

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри
Олександр КОВАЛЬ

«___» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Програмна система аналізу емоційного контексту діалогу на основі засобів
машиного навчання»**

Виконав:

студент IV курсу, групи ТВ-22
Кривой Володимир Анатолійович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

Асистент, д.ф. Пироговська Т.В.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

Професор кафедри, Отрох С. І.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці Рівень вищої освіти
перший (бакалаврський)

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Олександр КОВАЛЬ

«__» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

Кривому Володимир

Анатолійовичу

(прізвище, ім'я, по батькові)

Тема роботи: Пограмна система аналізу емоціонного контексту діалогу на основі
засобів машинного навчання

керівник роботи Пироговська Т.В., д.ф., асистент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026р. № 1992-с

2. Строк подання студентом роботи «10» червня 2026р.

3. Вихідні дані до роботи: мова програмування JavaScript; серверний фреймворк
Express.js; платформа локального запуску LLM Ollama; мікросервіс аналізу емоцій
Emotion API; технологія передачі даних Server-Sent Events; сховище діалогів
localStorage; формат векторної графіки SVG.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно
розробити): вступ; аналіз предметної області та формування вимог до системи;
аналіз існуючих систем; засоби розробки; реалізація та використання програмної
системи; висновки; список використаних джерел.

5. Перелік ілюстративного матеріалу: архітектура системи; схема SSE-потoku даних;
структура клієнтських модулів; діаграма streak-механізму.

6. Дата видачі завдання «31» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	Виконано
2	Дослідження предметної області	01.11.2025 – 31.11.2025	Виконано
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	Виконано
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	Виконано
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	Виконано
6	Тестування	04.05.2026 – 10.05.2026	Виконано
7	Захист програмного продукту	11.05.2026 – 15.05.2026	Виконано
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	Виконано
9	Передзахист	01.06.2026 – 05.06.2026	Виконано
10	Захист	15.06.2026 – 19.06.2026	Виконано

Студент

(підпис)

Володимир КРИВОЙ

(ім'я, прізвище)

Керівник роботи

(підпис)

Тетяна ПИРОГОВСЬКА

(ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. 111 с., 17 рис., 1 дод., 20 джерел.

Об'єктом дослідження є процес емоційно адаптивної взаємодії користувача з локальною мовною моделлю у реальному часі. Предметом дослідження є методи та засоби побудови вебзастосунків із візуалізацією емоційного контексту діалогу на основі великих мовних моделей та автоматичного аналізу емоцій.

Мета роботи: розробка вебзастосунку EmotiChat, що забезпечує природне спілкування з локальною мовною моделлю, аналізує емоційний стан обох учасників діалогу в реальному часі та відображає результати аналізу через анімований SVG-аватар і панель емоційних метрик.

У роботі проаналізовано сучасні підходи до побудови емоційно адаптивних AI-систем, обґрунтовано вибір клієнт-серверної архітектури на основі Node.js та Express.js, реалізовано стрімінгову передачу відповідей локальної мовної моделі Ollama через Server-Sent Events, інтегровано Emotion API для контекстного аналізу емоцій обох учасників діалогу, розроблено SVG-аватар із двадцятьма виразами обличчя та streak-механізмом ескалації реакцій, реалізовано панель емоційного контексту з траєкторією, температурою та sparkline-графіком, забезпечено персистентність діалогів із повним відновленням емоційного контексту через localStorage.

Результатом роботи є працездатний програмний продукт, що демонструє стабільну автономну роботу без хмарних залежностей та забезпечує наочну візуалізацію емоційної динаміки діалогу в реальному часі. Розроблений застосунок може бути використано як платформу для емоційної підтримки, персоналізованого навчання та дослідження людино-машинної взаємодії.

Ключові слова: ЕМОЦІЙНИЙ АНАЛІЗ, ВЕЛИКА МОВНА МОДЕЛЬ, OLLAMA, SERVER-SENT EVENTS, SVG-АВАТАР, STREAK-МЕХАНІЗМ, EMOTION API, NODE.JS, ЛОКАЛЬНИЙ AI, ВЕБЗАСТОСУНОК.

ABSTRACT

Structure and Scope of the Thesis. 111 pages, 17 figures, 1 appendix, 20 references.

The object of the study is the process of emotionally adaptive user interaction with a local language model in real time. The subject of the study is the methods and tools for building web applications that visualize the emotional context of a dialogue based on large language models and automatic emotion analysis.

The goal of the work is to develop the EmotiChat web application, which enables natural communication with a local language model, analyzes the emotional state of both dialogue participants in real time, and displays the analysis results via an animated SVG avatar and an emotional metrics panel.

The paper analyzes modern approaches to building emotionally adaptive AI systems, justifies the choice of a client-server architecture based on Node.js and Express.js, implements streaming transmission of responses from the local Ollama language model via Server-Sent Events, the Emotion API is integrated for contextual analysis of the emotions of both dialogue participants, an SVG avatar with twenty facial expressions and a streak-based reaction escalation mechanism is developed, an emotional context panel with a trajectory, temperature, and a sparkline graph; ensured the persistence of dialogs with full restoration of emotional context via localStorage.

The result of this work is a fully functional software product that demonstrates stable, standalone operation without cloud dependencies and provides a visual representation of the emotional dynamics of a dialogue in real time. The developed application can be used as a platform for emotional support, personalized learning, and research into human-computer interaction.

Keywords: EMOTIONAL ANALYSIS, LARGE LANGUAGE MODEL, OLLAMA, SERVER-SENT EVENTS, SVG AVATAR, STREAK MECHANISM, EMOTION API, NODE.JS, LOCAL AI, WEB APPLICATION.

ЗМІСТ

	стор.
ВСТУП.....	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ	9
1.1 Постановка задачі.....	12
1.2 Вимоги до системи	14
2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ.....	17
2.1 Порівняльний аналіз із системою Replika.....	17
2.2 Порівняльний аналіз із системою Character.AI.....	19
2.3 Порівняльний аналіз із системою Kindroid.....	20
3 ЗАСОБИ РОЗРОБКИ	23
3.1 Серверна частина застосунку	23
3.2 Мовна модель та аналіз емоцій.....	24
3.2.1 Локальна мовна модель Ollama.....	24
3.2.2 Інтеграція з Emotion API	26
3.3 Клієнтська частина	29
3.3.1 HTML5 та структура інтерфейсу	29
3.3.2 JavaScript та система анімацій	30
3.3.3 CSS3 та візуальне оформлення	32
3.4 Серверна частина застосунку	33
4 РЕАЛІЗАЦІЯ ТА ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ.....	36
4.1 Архітектура вебзастосунку	36
4.2 Робота користувача з програмною системою	40
4.2.1 Головний інтерфейс чату	40
4.2.2 Аватар та відображення емоцій	44
4.2.3 Історія діалогів та налаштування	50
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	57

ВСТУП

Сучасні технології штучного інтелекту відкривають принципово нові можливості для побудови комунікаційних систем, здатних не лише обмінюватися текстовими повідомленнями, але й аналізувати емоційний стан учасників діалогу. Зростаючий інтерес до емоційно адаптивних інтерфейсів обумовлений потребою у більш природній та людиноподібній взаємодії між користувачем і програмною системою.

Предметною областю практики є розробка вебзастосунку EmotiChat, що поєднує локальну велику мовну модель із системою аналізу емоцій у реальному часі та виразним SVG аватаром із динамічною мімікою. Основною ідеєю є створення чат системи, в якій штучний інтелект не просто генерує текстові відповіді, а відстежує емоційний контекст кожного повідомлення та відповідним чином змінює поведінку свого візуального представника. Такий підхід дозволяє зробити взаємодію з ШІ більш виразною та інформативною, відображаючи емоційну динаміку діалогу через реакції аватара, емоційні чіпи повідомлень і аналітичну панель контексту.

У межах практики було реалізовано повнофункціональну систему емоційно адаптивного чату з підтримкою стрімінгу відповідей через SSE, двох послідовних звернень до Emotion API для аналізу емоцій користувача та асистента, персистентного збереження діалогів із повним відновленням емоційного стану, а також кастомізованого SVG аватара із двадцятьма унікальними виразами обличчя. Для реалізації серверної частини використовувався Node.js із Express.js, а локальний LLM рушій Ollama забезпечив повну автономність системи без залежності від хмарних сервісів.

Практична цінність розробленої системи полягає у демонстрації підходу до побудови емоційно чутливих інтерфейсів на основі локальних мовних моделей без використання зовнішніх API та передачі персональних даних. У перспективі

систему може бути розширено до повноцінного інструменту психологічної підтримки або навчальної платформи, де емоційний контекст діалогу використовується для адаптації стилю спілкування та надання персоналізованих рекомендацій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

У системах діалогової взаємодії з штучним інтелектом однією з ключових проблем є відсутність емоційного зворотного зв'язку. У переважній більшості існуючих рішень текстова відповідь генерується без урахування емоційного стану користувача, що унеможливорює природну та емпатичну комунікацію. Через це взаємодія з чат-системами залишається суто утилітарною і не відображає реальну динаміку діалогу.

Проблема емоційної нейтральності AI-систем є не лише технічною, а й принципово гуманітарною. Людська комунікація за своєю природою є емоційно навантаженою: навіть найпростіший обмін репліками між двома людьми відбувається в контексті настрою, тривоги, радості, роздратування чи зацікавленості. Ці емоційні сигнали є не другорядними прикрасами розмови, а її структурними елементами, що визначають вибір слів, темп відповіді, рівень деталізації та загальний тон взаємодії. Ігнорування цього виміру перетворює AI-асистент на інструмент, який хоч і може генерувати граматично правильні та семантично точні відповіді, але залишається нечутливим до реального стану людини, що з ним взаємодіє. Результатом такого підходу є те, що навіть технічно досконалі системи сприймаються користувачами як холодні, відсторонені та нездатні до справжнього контакту.

Сучасний стан розвитку мовних моделей та методів обробки природної мови вже надає технічні можливості для подолання цього обмеження. Дослідження в галузі sentiment analysis, emotion recognition та affective computing демонструють, що автоматичне визначення емоційного тону тексту є цілком вирішуваною задачею з достатньо високою точністю для практичного застосування. Водночас більшість комерційних AI-продуктів не інтегрують ці можливості у свої інтерфейси в осмисленому вигляді. Причинами цього є як складність архітектурної інтеграції

двох окремих систем, так і відсутність чіткого розуміння того, яким саме чином результати емоційного аналізу мають бути представлені користувачу, щоб вони несли практичну цінність, а не перетворювались на черговий інтерфейсний шум.

У межах практики було досліджено підхід до реалізації системи EmotiChat, в якій механізм генерації відповідей доповнено компонентом аналізу емоційного контексту кожного повідомлення з подальшим відображенням реакції аватара та збереженням емоційної траєкторії діалогу. Такий підхід дозволяє надати взаємодії з ІІІ більш людиноподібного характеру та зробити емоційний стан розмови видимим і вимірюваним у реальному часі.

Концептуальну основу системи складає ідея про те, що емоційний аналіз не повинен бути прихованим фоновим процесом, результати якого лише опосередковано впливають на поведінку системи. Натомість EmotiChat виносить цей аналіз на поверхню інтерфейсу, роблячи його повноправним елементом взаємодії. Користувач у будь-який момент бачить, яку емоцію система ідентифікувала в його останньому повідомленні, як на цю емоцію реагує AI-компаньйон, як змінювалась загальна тональність розмови від її початку і яка емоція домінувала протягом усього сеансу. Це перетворює EmotiChat не просто на чат-систему з естетичним аватаром, а на середовище, в якому емоційний вимір спілкування стає предметом усвідомленої уваги з боку користувача.

Під час виконання практики основна увага приділялась реалізації алгоритмів визначення емоційного контексту повідомлень, системи реакцій аватара на основі streak-механізму та організації стрімінгової взаємодії між клієнтською і серверною частинами системи. Для цього використовувались Node.js, Express.js, SSE та Emotion API. Особливу роль у системі відіграє SVG-аватар із двадцятьма унікальними виразами обличчя, який автоматично реагує на емоційну тональність діалогу відповідно до визначених категорій та інтенсивності реакцій.

