CustomUserDetailsService
customUserDetailsService) {
      this.customUserDetailsService =
customUserDetailsService;
   }

   // Конфігурація ланцюжка фільтрів безпеки
   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
      http
            .authorizeHttpRequests(authorize -> authorize
                  .requestMatchers("/login", "/register",
"/css/**").permitAll() // Дозволити доступ без
автентифікації
                  // TODO: Можливо, дозволити POST /analyze
для неавторизованих, якщо аналіз без збереження
дозволено
                  .anyRequest().authenticated() // Всі інші запити
вимагають автентифікації
            )
            .formLogin(form -> form
                  .loginPage("/login") // Вказуємо URL сторінки
входу
                  .defaultSuccessUrl("/", true) // URL, куди
перенаправляти після успішного входу
                  .permitAll() // Дозволити доступ до форми
входу всім
            )
            .logout(logout -> logout
                  .logoutUrl("/logout") // URL для виходу (за
замовчуванням POST)
                  .logoutSuccessUrl("/login") // URL, куди
перенаправляти після успішного виходу
                  .permitAll() // Дозволити доступ до виходу
всім
            )
            .userDetailsService(customUserDetailsService); //
Вказуємо наш сервіс для завантаження користувачів

      // TODO: Налаштувати CSRF захист, якщо потрібно
(зазвичай вмикається за замовчуванням)
      // http.csrf(AbstractHttpConfigurer::disable); // Можна
тимчасово вимкнути для тестів POST запитів без форм
Spring Security

      return http.build();
   }

   // Визначаємо Bean для хешера паролів
   @Bean
   public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder(); //
Рекомендований алгоритм
   }

   // Bean для UserDetailsService вже в
CustomUserDetailsService з анотацією @Service
   // Spring Boot автоматично його знайде.

}

// ======= User.java =======
package org.example;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Set; // Або List
import java.util.HashSet; // Або ArrayList

@Entity
@Table(name = "users") // Уникаємо конфлікту зі
зарезервованим словом 'user'
@Data // Lombok: генерує гетери, сетери, toString, equals,
hashCode
@NoArgsConstructor // Lombok: генерує конструктор без
аргументів
@AllArgsConstructor // Lombok: генерує конструктор з
усіма аргументами
public class User {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY) //
Автоінкрементний ID
   private Long id;

   @Column(nullable = false, unique = true) // Ім'я
користувача не може бути null і має бути унікальним
   private String username;

   @Column(nullable = false)
   private String password; // Зберігатиметься в хешованому
вигляді

   // Приклад простих ролей як Set рядків. Для складніших
сценаріїв потрібна окрема сутність Role.
   // @ElementCollection(fetch = FetchType.EAGER) //
Завантажувати ролі одразу з користувачем
   // private Set<String> roles = new HashSet<>();

   // Якщо потрібні зв'язки з DreamEntry - додаємо:
   // @OneToMany(mappedBy = "user", cascade =
CascadeType.ALL, fetch = FetchType.LAZY)
   // private Set<DreamEntry> dreamEntries;

   // Метод для додавання ролі, якщо використовуєте
Set<String>
   // public void addRole(String role) {
   //     this.roles.add(role);
```

```
    // }
}

// ======= UserRepository.java =======
package org.example;

import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

// Інтерфейс, який розширює JpaRepository для роботи з сутністю User
public interface UserRepository extends JpaRepository<User, Long> {

    // Spring Data JPA автоматично створить реалізацію цього методу
    Optional<User> findByUsername(String username);
}


// ======= UserService.java =======
package org.example;

import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service // Позначає клас як Сервіс Spring
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder; // Для хешування паролів

    // Впровадження залежностей через конструктор (рекомендовано)
    public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    // Метод для реєстрації нового користувача
    public User registerNewUser(RegistrationForm registrationForm) throws Exception {
        // TODO: Додати перевірку, чи користувач з таким ім'ям вже існує
        if (userRepository.findByUsername(registrationForm.getUsername()).isPresent()) {
            throw new Exception("Username already exists"); // Або кинути більш специфічне виключення
        }
        // TODO: Додати перевірку співпадіння паролів з форми

        User newUser = new User();
        newUser.setUsername(registrationForm.getUsername());

        newUser.setPassword(passwordEncoder.encode(registrationForm.getPassword())); // Хешуємо пароль
        // TODO: Присвоїти ролі (наприклад, "ROLE_USER") - якщо використовуєте Set<String>
        // newUser.addRole("ROLE_USER");

        return userRepository.save(newUser); // Зберігаємо користувача в базу
    }

    // Метод для пошуку користувача за іменем
    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    // Метод для пошуку користувача за ID (може знадобитися)
    public Optional<User> findById(Long id) {
        return userRepository.findById(id);
    }
}
```