Архітектурне рішення побудоване на принципі мінімальних зовнішніх залежностей при максимальній функціональній насиченості. Серверна частина реалізована на Node.js з використанням Express.js як HTTP-фреймворку і виконує

роль проксі між клієнтом, локальною мовною моделлю через Ollama та Emotion API. Клієнтська частина побудована на Vanilla JavaScript без використання зовнішніх фронтенд-фреймворків, що забезпечує мінімальний час завантаження та повний контроль над логікою відображення. Передача стрімінгових даних реалізована через Server-Sent Events, які дозволяють організувати однонаправлений потік подій від сервера до клієнта без накладних витрат, притаманних WebSocket-з'єднанням. Ця архітектура забезпечує плавне відображення відповіді мовної моделі в режимі реального часу, а також асинхронну доставку результатів емоційного аналізу після завершення генерації повного тексту.

Streak-механізм, реалізований у системі реакцій аватара, є одним із ключових алгоритмічних рішень проєкту. Його суть полягає в тому, що аватар не реагує на кожну нову емоцію незалежно, а відстежує послідовності повторюваних або схожих емоційних станів і підвищує інтенсивність своєї реакції пропорційно до довжини такої послідовності. Це наближає поведінку аватара до природної людської реакції: якщо людина раз посміхнулася, це помітна, але стримана реакція; якщо вона посміхається протягом усього діалогу, очевидно, що їй по-справжньому весело і загальний настрій підноситься значно вище. Така логіка робить поведінку аватара більш правдоподібною і семантично значущою.

Розроблений застосунок розглядається не лише як навчальний проєкт, а і як основа для подальшого розвитку повноцінного інструменту емоційно адаптивної комунікації. У перспективі систему може бути розширено функціоналом адаптації стилю відповідей відповідно до виявленого емоційного стану, що відкриває можливість використання платформи як інструменту психологічної підтримки або персоналізованого навчального середовища. Подальший розвиток платформи може передбачати інтеграцію мультимодального аналізу емоцій, що включатиме не лише текстовий, а й голосовий ввід із визначенням емоційного забарвлення через інтонаційні характеристики мовлення. Система збереження та аналізу довгострокових емоційних патернів могла б дозволити формувати персоналізований профіль користувача та адаптувати поведінку компаньйона

відповідно до його індивідуальних емоційних особливостей. Всі ці можливості органічно випливають із закладеної архітектури і можуть бути реалізовані без кардинальної перебудови існуючого коду, що свідчить про правильність обраних архітектурних принципів на початковому етапі проектування.

1.1 Постановка задачі

Основною вимогою до системи була реалізація механізму, який дозволяє аналізувати емоційний стан користувача та відповіді асистента після кожного обміну повідомленнями без блокування основного потоку стрімінгу та без суттєвого збільшення часу відповіді. Ця вимога є принципово важливою з точки зору сприйняття системи кінцевим користувачем, оскільки будь-яка помітна затримка в отриманні відповіді руйнує відчуття живої та природної розмови, яке є ключовою цільовою характеристикою EmotiChat як продукту.

Задача поєднання стрімінгової генерації тексту та асинхронного емоційного аналізу є нетривіальною з архітектурної точки зору. Стрімінг передбачає, що відповідь мовної моделі надходить до клієнта поступово, частинами, в міру її генерації. Емоційний аналіз, натомість, вимагає наявності повного тексту повідомлення, оскільки визначення емоційного тону є контекстно залежним і не може виконуватись достовірно на основі окремих фрагментів. Це означає, що аналіз емоцій об'єктивно не може починатись до завершення генерації, однак він також не повинен затримувати передачу вже готового тексту клієнту. Вирішення цієї суперечності і стало однією з центральних архітектурних задач проєкту. Обраний підхід передбачає, що сервер спочатку повністю завершує стрімінг відповіді мовної моделі, після чого виконує послідовний аналіз емоцій повідомлення користувача та відповіді асистента через Emotion API, і лише потім надсилає клієнту результати аналізу у вигляді окремої SSE-події типу emotion. Таким чином клієнт отримує повний текст відповіді без жодних затримок, а

оновлення інтерфейсу емоційного контексту відбувається відразу після завершення генерації, не перериваючи і не гальмуючи основний потік даних.

Система повинна забезпечувати стрімінг відповідей локальної мовної моделі, аналіз емоцій обох учасників діалогу через Emotion API, відображення реакцій аватара відповідно до виявленої емоції та її інтенсивності, збереження повного емоційного контексту між сесіями та відновлення стану діалогу при повторному відкритті. При цьому важливим аспектом є забезпечення коректної роботи системи у режимі реального часу без хмарних залежностей.

Вимога щодо відновлення стану діалогу між сесіями передбачає, що система повинна зберігати не лише текстовий вміст повідомлень, а й повний емоційний контекст кожного обміну, включаючи ідентифікатор сесії Emotion API, виявлені емоції обох учасників для кожного повідомлення, значення загальної температури діалогу та дані траєкторії на момент завершення сесії. Це дозволяє користувачу повернутися до будь-якого збереженого діалогу і побачити його емоційну картину в тому стані, в якому вона була на момент останнього повідомлення, що суттєво підвищує цінність збереженої історії порівняно з системами, які зберігають лише текст.

Для реалізації поставленої задачі необхідно було реалізувати серверний проксі до Ollama з підтримкою SSE-стрімінгу, розробити інтеграцію з Emotion API для послідовного аналізу обох учасників діалогу, реалізувати SVG-аватар із двадцятьма виразами обличчя та додатковими візуальними ефектами, розробити систему емоційних реакцій із десятьма категоріями та streak-механізмом, реалізувати панель емоційного контексту з траєкторією, температурою та sparkline-графіком, забезпечити персистентність діалогів із повним збереженням емоційних даних, реалізувати систему налаштувань аватара та кольорових тем інтерфейсу, а також виконати тестування основних компонентів застосунку.

SVG-аватар як окрема задача вимагав розробки повноцінної системи анімацій на основі чистого JavaScript без використання зовнішніх бібліотек. Кожен із двадцяти виразів обличчя є набором параметрів, що описують форму брів, очей,

зіниць, рота, рум'янців та додаткових графічних елементів. Перехід між виразами реалізований через плавну анімацію з інтерполяцією значень атрибутів SVG-елементів, що забезпечує природний візуальний ефект зміни настрою. Окремо реалізовані ефекти idle-анімації, які підтримують ілюзію живого аватара в паузах між повідомленнями: плавне моргання, мікрорухи очей та пульсація свічення навколо голови персонажа.

Для розробки застосунку використовувались Node.js, Express.js, Vanilla JavaScript, CSS3 та SVG. Ollama забезпечив локальний запуск мовної моделі, а Emotion API виконував аналіз емоцій і підтримував контекст сесії між запитами. Вибір на користь Vanilla JavaScript для клієнтської частини був свідомим архітектурним рішенням, що мотивоване прагненням зберегти повний контроль над продуктивністю та поведінкою інтерфейсу. Використання важких фреймворків у застосунку такого масштабу призвело б до невиправданого збільшення обсягу завантажуваних ресурсів та ускладнення налагодження компонентів, що працюють з анімаціями в реальному часі.

1.2 Вимоги до системи

У процесі проходження практики було визначено перелік функціональних та нефункціональних вимог до системи EmotiChat. Особлива увага приділялась реалізації механізму емоційного аналізу в реальному часі та забезпеченню повного відновлення стану діалогу між сесіями. Процес формування вимог відбувався ітеративно: початковий перелік, визначений на основі концептуального опису системи, уточнювався і доповнювався в міру реалізації окремих компонентів та виявлення технічних обмежень і можливостей обраного стеку.

Функціональні вимоги охоплюють усі аспекти взаємодії користувача із системою. Система повинна надавати можливість вибору локальної мовної моделі з переліку встановлених в Ollama моделей, що підвантажується динамічно при запуску застосунку. Це забезпечує гнучкість у виборі балансу між якістю

відповідей та швидкодією в залежності від апаратних можливостей пристрою користувача. Система повинна передавати відповідь мовної моделі на клієнт у потоковому режимі з відображенням кожного нового токена в інтерфейсі без очікування повної відповіді, що забезпечує суб'єктивне відчуття миттєвої реакції системи. Після завершення генерації відповіді система повинна виконати аналіз емоцій повідомлення користувача, а потім аналіз відповіді асистента через Emotion API з передачею ідентифікатора сесії для збереження контексту між запитами, і надіслати результати аналізу клієнту у вигляді окремої події.

На стороні клієнта система повинна відображати виявлену емоцію у вигляді анімованої зміни виразу обличчя SVG-аватара з відповідним підписом та кольоровим забарвленням, що відповідає емоційній категорії. Система реакцій повинна враховувати streak-послідовності, збільшуючи інтенсивність анімацій при повторенні схожих емоцій, і спадати до базового стану при зміні емоційного тону. Панель емоційного контексту повинна відображати домінуючу емоцію поточного сеансу, траєкторію зміни емоційного тону від першого до останнього повідомлення, показник волатильності діалогу та sparkline-графік температурних значень, отриманих від Emotion API протягом сеансу.

Визначені вимоги відображають основні можливості системи, пов'язані з процесами стрімінгу відповідей, аналізу емоцій, відображення реакцій аватара та збереження діалогів, а також демонструють вимоги до забезпечення автономної роботи без хмарних сервісів, підтримки конкурентного відображення даних у реальному часі та можливості подальшого розширення функціональності. Вони формують основу архітектури застосунку та визначають ключові принципи його функціонування.

Нефункціональні вимоги стосуються якісних характеристик роботи системи. Час між надсиланням повідомлення і початком відображення першого токена відповіді не повинен перевищувати значення, прийнятного для сприйняття як миттєвої реакції, що залежить від продуктивності мовної моделі, але не повинно збільшуватись через затримки на стороні серверного проксі. Інтерфейс повинен

залишатись повністю інтерактивним протягом усього процесу генерації відповіді та виконання емоційного аналізу, не блокуючи взаємодію користувача з елементами управління. Система повинна коректно обробляти ситуації недоступності Ollama або Emotion API, відображаючи відповідні повідомлення про помилки без аварійного завершення роботи та без втрати вже збережених даних діалогів. Збереження та завантаження діалогів повинно відбуватись без помітних затримок навіть за наявності значної кількості збережених сесій та великого обсягу даних у кожній з них.

Вимога до автономності роботи без хмарних залежностей є наскрізною і впливає на всі архітектурні рішення системи. Єдиними зовнішніми сервісами, від яких залежить EmotiChat, є Ollama та Emotion API, обидва з яких можуть функціонувати локально на тому самому пристрої, що і сам застосунок. Це означає, що після початкового налаштування система здатна повноцінно працювати в повністю ізольованому середовищі без доступу до інтернету, що є принциповою перевагою з точки зору конфіденційності та надійності у порівнянні з хмарними платформами, робота яких цілком залежить від стабільності зовнішньої інфраструктури.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

2.1 Порівняльний аналіз із системою Replika

Replika є одним із найвідоміших представників категорії емоційних AI-компаньйонів і з моменту свого виходу на ринок у 2017 році здобула значну популярність серед користувачів, які шукають можливість емоційної підтримки та усвідомленого спілкування з штучним інтелектом. Платформа позиціонує себе як персональний щоденник і водночас як друга, здатного підтримати розмову в будь-який час доби. Проте глибший технічний і функціональний аналіз цього рішення дозволяє виявити суттєві обмеження, які роблять Replika менш привабливою порівняно з розробленим застосунком EmotiChat.

Першою і, мабуть, найбільш критичною проблемою Replika є повна залежність від хмарної інфраструктури компанії-розробника. Усі повідомлення користувача передаються на зовнішні сервери, де обробляються моделями, до яких кінцевий користувач не має жодного доступу та контролю. Це породжує серйозні питання щодо конфіденційності особистих даних, адже розмови з AI-компаньйоном нерідко торкаються вкрай чутливих тем, включаючи особисті переживання, психологічні труднощі та інтимні деталі повсякденного життя. EmotiChat, натомість, побудований на принципі повної локальності: мовна модель запускається безпосередньо на пристрої користувача через Ollama, а отже жодне повідомлення не покидає межі локального середовища. Це не просто технічна перевага, а фундаментальна різниця у філософії роботи з персональними даними.

Другим значущим недоліком Replika є закритість і непрозорість використовуваних моделей. Користувач не може вибрати, яка саме мовна модель обробляє його запити, не може змінити параметри генерації тексту, не може адаптувати поведінку системи під власні потреби. Replika пропонує певний рівень персоналізації через налаштування «особистості» компаньйона, однак ці налаштування є поверхневими і не стосуються базового механізму генерації

відповідей. EmotiChat реалізує принципово інший підхід: завдяки інтеграції з Ollama користувач отримує можливість у будь-який момент змінити мовну модель через випадаючий список у шапці інтерфейсу, обираючи між різними варіантами залежно від потужності свого пристрою та бажаного стилю спілкування. Ця архітектурна відкритість надає EmotiChat гнучкості, яка у Replika принципово недосяжна.

Третім суттєвим обмеженням Replika є монетизаційна модель, яка закриває значну частину функціональності за платним підписком. Емоційний аналіз, голосові повідомлення, розширені режими спілкування та доступ до більш «розумних» версій компаньйона доступні лише підписникам преміального плану. Окрім того, у 2023 році компанія без попередження відключила для існуючих користувачів низку функцій, що спричинило широкий резонанс і підтвердило нестабільність хмарних платформ, де правила гри встановлює і може в будь-який момент змінити виключно компанія-розробник. EmotiChat є повністю відкритим рішенням без жодних платних рівнів доступу: весь функціонал, включаючи аналіз емоцій у реальному часі, анімований аватар, хронологію реакцій та збереження діалогів, доступний одразу після запуску застосунку без будь-яких обмежень і без ризику раптового відключення функцій через комерційні рішення третьої сторони.

Нарешті, Replika не надає користувачу жодної прозорості інформації про те, як саме система інтерпретує емоційний контекст розмови. Компаньйон реагує на тональність повідомлень, однак ці реакції є непрямими і не виносяться в окремий аналітичний шар. EmotiChat реалізує повністю прозору систему емоційного аналізу: панель емоційного контексту в режимі реального часу відображає домінуючу емоцію, траєкторію розмови від початку до поточного моменту, показник волатильності та температурний графік усього діалогу. Це перетворює спілкування не лише на інтерактивний процес, а й на інструмент самоспостереження та рефлексії.

2.2 Порівняльний аналіз із системою Character.AI

Character.AI представляє собою платформу іншого типу порівняно з Replika: якщо Replika акцентує увагу на єдиному персоналізованому компаньйоні, то Character.AI пропонує екосистему персонажів, створених як командою розробників, так і самими користувачами. Платформа здобула колосальну аудиторію завдяки можливості спілкуватися з персонажами з популярних медіафраншиз, вигаданими особистостями та різноманітними AI-ботами. Попри безперечну масштабність і різноманітність пропозиції, Character.AI має ряд принципових обмежень, які дозволяють EmotiChat зайняти якісно інше місце в ніші емоційного AI-спілкування.

Центральною проблемою Character.AI з точки зору конфіденційності є те, що платформа збирає та зберігає величезні обсяги даних про поведінку користувачів, їхні уподобання та вміст розмов. Ця інформація використовується для навчання і вдосконалення моделей, що означає: кожен діалог на платформі потенційно стає частиною тренувального датасету. У 2024 році платформа опинилася в центрі судових позовів, пов'язаних зі шкодою, завданою неповнолітнім користувачам внаслідок нерегульованого емоційного контенту, що поставило під серйозний сумнів відповідальність платформи за безпеку взаємодії. EmotiChat функціонує виключно в локальному середовищі, де жодні дані не збираються, не передаються і не використовуються для жодних сторонніх цілей. Усі розмови зберігаються в пам'яті браузера на пристрої користувача і не залишають його межі, що гарантує абсолютну приватність.

Технічна архітектура Character.AI орієнтована на масштаб і широту охоплення, а не на глибину аналізу окремої розмови. Платформа не надає жодних інструментів для аналізу емоційного стану користувача або емоційної динаміки діалогу. Система може відображати певну емоційну забарвленість у відповідях персонажів, але це є частиною рольової гри, а не результатом реального аналізу тексту. EmotiChat інтегрує спеціалізований Emotion API, який після кожного

обміну повідомленнями виконує повноцінний аналіз тексту обох учасників розмови, визначає емоційний стан як користувача, так і AI, обчислює загальну температуру діалогу і фіксує траєкторію емоційних змін протягом усього сеансу. Це перетворює EmotiChat на унікальний інструмент, який не просто розважає, а надає реальну інформацію про емоційний стан комунікації.

Важливим аспектом є також якість і природність самого спілкування. Character.AI налаштований на рольову взаємодію та виконання персонажних завдань, що суттєво обмежує його ефективність як інструменту щирого, людського діалогу. Відповіді нерідко є шаблонними і підпорядкованими заданому персонажному образу, а не дійсному емоційному контексту розмови. EmotiChat налаштований на протилежну мету: системний промпт мовної моделі орієнтований на максимально природне, неформальне спілкування у стилі живої людської переписки, з урахуванням емоційного тону, гумору, емпатії та реакції на настрій співрозмовника. Модель отримує чіткі інструкції дзеркалити емоційний стан користувача, що у поєднанні з реальним емоційним аналізом створює ефект справжньої, а не симульованої емоційної присутності.

Окремо варто зазначити, що Character.AI, як і Replika, повністю залежить від безперервного функціонування хмарної інфраструктури. Будь-яка недоступність серверів, технічні збої або рішення компанії обмежити доступ одразу виводять платформу з ладу для всіх користувачів. EmotiChat не має цієї вразливості: застосунок працює незалежно від стану інтернет-з'єднання і зовнішніх сервісів, а єдиною умовою роботи є наявність запущеного локального сервісу Ollama на машині користувача.

2.3 Порівняльний аналіз із системою Kindroid

Kindroid є відносно новим гравцем на ринку AI-компаньйонів і позиціонує себе як більш просунуту альтернативу Replika з акцентом на кастомізацію зовнішнього вигляду аватара та глибоку персоналізацію особистості компаньйона.

Платформа дозволяє користувачам детально налаштовувати характер, манеру мовлення, зовнішність та поведінкові паттерни свого AI-компаньйона, використовуючи для цього розгорнуті текстові описи та приклади діалогів. Незважаючи на те, що Kindroid пропонує більш зрілий підхід до кастомізації порівняно з конкурентами, він має низку принципових обмежень, які EmotiChat долає завдяки своїй архітектурі та концептуальному підходу.

Перш за все, Kindroid залишається виключно хмарним продуктом, що означає всі ті самі проблеми з конфіденційністю, що характерні для Replika та Character.AI. Персональні описи компаньйона, вміст розмов і дані про поведінку користувача зберігаються та обробляються на серверах компанії. Для аудиторії, яка використовує AI-компаньйона для щирого та відкритого спілкування, це є неприйнятним з точки зору приватності. EmotiChat вирішує цю проблему радикально: вся обробка відбувається локально, і навіть технічно неможливо перехопити або зберегти дані розмови поза межами пристрою користувача.

Kindroid пропонує статичні або слабко анімовані зображення аватара, згенеровані через AI-зображення на основі текстового опису. Ці аватари є естетично привабливими, однак вони не пов'язані з реальним перебігом розмови і не змінюються залежно від емоційного контексту діалогу. Аватар у Kindroid є декоративним елементом, а не функціональним компонентом системи. EmotiChat реалізує принципово інший підхід: SVG-аватар є динамічним і реактивним елементом інтерфейсу, який у реальному часі змінює вираз обличчя, жести, колір свічення та анімацію відповідно до результатів аналізу емоцій. Брови, очі, рот, рум'янець, додаткові графічні елементи та загальний колірний тон персонажа адаптуються до кожної нової емоції, перетворюючи аватар на живий індикатор емоційного стану розмови, а не просто картинку.

Система аналізу емоцій у Kindroid, якщо і присутня, є імпліцитною і не виноситься в окремий інтерфейсний шар. Користувач не отримує жодної структурованої інформації про те, як змінювався емоційний тон протягом розмови, яка емоція домінувала в діалозі, як корелюють емоційні стани обох учасників.

EmotiChat надає всю цю інформацію у вигляді окремої панелі емоційного контексту, яка містить блок домінуючої емоції, траєкторію змін від початку до кінця сеансу з відображенням початкової та поточної емоції і стрілкою напрямку, показник волатильності розмови у вигляді прогресбару та спарклайн-графік температурної динаміки всього діалогу. Такий рівень аналітичної прозорості у жодному з розглянутих аналогів не реалізований.

3 ЗАСОБИ РОЗРОБКИ

3.1 Серверна частина застосунку

Серверна частина EmotiChat реалізована на платформі Node.js - середовищі виконання JavaScript поза браузером, побудованому на рушії V8 від Google. Node.js використовує асинхронну, неблокуючу модель введення-виведення, що робить його особливо ефективним для застосунків, які обробляють велику кількість одночасних з'єднань із відносно невеликим обчислювальним навантаженням. У контексті EmotiChat це критично важливо: сервер одночасно підтримує стрімінгове з'єднання з клієнтом через SSE та відправляє HTTP-запити до двох зовнішніх сервісів (Ollama та Emotion API), не блокуючи при цьому основний потік виконання.

Для організації маршрутизації та обробки HTTP-запитів використовується фреймворк Express.js. Він забезпечує мінімалістичний, але функціональний рівень абстракції над вбудованим модулем http Node.js. У проєкті Express виконує кілька ролей: обробляє POST-запити на маршруті /api/chat для ведення діалогу, GET-запити на /api/models для отримання переліку доступних моделей, а також роздає статичні файли клієнтської частини через вбудований middleware express.static. Архітектура Express дозволяє легко розширювати функціональність через ланцюжок middleware-функцій, що спрощує подальший розвиток проєкту.

Ключовою особливістю реалізації є використання технології Server-Sent Events (SSE) для передачі відповіді моделі в реальному часі. На відміну від звичайного HTTP-запиту, де клієнт отримує відповідь цілком після її формування, SSE дозволяє серверу відправляти дані частинами у міру їх надходження.

Конфігурація застосунку винесена у файл .env і завантажується за допомогою бібліотеки dotenv. Це дозволяє змінювати адреси зовнішніх сервісів, порт сервера та назву моделі без модифікації коду, що є хорошою практикою при розгортанні в різних середовищах.

3.2 Мовна модель та аналіз емоцій

3.2.1 Локальна мовна модель Ollama

Для генерації відповідей у ролі віртуального співрозмовника використовується Ollama - інструмент для локального запуску великих мовних моделей. Ollama надає уніфікований REST API, що дозволяє взаємодіяти з різними моделями за допомогою однакового інтерфейсу. Застосунок звертається до ендпоінту `/api/chat` з переліком повідомлень діалогу та системним промптом, який задає поведінкову роль моделі.

Вибір Ollama як основи для мовної складової EmotiChat обумовлений кількома принциповими характеристиками цього інструменту. Ollama забезпечує повністю локальне виконання мовної моделі, що означає відсутність будь-яких зовнішніх мережевих залежностей у процесі генерації відповідей. Усі обчислення виконуються безпосередньо на процесорі або відеокарті пристрою користувача, а жодне повідомлення не залишає межі локального середовища. Ця характеристика є принципово важливою для застосунку, орієнтованого на особисте і відверте спілкування, де питання конфіденційності стоїть особливо гостро. Ollama також абстрагує складність завантаження, квантизації та запуску моделей, надаючи єдиний уніфікований інтерфейс для роботи з десятками різних архітектур і розмірів моделей. Для EmotiChat це означає, що зміна базової мовної моделі не вимагає жодних змін у коді застосунку і виконується через єдину змінну середовища `OLLAMA_MODEL`.

Системний пром프트 є центральним інструментом формування особистості та поведінки AI-компаньйона в EmotiChat. Він налаштовує модель на поведінку реального людського друга: відповіді мають бути неформальними, емоційно забарвленими, короткими та природними, без використання markdown-розмітки, структурованих переліків або академічного стилю викладу. Пром프트 містить явну інструкцію дзеркалити емоційний тон користувача: якщо людина пише радісно і збуджено, модель відповідає з відповідним ентузіазмом; якщо людина сумна або

пригнічена, модель переходить до теплого і емпатичного тону; якщо людина жартує, модель підхоплює жарт. Ця інструкція є критично важливою для узгодженості між текстовою реакцією моделі та візуальною реакцією аватара, оскільки обидва компоненти орієнтовані на одну і ту саму емоційну тональність повідомлення.

Окремою і принципово важливою частиною системного пром프트 є інструкція відхиляти будь-які запити, пов'язані з написанням коду, технічними поясненнями або структурованими інструкціями. Ця вимога впливає з концептуального позиціонування EmotiChat як соціального, а не технічного інструменту. Модель отримує чітку вказівку реагувати на такі запити так, як реагував би друг, який просто не зацікавлений у технічних темах: неформально відмовитись і перевести розмову в інше русло. Такий підхід підтримує цілісність соціального образу компаньйона і запобігає розриву між очікуваним і реальним характером взаємодії. За замовчуванням використовується модель Llama 3.2, однак через змінну середовища OLLAMA_MODEL можна підключити будь-яку іншу модель, доступну в локальному екземплярі Ollama.

Стрімінг відповіді реалізований через читання тіла HTTP-відповіді від Ollama частинами за допомогою ReadableStream та TextDecoder. Ollama при стрімінгу повертає послідовність JSON-рядків, кожен з яких містить черговий фрагмент відповіді у полі message.content. Сервер EmotiChat читає цей потік побайтово, акумулює отримані дані в буфері, розділяє їх по символу нового рядка і обробляє кожен повний рядок як окремий JSON-об'єкт. Витягнутий текстовий фрагмент негайно передається клієнту через SSE-канал у вигляді події типу text, не чекаючи завершення генерації всього тексту. Паралельно з цим сервер акумулює повний текст відповіді у змінній fullText, яка буде використана після завершення стрімінгу для передачі в Emotion API.

Такий підхід до стрімінгу забезпечує важливу перевагу з точки зору сприйняття: користувач починає читати відповідь практично одразу після її відправки, ще в процесі генерації. Це кардинально змінює суб'єктивне відчуття

швидкості роботи системи. Навіть якщо генерація відповіді займає кілька секунд, відчуття очікування зникає, оскільки інтерфейс одразу починає наповнюватись текстом. Саме ця характеристика є однією з ключових для підтримки ілюзії живої розмови, на якій побудована концепція EmotiChat.

Управління доступними моделями в інтерфейсі EmotiChat реалізовано через окремий маршрут GET `/api/models`, який виконує запит до ендпоінту `/api/tags` локального екземпляра Ollama і повертає список назв усіх встановлених моделей. Цей список підвантажується при кожному відкритті застосунку і наповнює випадаючий список у шапці інтерфейсу. Вибрана модель передається на сервер у кожному запиті до `/api/chat`, що дозволяє динамічно змінювати модель між розмовами без перезапуску сервера. Такий підхід надає користувачу гнучкість у виборі балансу між якістю і швидкістю відповідей відповідно до поточних потреб і апаратних можливостей пристрою.

3.2.2 Інтеграція з Emotion API

Emotion API є зовнішнім мікросервісом, що виконує аналіз емоційного стану учасників діалогу. Після завершення генерації відповіді моделі сервер послідовно відправляє до Emotion API два запити: перший аналізує повідомлення користувача, другий - відповідь асистента. Обидва запити прив'язані до спільного `sessionId`, що дозволяє сервісу враховувати контекст усього діалогу при класифікації кожного нового повідомлення.

Послідовність двох запитів є принципово важливою для коректної роботи контекстного аналізу. Перший запит передає до Emotion API текст повідомлення користувача з параметром `speaker` зі значенням `User`. Emotion API обробляє цей текст в контексті всього попереднього діалогу в межах поточної сесії і повертає розпізнану емоцію разом із оновленим `sessionId`. Цей `sessionId` використовується у другому запиті, що передає текст відповіді асистента з параметром `speaker` зі значенням `Assistant`. Таким чином Emotion API має повну картину обміну: він

бачить не лише останнє повідомлення, а й те, на яке повідомлення воно є реакцією, що суттєво підвищує точність контекстного аналізу порівняно з незалежною класифікацією окремих фрагментів тексту.

Рішення виконувати аналіз емоцій після завершення стрімінгу, а не паралельно з ним, є архітектурним компромісом між точністю і часом відповіді. З одного боку, аналіз повного тексту завжди точніший за аналіз фрагментів: емоційний тон фрази може суттєво змінюватись залежно від її завершення, а переривання на довільному токени призводить до аналізу неповного і потенційно хибного фрагмента. З іншого боку, очікування завершення генерації перед початком аналізу означає, що клієнт отримає результати емоційного аналізу з певною затримкою відносно завершення тексту. На практиці ця затримка є мінімальною і складається виключно з часу двох HTTP-запитів до Emotion API, що виконуються послідовно. Оскільки обидва сервіси зазвичай запуснені локально, затримка мережі є нехтовно малою і не помітна для користувача.

Відповідь Emotion API містить розпізнану емоцію для кожного повідомлення, а також агреговані метрики сесії: загальну емоційну температуру у полі `overallTemperature`, домінуючу емоцію діалогу у полі `dominantDialogueEmotion` та вектор розвитку емоційного стану у полі `trajectory`. Ці дані відправляються клієнту окремою SSE-подією типу `emotion` після завершення текстової відповіді.

Поле `overallTemperature` є числовим показником загальної емоційної інтенсивності діалогу, що акумулює значення з усіх попередніх повідомлень сесії. Це значення використовується в EmotiChat для побудови `sparkline`-графіка на панелі емоційного контексту: кожне нове значення температури додається до масиву точок графіка, що дозволяє користувачу спостерігати динаміку емоційного накалу розмови у вигляді візуальної кривої. Поле `dominantDialogueEmotion` відображає емоцію, яка найчастіше зустрічалась або мала найбільшу сумарну інтенсивність протягом усієї сесії. Це значення виводиться у верхній частині панелі емоційного контексту як ключова характеристика поточного діалогу. Поле

trajectory описує напрямок зміни емоційного стану від початку сесії до її поточного моменту і використовується для відображення стрілки тренду у блоці траєкторії, де поруч відображаються початкова і поточна емоції разом з показником волатильності.

Клієнтська частина EmotiChat обробляє отриману подію типу emotion у обробнику SSE-потoku в chat.js, після чого передає розпізнані емоції до reactions.js для вибору відповідного виразу обличчя аватара і оновлює всі елементи панелі емоційного контексту. Розпізнана емоція користувача і розпізнана емоція асистента обробляються окремо: перша впливає переважно на вибір виразу аватара через streak-механізм, тоді як друга використовується поряд із першою для формування повної картини емоційного обміну в хронологічному рядку реакцій. Хронологічний рядок у лівій панелі відображає іконки емоцій у порядку їх появи протягом сесії і дозволяє побачити загальний емоційний ритм розмови одним поглядом.

Якщо Emotion API недоступний, сервер перехоплює помилку, логує її у консоль і відправляє клієнту подію типу emotion_error з текстом помилки, не перериваючи основний потік діалогу. На стороні клієнта отримання такої події не спричиняє жодних видимих помилок в інтерфейсі: аватар залишається у поточному стані, панель емоційного контексту не оновлюється, а розмова продовжується в звичайному режимі. Такий підхід до обробки відмов забезпечує відмовостійкість системи: емоційний аналіз є збагачуючим шаром поверх основного функціоналу чату, а не його критичною залежністю, тому недоступність Emotion API деградує функціональність, але не робить застосунок непрацездатним.

3.3 Клієнтська частина

3.3.1 HTML5 та структура інтерфейсу

Клієнтська частина побудована на чистому HTML5 без використання фреймворків. Структура сторінки поділена на три логічні колонки: ліва панель містить аватар та хронологію емоційних реакцій, центральна - основний чат з полем введення, права - емоційний контекст поточної сесії. Аватар розміщено у лівій панелі і реалізований як вбудований SVG-документ, що дозволяє маніпулювати його елементами безпосередньо через JavaScript DOM API.

Відмова від будь-якого фронтенд-фреймворку є свідомим архітектурним рішенням. Більшість оновлень інтерфейсу є точковими маніпуляціями конкретними DOM-елементами, а не повторним рендерингом дерева компонентів, тому реактивний підхід тут не дає переваг. Пряма робота з DOM API надає повний контроль над продуктивністю анімацій: критичні операції з SVG-атрибутами виконуються без посередників і без накладних витрат, що є принципово важливим для плавності анімацій аватара в реальному часі.

SVG-аватар складається з набору іменованих елементів: контур обличчя, очі у вигляді еліпсів, брови у вигляді кривих Безьє, рот у вигляді SVG-шляху, елементи рум'янців, сльозинок, символів сердечок та парових хмарин. Кожен елемент має унікальний ідентифікатор, що дозволяє JavaScript-коду точково змінювати їх атрибути для відтворення конкретного виразу обличчя. Група `extraLayer` призначена для динамічного додавання ситуативних візуальних ефектів - сліз, сердечок, пари - що генеруються JavaScript-кодом та вставляються як `innerHTML`.

Структура SVG-документа аватара спроектована з урахуванням ієрархії шарів. Задній шар капелюха розміщений до елементів обличчя і очей, передній - після них, що дозволяє коректно відображати аксесуари, які частково перекривають верхню частину голови персонажа. Ефект свічення навколо голови реалізований через напівпрозорий еліпс, колір якого відповідає поточному

акцентному кольору персонажа і змінюється разом із ним при застосуванні налаштувань.

3.3.2 JavaScript та система анімацій

Клієнтська логіка розподілена між кількома модулями: `chat.js` відповідає за надсилання повідомлень, відображення тексту та управління історією, `avatar.js` керує станом і анімаціями аватара, `reactions.js` містить логіку вибору виразу обличчя на основі розпізнаних емоцій, `idle.js` забезпечує фонові анімації у паузах між повідомленнями, а `settings.js` реалізує логіку панелі налаштувань аватара та кольорових тем.

Такий поділ на модулі забезпечує чітке розмежування відповідальності між компонентами системи. Кожен файл має єдиний чітко визначений обов'язок і взаємодіє з іншими через добре визначені функції та глобальні змінні стану. Відсутність модульної системи в браузерному JavaScript без збирача не є обмеженням в контексті даного застосунку: порядок підключення скриптів у HTML-документі явно визначає залежності між модулями, а невеликий обсяг коду в кожному файлі робить цю структуру достатньо прозорою і зрозумілою.

У `avatar.js` визначено об'єкт `EXPRESSIONS` із двадцятьма записами - по одному на кожен вираз обличчя. Кожен запис описує повний стан аватара: колір обличчя та очей, масштаб очей по вертикалі, шляхи бровей та рота, наявність рум'янців, поту, сонливості, заповнення рота, язика та вмісту `extraLayer`. Функція `applyExpression(key)` зчитує відповідний запис і послідовно оновлює атрибути SVG-елементів, CSS-змінні кольорів та `opacity` декоративних шарів. Для емоцій гніву та люті додатково запускається CSS-анімація тремтіння через маніпуляцію класом елемента.

Двадцять виразів обличчя охоплюють широкий емоційний спектр і згруповані за десятьма категоріями, кожна з яких має два рівні інтенсивності. Категорія позитивних емоцій включає вирази від спокійного задоволення до відкритої радості зі сміхом. Категорія смутку охоплює легку меланхолію та

виражений сум зі сльозами. Категорія гніву прогресує від збентеження через дискомфорт і роздратування до відкритого гніву та крайнього обурення. Категорія здивування представлена двома рівнями від легкого подиву до шокованої реакції. Категорія ніяковості включає сором'язливість із рум'янцями та збентеженість. Додатково реалізовані категорії закоханості, втоми, задумливості та нейтрального стану очікування. Такий охоплення дозволяє системі підбирати адекватний вираз для переважної більшості емоційних ситуацій, що виникають у природній розмові.

Переходи між виразами обличчя реалізовані через CSS-властивість `transition` на ключових SVG-атрибутах, що забезпечує плавну анімацію зміни форми. Зокрема, застосовуються сплайн-інтерполяція для шляхів та лінійна інтерполяція для числових властивостей кольору і масштабу. Тривалість переходів підібрана таким чином, щоб зміна виразу відчувалась природною: занадто короткий перехід виглядає як різкий стрибок, тоді як занадто довгий - сповільнює сприйняття реакції аватара і порушує відчуття синхронності з перебігом розмови. Обраний баланс забезпечує візуально плавну, але помітно швидко зміну настрою, що відповідає природній людській міміці.

Idle-анімації, реалізовані у модулі `idle.js`, вирішують важливу задачу підтримки відчуття присутності аватара під час пауз у розмові. Без фонових анімацій аватар перетворювався б на статичне зображення, що руйнує ілюзію живого персонажа. Idle-система включає автоматичне моргання з випадковим інтервалом у діапазоні, типовому для людини, мікрорухи зіниць у межах очей, що імітують природній погляд, а також поступове повернення до нейтрального виразу після тривалої відсутності повідомлень. Таймер idle-стану скидається при кожному новому повідомленні і запускається знову після визначеного часу бездіяльності, що забезпечує коректну поведінку в різних сценаріях використання.

Система реакцій у `reactions.js` використовує streak-лічильник: якщо протягом кількох повідомлень підряд визначається одна й та сама категорія емоцій, вираз обличчя ескалюється по ланцюжку інтенсивності. Наприклад, при категорії `anger` послідовність виглядає так: `confused` → `uncomfortable` → `irritated` → `angry` →

steaming. Це дозволяє аватару відображати накопичення емоцій, а не реагувати різкими стрибками на кожне повідомлення окремо.

Streak-механізм також передбачає логіку спаду: при зміні емоційної категорії лічильник скидається і наступна реакція розпочинається з першого рівня інтенсивності відповідної категорії. Таким чином аватар ніколи не застрягає на високому рівні однієї емоції при переході до іншої, що забезпечує коректне відображення швидких змін настрою в динамічних розмовах. Додатково реалізований поріг мінімальної інтенсивності, нижче якого зміна виразу не відбувається: якщо Emotion API повертає емоцію з дуже низьким значенням впевненості, система зберігає поточний вираз, уникаючи невиправданих мікрозмін міміки, які знижували б загальну якість сприйняття аватара.

3.3.3 CSS3 та візуальне оформлення

Стилізація застосунку реалізована за допомогою CSS3 з використанням кастомних властивостей для централізованого управління кольоровою палітрою. Загальна стилістика витримана в темній гамі з акцентами у фіолетово-синіх тонах, що відповідає концепції нічного, особистого простору для спілкування. Ефект скляних панелей досягається поєднанням напівпрозорих фонів з каналом прозорості, розмиття фону через backdrop-filter та тонких рамок із низькою непрозорістю. Цей підхід надає інтерфейсу візуальної глибини, при якій панелі сприймаються як об'єкти, що ширяють над фоновим шаром, а не як плоскі прямокутники з однорідним заповненням.

Система кольорових тем реалізована через набір CSS-змінних, визначених на рівні кореневого елемента документа. Перемикання між темами здійснюється шляхом заміни значень цих змінних через JavaScript, що миттєво перефарбовує всі елементи інтерфейсу без перезавантаження сторінки та без необхідності підключати окремі CSS-файли для кожної теми. Така архітектура теміфікації є ефективнішою порівняно з підходом на основі класів-модифікаторів, оскільки не

потребує дублювання правил стилів і забезпечує гарантовану узгодженість кольорів у всіх компонентах при будь-якій активній темі.

Адаптивність інтерфейсу забезпечується flexbox-розкладкою та медіа-запитами. Анімації реалізовані через @keyframes для циклічних ефектів у вигляді пульсації та тремтіння, а також через transition для інтерактивних переходів стану елементів. Прокрутка повідомлень чату стилізована через overflow-y: auto з кастомним скроллбаром, оформленим у загальній кольоровій схемі. Кастомний скроллбар реалізований через псевдоелементи ::-webkit-scrollbar та відповідні дочірні псевдоелементи, що дозволяє повністю стилізувати його зовнішній вигляд відповідно до загальної кольорової схеми та мінімізувати його візуальну помітність, не прибираючи функціональності прокрутки.

Кожна кольорова тема визначає не лише основні кольори фону та акцентів, а й похідні значення, що використовуються для станів hover, активних елементів та декоративних ефектів. Це забезпечує візуальну когерентність теми в усіх станах інтерактивних елементів, а не лише в їх спокійному стані. Окремо стилізовані компоненти повідомлень чату для повідомлень користувача та асистента: вони розрізняються кольором бульбашки, вирівнюванням та наявністю аватарної іконки, що відповідає загальноприйнятій конвенції дизайну месенджерів і знижує когнітивне навантаження при читанні довгих діалогів.

3.4 Серверна частина застосунку

Для збереження історії діалогів на стороні клієнта використовується Web Storage API, зокрема localStorage. Кожен діалог зберігається як JSON-об'єкт із унікальним числовим ідентифікатором, заголовком що формується з перших слів першого повідомлення, часовою міткою та масивом повідомлень. Поточний sessionId від Emotion API також зберігається між сторінками і відновлюється при завантаженні, що забезпечує неперервний контекст емоційного аналізу в рамках одного діалогу.

Вибір `localStorage` як сховища для персистентності діалогів є обґрунтованим рішенням для застосунку такого масштабу і архітектури. Альтернативою міг бути `IndexedDB`, який надає більш потужні можливості для роботи зі структурованими даними, підтримує транзакції та дозволяє зберігати значно більші обсяги інформації. Однак асинхронний API `IndexedDB` суттєво ускладнює код і є виправданим лише за наявності реальних обмежень `localStorage` або необхідності в складних запитах до даних. У контексті `EmotiChat` обсяг збережених даних є прогнозовано невеликим, запити до сховища є простими операціями читання і запису за ключем, а синхронний API `localStorage` спрощує логіку завантаження стану при ініціалізації застосунку, що робить цей вибір цілком доцільним.

Функції роботи з `localStorage` у вигляді `getSavedConversations`, `saveConversation` та `deleteConversation` інкапсульовані в окрему групу у файлі `chat.js`. Сериалізація та десериалізація даних виконується через `JSON.stringify` та `JSON.parse`. Такий підхід не потребує серверної бази даних, що суттєво спрощує розгортання застосунку: достатньо мати встановлені `Node.js` та `Ollama`.

Серверна частина застосунку реалізована як мінімалістичний `Node.js`-сервер на базі `Express.js`, що виконує дві основні функції: роздачу статичних файлів клієнтської частини та проксіювання запитів до `Ollama` і `Emotion API`. Мінімалізм серверного коду є свідомою характеристикою архітектури, а не наслідком обмежень реалізації. Вся бізнес-логіка, пов'язана з відображенням даних, управлінням станом і взаємодією з користувачем, зосереджена на стороні клієнта, тоді як сервер є суто транспортним шаром, що усуває потребу в складній серверній логіці та базах даних.

Маршрут `POST /api/chat` є центральним ендпоінтом серверної частини. Він приймає від клієнта масив повідомлень поточного діалогу та ідентифікатор сесії, встановлює SSE-з'єднання через відповідні заголовки відповіді, після чого виконує запит до `Ollama` з повним контекстом діалогу та заданим системним промптом. Поточкові дані від `Ollama` зчитуються через `ReadableStream API`, парсяться рядок за рядком і негайно перенаправляються клієнту у вигляді SSE-подій типу `text`, що

забезпечує відображення відповіді на стороні клієнта практично без затримки відносно генерації моделлю. Після завершення стрімінгу сервер послідовно виконує два запити до Emotion API - для повідомлення користувача і для відповіді асистента - і надсилає зведені результати аналізу клієнту у вигляді події типу emotion. Завершення потоку сигналізується службовою подією з маркером DONE.

Маршрут GET /api/models забезпечує динамічне підвантаження переліку встановлених мовних моделей через запит до відповідного ендпоінту Ollama. Отриманий список передається клієнту у вигляді JSON-масиву рядків, на основі якого формується випадаючий список вибору моделі в шапці інтерфейсу. Цей маршрут викликається при кожному завантаженні сторінки, що гарантує актуальність переліку доступних моделей навіть якщо користувач встановив або видалив моделі через Ollama після попереднього запуску застосунку. Конфігурація URL-адрес Ollama та Emotion API, а також порту сервера, винесена у змінні середовища і зчитується через dotenv, що дозволяє адаптувати застосунок до різних середовищ розгортання без зміни коду.

4 РЕАЛІЗАЦІЯ ТА ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі розглядається програмна реалізація EmotiChat, загальна архітектура системи та основні сценарії взаємодії користувача з інтерфейсом застосунку. Наведено опис ключових компонентів системи, механізмів їх взаємодії та принципів відображення інформації в інтерфейсі. Розділ охоплює як структурні аспекти побудови системи, так і конкретні технічні рішення, що були прийняті в процесі реалізації окремих компонентів. Особлива увага приділяється механізму стрімінгової передачі даних, логіці емоційного аналізу та принципам побудови реактивного SVG-аватара, оскільки саме ці компоненти визначають унікальність EmotiChat як системи та відрізняють її від традиційних чат-застосунків.

Опис реалізації системи будується від загального до конкретного: спочатку розглядається архітектура на рівні взаємодії компонентів, після чого детально описується реалізація кожного з них. Такий підхід дозволяє спочатку сформулювати цілісне уявлення про систему, а потім заглибитись у технічні деталі кожної складової без втрати загального контексту.

4.1 Архітектура вебзастосунку

EmotiChat реалізовано як клієнт-серверний вебзастосунок із чітким розподілом відповідальності між складовими системи. Серверна частина побудована на платформі Node.js з використанням фреймворку Express.js і виступає єдиною точкою взаємодії між браузером користувача, локальною мовною моделлю Ollama та зовнішнім сервісом аналізу емоцій Emotion API. Сервер обробляє вхідні HTTP-запити, формує запити до зовнішніх сервісів та повертає результати клієнту через стандартизований інтерфейс. Загальну архітектуру системи наведено на рисунку 4.1.

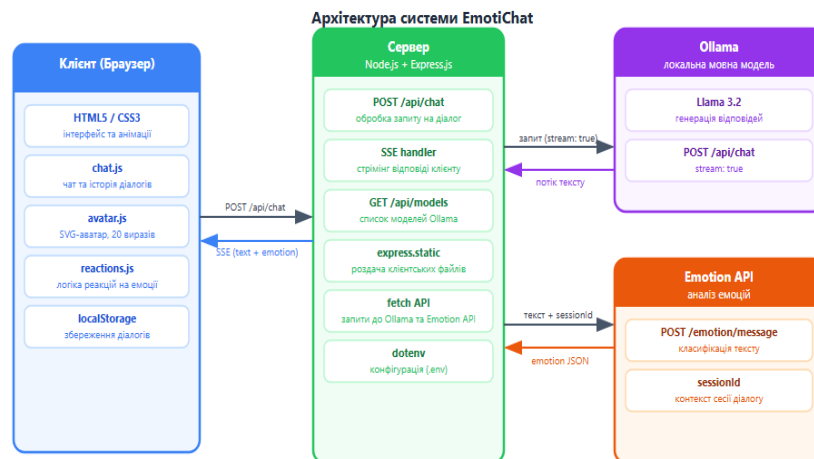


Рисунок 4.1 – Архітектура системи EmotiChat

Клієнтська частина реалізована на чистому HTML5, CSS3 та Vanilla JavaScript без використання додаткових фронтенд-фреймворків, що забезпечує мінімальний час завантаження сторінки та повний контроль над логікою відображення. Логіка клієнта розподілена між кількома спеціалізованими модулями: chat.js відповідає за надсилання повідомлень та управління історією діалогів, avatar.js керує анімаціями та виразами обличчя SVG-аватара, reactions.js містить логіку вибору виразу на основі даних від Emotion API, idle.js забезпечує фонові анімації у паузах між повідомленнями, а settings.js реалізує логіку панелі налаштувань. Зв'язок між клієнтом і сервером під час генерації відповіді здійснюється через технологію Server-Sent Events, яка дозволяє серверу надсилати дані клієнту у вигляді безперервного потоку подій без встановлення постійного двостороннього з'єднання.

Архітектурне рішення розділити систему на три незалежні сервіси - Express-сервер, Ollama та Emotion API - є усвідомленим вибором, що відображає принцип єдиної відповідальності на рівні мікросервісів. Кожен із трьох компонентів має чітко обмежену зону відповідальності і взаємодіє з іншими виключно через HTTP-інтерфейс. Ollama відповідає виключно за генерацію тексту і не знає нічого про емоційний аналіз. Emotion API відповідає виключно за класифікацію емоцій і не

знає нічого про мовну модель. Express-сервер є оркестратором, що послідовно задіює обидва сервіси і повертає клієнту зведений результат. Така ізоляція означає, що будь-який із компонентів може бути замінений альтернативною реалізацією без змін у решті системи: наприклад, Ollama можна замінити будь-яким іншим локальним або віддаленим провайдером мовної моделі, що підтримує сумісний API, а Emotion API можна замінити іншим сервісом аналізу настроїв без жодних змін у клієнтській частині.

Потік даних у системі під час одного обміну повідомленнями відбувається у кілька послідовних фаз. На першій фазі клієнт формує POST-запит до ендпоінту `/api/chat` сервера, передаючи масив повідомлень поточного діалогу, ідентифікатор сесії Emotion API та назву обраної мовної моделі. На другій фазі сервер встановлює SSE-з'єднання з клієнтом, надсилаючи відповідні заголовки, після чого виконує стрімінговий запит до Ollama з повним контекстом діалогу і системним промптом. На третій фазі сервер зчитує потокову відповідь від Ollama і негайно перенаправляє кожен отриманий текстовий фрагмент клієнту у вигляді SSE-події типу `text`, одночасно акумулюючи повний текст відповіді в пам'яті. Після завершення генерації починається четверта фаза: сервер послідовно виконує два POST-запити до Emotion API - перший з текстом повідомлення користувача, другий з текстом відповіді асистента - і надсилає клієнту зведені результати аналізу у вигляді SSE-події типу `emotion`. Завершення потоку сигналізується службовою подією з маркером `DONE`, після отримання якої клієнт закриває SSE-з'єднання і виконує фінальне оновлення інтерфейсу.

На стороні клієнта отримані SSE-події обробляються обробником у `chat.js`, який маршрутизує їх залежно від типу. Події типу `text` додають черговий фрагмент тексту до поточної бульбашки повідомлення асистента в DOM, забезпечуючи ефект покрокового набору тексту. Подія типу `emotion` передається до `reactions.js` для вибору виразу обличчя аватара і до функцій оновлення панелі емоційного контексту. Подія типу `emotion_error` відображає ненав'язливе попередження про недоступність Emotion API, не перериваючи роботу чату. Така подієво-орієнтована

обробка даних на стороні клієнта забезпечує повну незалежність між оновленням текстового вмісту і оновленням емоційних компонентів інтерфейсу, що є ключовою умовою плавної та синхронної роботи всіх частин застосунку.

Конфігурація системи реалізована через змінні середовища, що зчитуються із файлу `.env` при запуску сервера. Параметр `OLLAMA_URL` визначає адресу локального екземпляра Ollama і за замовчуванням вказує на стандартний порт 11434 на локальному хості. Параметр `OLLAMA_MODEL` задає назву мовної моделі, що використовується за замовчуванням при запуску сервера, хоча клієнт може динамічно перевизначати цей параметр для кожного запиту. Параметр `EMOTION_API_URL` визначає базову адресу сервісу аналізу емоцій. Параметр `PORT` задає порт, на якому запускається Express-сервер. Винесення конфігурації у змінні середовища є стандартною практикою, що дозволяє розгортати застосунок у різних середовищах без жодних змін у коді, а також забезпечує безпечне зберігання адрес внутрішніх сервісів поза кодовою базою.

Відсутність серверної бази даних є свідомою архітектурною характеристикою EmotiChat, яка суттєво спрощує розгортання і обслуговування системи. Уся персистентність даних реалізована на стороні клієнта через `localStorage` браузера. Кожен діалог зберігається як JSON-об'єкт із унікальним ідентифікатором, заголовком, часовою міткою, масивом повідомлень та збереженим `sessionId` від Emotion API. При відкритті збереженого діалогу клієнт відновлює не лише текстовий вміст повідомлень, а й повний емоційний контекст сесії, що дозволяє продовжити розмову з тієї точки, де вона була перервана, зі збереженням усіх накопичених метрик. Такий підхід означає, що для повноцінного розгортання EmotiChat достатньо встановити Node.js, Ollama та Emotion API - жодних баз даних, жодних міграцій, жодних серверних процесів збереження стану, що робить систему максимально простою у розгортанні та підтримці.

4.2 Робота користувача з програмною системою

4.2.1 Головний інтерфейс чату

Після відкриття застосунку в браузері користувач потрапляє на єдину сторінку EmotiChat, організовану за принципом триколонкового макету. Ліва панель містить анімований SVG-аватар та хронологічний рядок емоційних реакцій поточного діалогу, центральна область відведена під основний чат з полем введення повідомлень, права панель відображає емоційний контекст поточної сесії у вигляді агрегованих метрик і графіків. Такий розподіл дозволяє тримати всі ключові елементи системи у полі зору одночасно, без необхідності переходу між сторінками. Загальний вигляд головної сторінки представлено на рисунку 4.2.

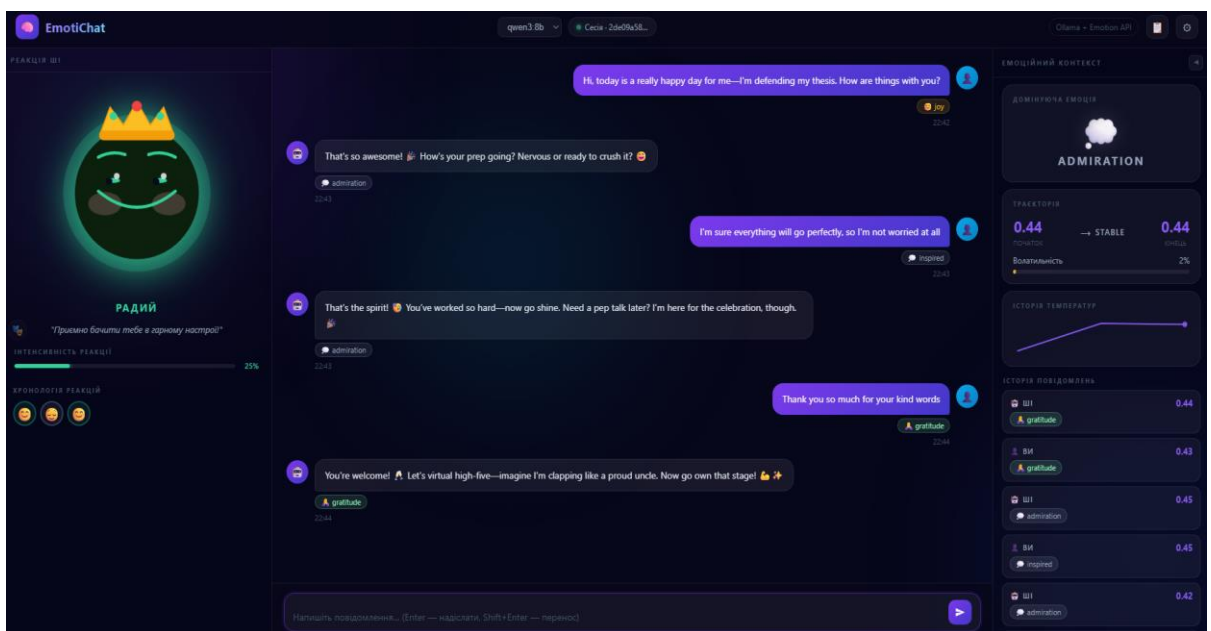


Рисунок 4.2 – Головна сторінка застосунку EmotiChat

Шапка застосунку розміщена у верхній частині сторінки і містить логотип EmotiChat, випадаючий список для вибору активної мовної моделі Ollama, індикатор стану підключення у вигляді кольорової крапки з текстовим підписом, а також кнопки для відкриття панелі збережених діалогів і модального вікна налаштувань аватара. Індикатор стану в реальному часі відображає три можливі

стани системи: очікування введення, активна генерація відповіді та помилка підключення. Це дозволяє користувачу завжди чітко розуміти поточний стан системи, не вдаючись до жодних додаткових дій для перевірки. Випадаючий список моделей наповнюється динамічно при завантаженні сторінки і відображає актуальний перелік усіх встановлених в Ollama моделей без необхідності перезавантаження застосунку. Система кольорових тем інтерфейсу також відображається у загальній стилістиці шапки, підтримуючи візуальну цілісність при перемиканні між темами.

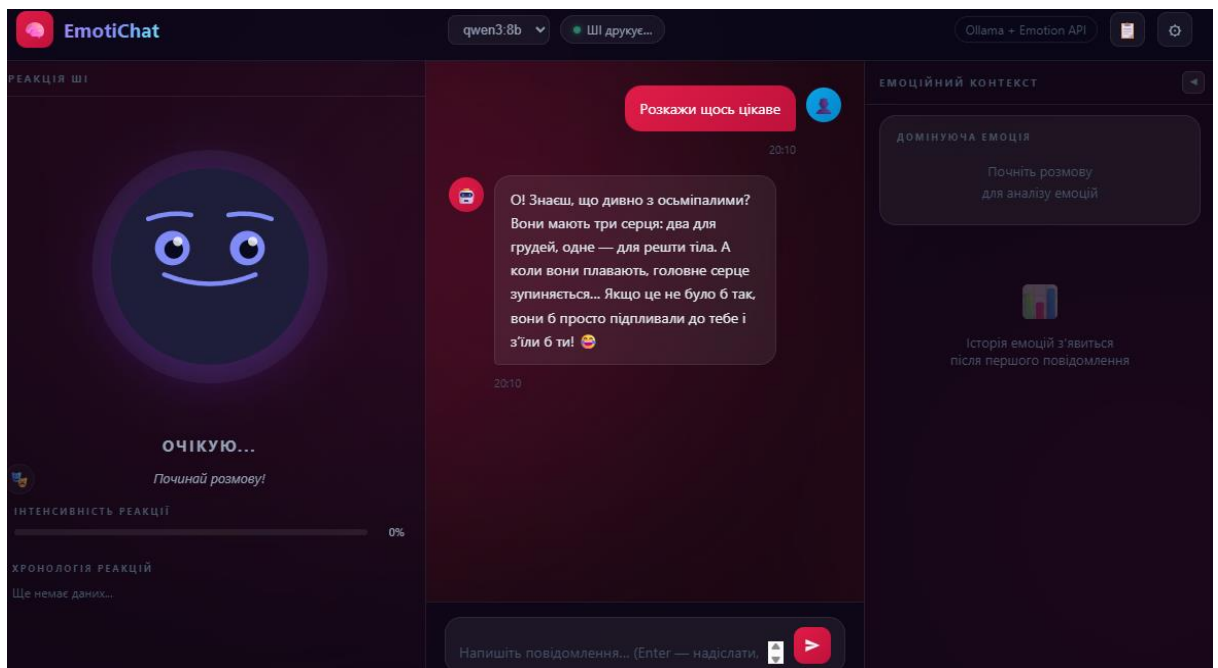


Рисунок 4.3 – Головна сторінка застосунку EmotiChat в трояндовій темі

Центральна колонка при першому відкритті застосунку або після створення нового діалогу відображає вітальний екран, що містить короткий опис можливостей EmotiChat та набір підказок у вигляді клікабельних чіпів із прикладами початкових повідомлень. Ці підказки дозволяють користувачу миттєво розпочати розмову без необхідності самотійно формулювати перше повідомлення і водночас задають тон спілкування, на який орієнтований застосунок. При натисканні на будь-який чіп його текст автоматично вставляється у поле введення і одразу надсилається, що мінімізує кількість дій, необхідних для початку взаємодії.

Для відправлення повідомлення користувач вводить текст у поле внизу центральної колонки та натискає кнопку надсилення або клавішу Enter. Shift+Enter додає перенос рядка без відправки, що дозволяє зручно вводити багаторядкові повідомлення. Поле введення автоматично розширюється по висоті відповідно до обсягу введеного тексту, забезпечуючи зручність роботи як з короткими репліками, так і з розгорнутими повідомленнями. Кнопка надсилення блокується на час генерації відповіді, запобігаючи відправленню нових повідомлень до завершення поточної відповіді і таким чином забезпечуючи цілісність контексту діалогу.

Після надсилення запит надходить до серверного обробника, який формує масив повідомлень разом із системним промптом і передає його до Ollama через HTTP POST-запит із увімкненим стрімінгом. Системний промпт налаштовує модель на роль неформального людського друга, який дзеркалить емоційний тон співрозмовника, відповідає коротко і природно, уникає технічних тем і структурованого академічного стилю. Масив повідомлень включає всю попередню переписку поточного сеансу, що забезпечує збереження контексту розмови між повідомленнями. Сервер зчитує відповідь моделі частинами у міру її генерації та кожен фрагмент тексту негайно надсилає клієнту через SSE-канал у вигляді JSON-події типу text. Завдяки цьому відповідь з'являється на екрані посимвольно в режимі реального часу, не чекаючи завершення генерації всього тексту, що створює природне відчуття живого набору тексту від співрозмовника. Процес потокового виведення відповіді зображено на рисунку 4.4.

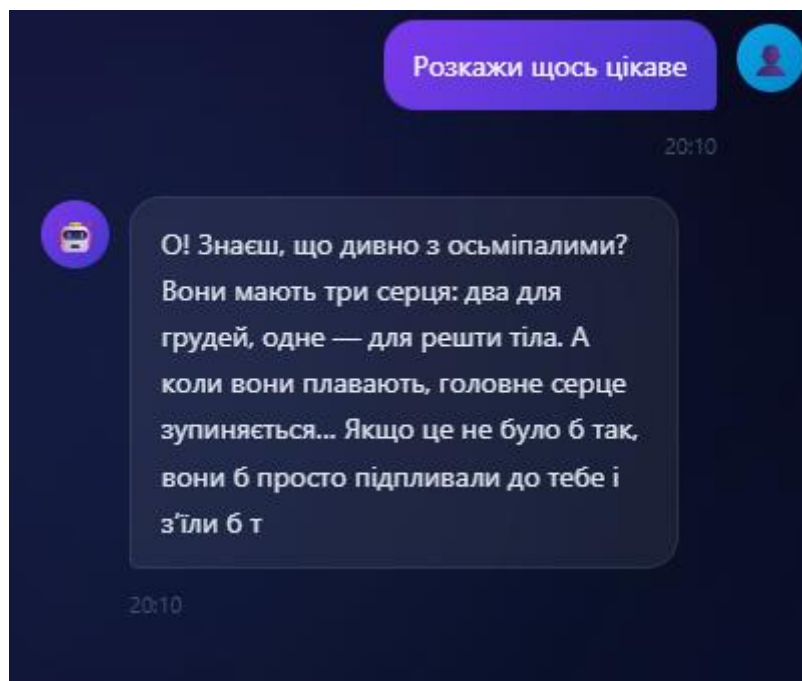


Рисунок 4.4 – Поточний вивід відповіді моделі у чаті

Під час генерації відповіді в стрічці повідомлень спочатку з'являється індикатор друку у вигляді трьох анімованих крапок, що імітує затримку перед початком набору тексту. Як тільки від сервера надходить перший текстовий фрагмент, індикатор зникає і на його місці формується бульбашка повідомлення асистента, яка поступово наповнюється текстом.

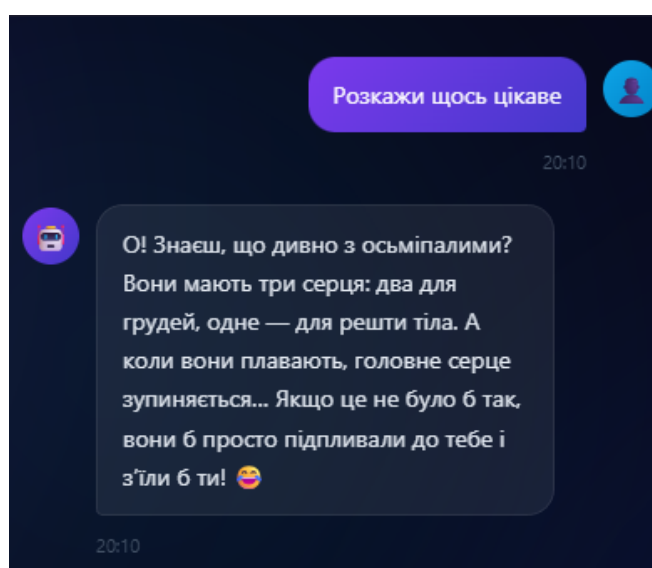


Рисунок 4.5 – Поточний вивід відповіді моделі у чаті

Стрічка повідомлень автоматично прокручується донизу при появі кожного нового фрагмента, забезпечуючи постійну видимість останнього надісланого тексту без ручного прокручування. Повідомлення користувача відображаються з правого боку стрічки з відповідним кольоровим акцентом, повідомлення асистента - з лівого боку з аватарною іконкою, що відповідає загальноприйнятій конвенції дизайну месенджерів і знижує когнітивне навантаження при читанні довгих діалогів. Прокладений між повідомленнями час не відображається у вигляді окремих розділювачів, що підтримує безперервність відчуття живої розмови.

4.2.2 Аватар та відображення емоцій

SVG-аватар розміщено у лівій колонці і є ключовим візуальним елементом емоційного зворотного зв'язку інтерфейсу. Аватар являє собою вбудований SVG-документ, усі елементи якого мають унікальні ідентифікатори і доступні для маніпуляцій через JavaScript DOM API безпосередньо під час роботи застосунку. Конструкція аватара включає контур обличчя у вигляді еліпса зі змінним кольором заповнення, дві брови у вигляді кривих Безьє зі змінними координатами опорних точок, пару очей з рухомими зіницями, рот у вигляді SVG-шляху зі змінною формою, еліпси рум'янців зі змінною непрозорістю, а також окремі шари для аксесуарів і ситуативних ефектів. Система підтримує двадцять виразів обличчя, кожен із яких описується повним набором параметрів стану всіх перелічених елементів.

У паузах між повідомленнями аватар перебуває у стані idle-анімації: очі повільно моргають з випадковим інтервалом у діапазоні, характерному для природного людського моргання, зіниці виконують мікрорухи в межах очей, імітуючи живий погляд, а контур свічення навколо голови персонажа пульсує у повільному ритмі. Ці анімації генеруються за допомогою JavaScript-таймерів з елементами випадковості, що запобігає механічній повторюваності і підтримує

ілюзію живої присутності навіть у відсутності активного діалогу. При отриманні нового повідомлення або початку генерації відповіді idle-таймери скидаються і не відновлюються до настання нової паузи в розмові. Перелік усіх доступних виразів та кнопка їх ручного вибору зображені на рисунку 4.6.



Рисунок 4.6 – Кнопка панелі вибору виразів обличчя аватара

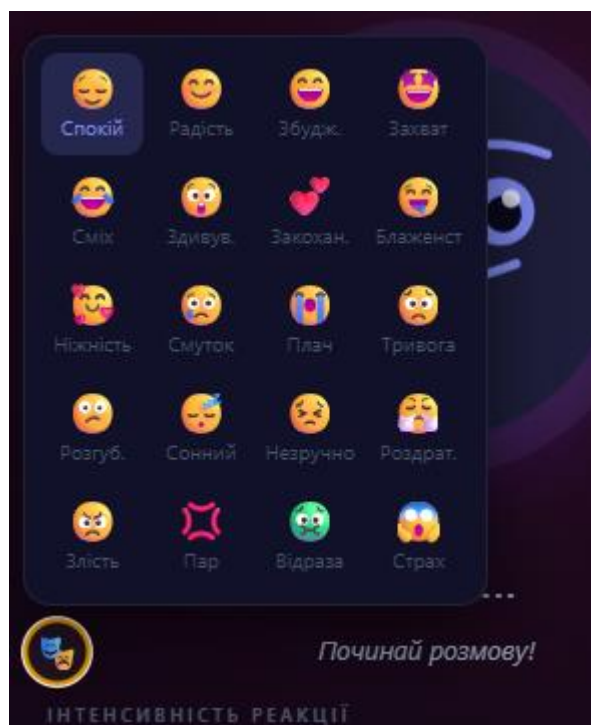


Рисунок 4.7 – Панель вибору виразів обличчя аватара

Після завершення генерації текстової відповіді сервер послідовно звертається до Emotion API двічі: спочатку для аналізу повідомлення користувача, потім для аналізу відповіді асистента. Перший запит передає текст повідомлення користувача з параметром `speaker` зі значенням `User` та ідентифікатор поточної

сесії. Emotion API обробляє текст у контексті всього попереднього діалогу в межах сесії і повертає розпізнану емоцію разом із оновленим sessionId. Другий запит використовує цей оновлений sessionId і передає текст відповіді асистента з параметром speaker зі значенням Assistant. Завдяки цьому Emotion API оцінює відповідь асистента з урахуванням того, на яке саме повідомлення вона є реакцією, що суттєво підвищує точність контекстного аналізу порівняно з незалежною класифікацією окремих фрагментів. Обидва запити прив'язані до спільного ідентифікатора сесії, що дозволяє сервісу враховувати емоційний контекст усього попереднього діалогу.

Результати аналізу надходять до клієнта як окрема SSE-подія типу emotion, після чого модуль reactions.js визначає відповідний вираз обличчя на основі розпізнаної категорії емоції та значення streak-лічильника. Streak-лічильник відстежує кількість послідовних повідомлень з однаковою або схожою категорією емоції: при першому повідомленні з новою категорією аватар відображає перший рівень інтенсивності цієї категорії, при кожному наступному повторенні інтенсивність ескалюється по визначеному ланцюжку до максимального рівня. При зміні категорії лічильник скидається і нова емоція відображається з першого рівня свого ланцюжка. Такий механізм дозволяє аватару відображати поступове наростання емоційного стану, а не реагувати різкими стрибками на кожне окреме повідомлення, що робить його поведінку значно природнішою. Перехід між виразами відбувається плавно завдяки CSS-переходам, застосованим до ключових атрибутів SVG-елементів. Приклади різних станів аватара наведено на рисунках 4.8 - 4.12.

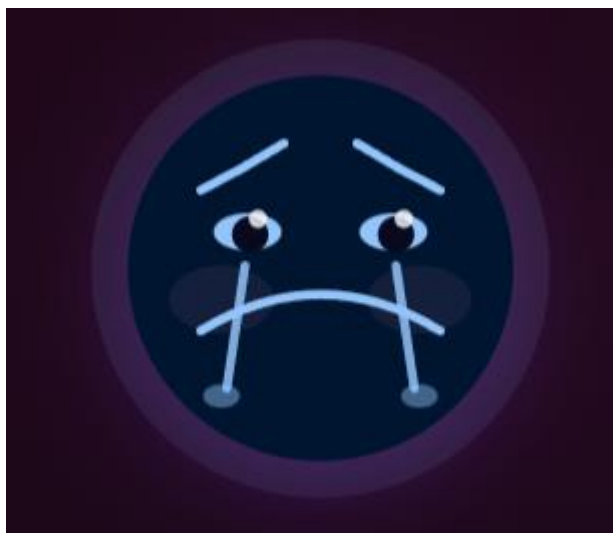


Рисунок 4.8 – Приклади виразів обличчя аватара (Плач)

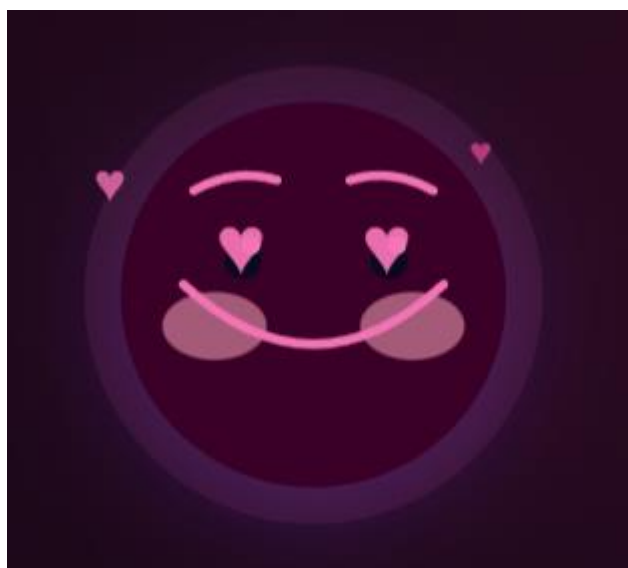


Рисунок 4.9 – Приклади виразів обличчя аватара (Кохання)

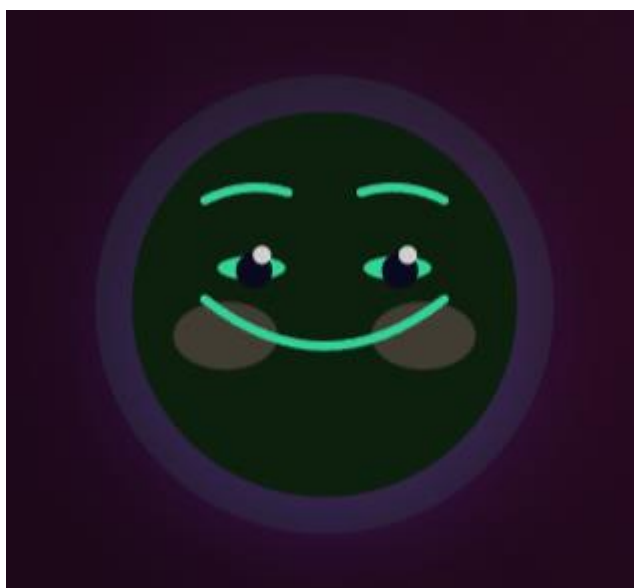


Рисунок 4.10 – Приклади виразів обличчя аватара (Радість)

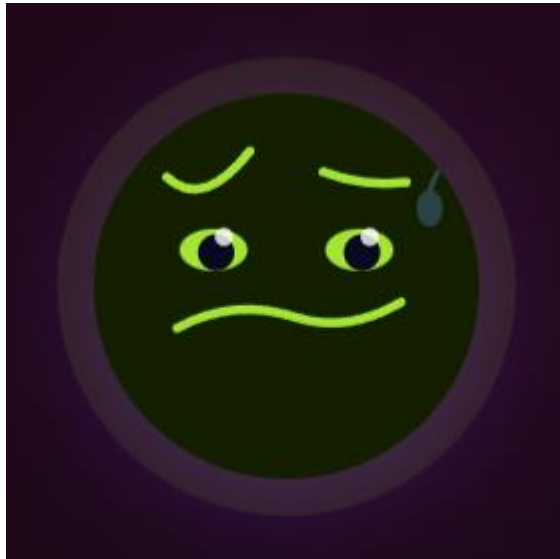


Рисунок 4.11 – Приклади виразів обличчя аватара (Відраза)

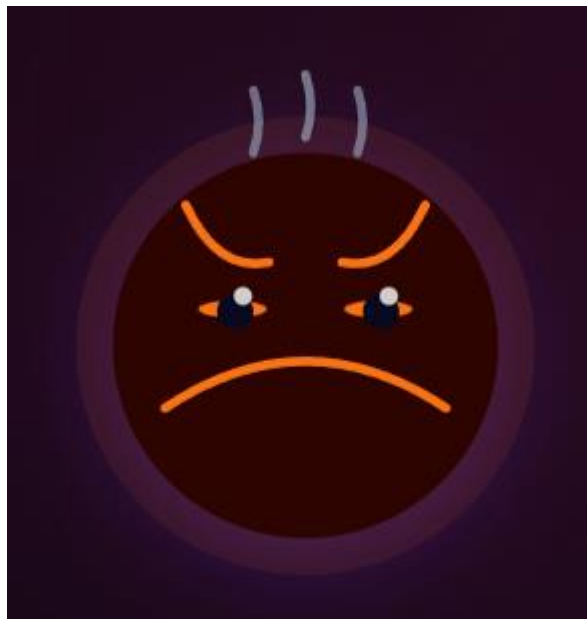


Рисунок 4.12 – Приклади виразів обличчя аватара (Агресія)

Права панель відображає структурований емоційний контекст поточної сесії і складається з кількох інформаційних блоків. Верхній блок показує домінуючу емоцію сесії у вигляді великої іконки з текстовим підписом і оновлюється після кожного нового обміну повідомленнями. Блок траєкторії відображає початкову і поточну емоції сесії зі стрілкою напрямку між ними, а також показник волатильності діалогу у вигляді прогресбару з числовим значенням. Висока волатильність свідчить про часті й різкі зміни емоційного тону протягом сесії, тоді

як низька означає стабільний і рівномірний емоційний фон розмови. Нижче розміщений sparkline-графік температурних значень, що показує динаміку емоційної інтенсивності протягом усього діалогу у вигляді лінійного графіка з точками для кожного повідомлення. Нижній хронологічний блок містить перелік усіх проаналізованих повідомлень із відповідними іконками емоцій і ідентифікаторами учасників, що дозволяє відтворити повну емоційну картину діалогу в деталях.



Рисунок 4.13 – Верхня частина панелі емоційного контексту діалогу



Рисунок 4.14 – Нижня частина панелі емоційного контексту діалогу

Ці дані дозволяють спостерігати за динамікою емоційного тону діалогу в цілому, а не лише реагувати на окремі повідомлення. Вигляд панелі емоційного контексту наведено на рисунках 4.13 і 4.14.

4.2.3 Історія діалогів та налаштування

Ліва панель містить хронологічний перелік усіх збережених діалогів, доступ до якого відкривається через кнопку у шапці застосунку. Кожен діалог зберігається у localStorage браузера як JSON-об'єкт із унікальним числовим ідентифікатором, автоматично сформованим заголовком на основі першого повідомлення, часовою міткою та повним масивом повідомлень разом із збереженими емоціонал-метриками сесії. Заголовок діалогу формується автоматично при надсиланні першого повідомлення шляхом обрізання його до визначеної кількості символів, що дозволяє однозначно ідентифікувати тему розмови без необхідності вручну вводити назву. У списку кожен діалог представлений у вигляді картки із заголовком та датою збереження, що дозволяє швидко орієнтуватися в історії попередніх розмов. Вигляд панелі історії діалогів зображено на рисунку 4.15.

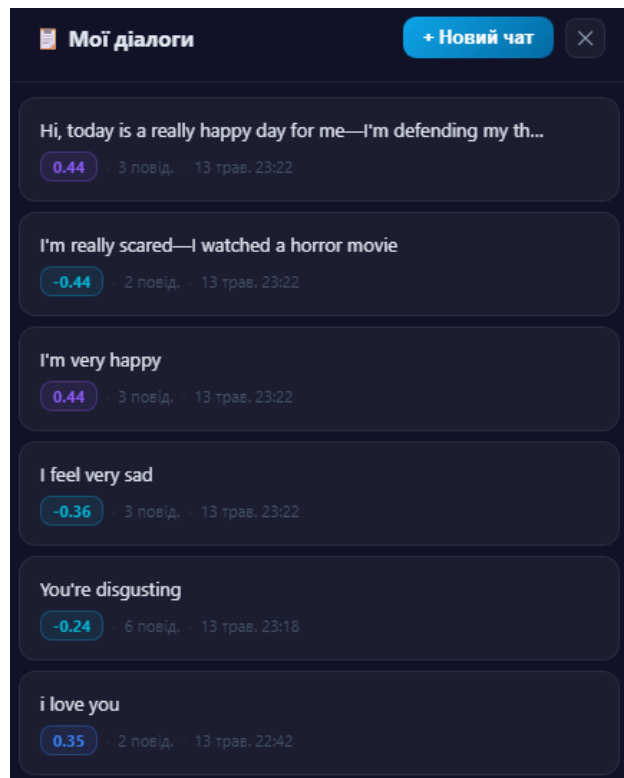


Рисунок 4.15 – Панель історії діалогів

Натискання на картку відновлює повну переписку у центральній колонці, прокручує стрічку повідомлень донизу і автоматично підвантажує збережений ідентифікатор сесії Emotion API. Відновлений sessionId передається у наступному запиті до Emotion API, що дозволяє сервісу продовжити аналіз нових повідомлень в контексті всієї попередньої переписки, а не розпочинати сесію з нуля. Таким чином завантаження збереженого діалогу є повноцінним відновленням стану системи, а не лише відображенням архіву тексту. Кожна картка у списку містить кнопку видалення, яка прибирає діалог зі списку та з localStorage негайно без переходу до окремого екрана підтвердження, що мінімізує кількість дій для виконання рутинної операції. Кнопка створення нового діалогу у шапці панелі очищує центральну колонку, скидає поточний sessionId та всі накопичені емоційні метрики, повертаючи застосунок до початкового вітального екрана.

Модальне вікно налаштувань аватара відкривається через кнопку у шапці застосунку і надає доступ до широкого набору параметрів персоналізації

зовнішнього вигляду персонажа та загального кольорового оформлення інтерфейсу. Блок вибору кольору персонажа містить набір кольорових зразків, кожен із яких відповідає певній акцентній палітрі, що застосовується до кольору очей аватара, свічення навколо голови та загальних акцентних кольорів інтерфейсу. Блок налаштування інтенсивності свічення дозволяє через повзунок плавно регулювати яскравість ефекту свічення навколо голови персонажа від повного відключення до максимальної інтенсивності, що дозволяє адаптувати візуальне навантаження під особисті уподобання або умови освітлення робочого місця. Усі зміни у налаштуваннях застосовуються миттєво в режимі попереднього перегляду без закриття модального вікна, що дозволяє зручно підбирати параметри в реальному часі, спостерігаючи результат безпосередньо на аватарі. Вигляд блоків налаштувань наведено на рисунках 4.16, 4.17 і 4.18.

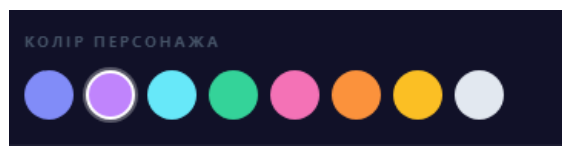


Рисунок 4.16 – Блок налаштувань кольору персонажа

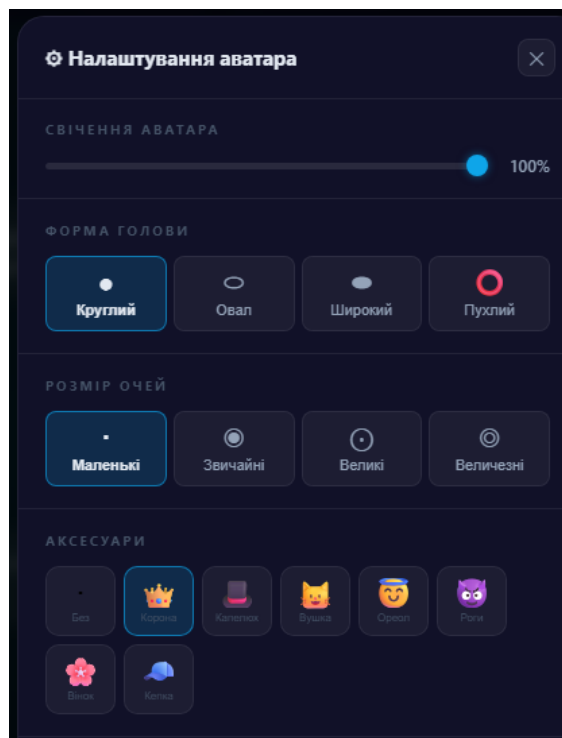


Рисунок 4.17 – Блок налаштувань аватара

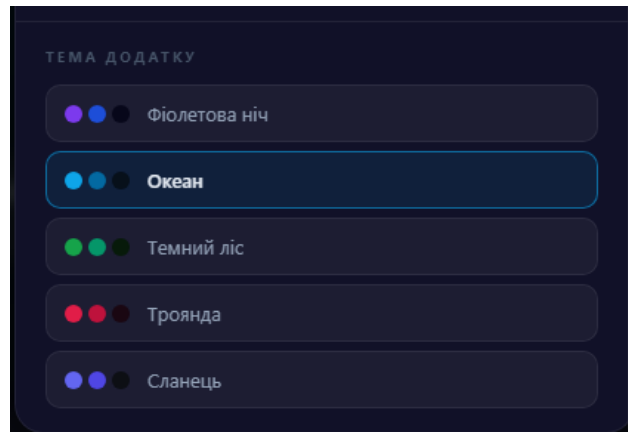


Рисунок 4.18 – Блок налаштувань теми графічного інтерфейсу

Блок вибору теми інтерфейсу надає набір попередньо визначених кольорових тем, кожна з яких задає повну палітру кольорів для всіх елементів застосунку: фонів панелей, акцентів, кольорів повідомлень, підсвічення активних елементів та декоративних ефектів. Перемикання між темами відбувається миттєво через заміну значень CSS-змінних кореневого елемента, що забезпечує одночасне оновлення всіх елементів інтерфейсу без перезавантаження сторінки. Усі вибрані налаштування, включаючи активну тему, колір персонажа, інтенсивність свічення та параметри форми аватара, зберігаються у `localStorage` і відновлюються при наступному відкритті застосунку, що забезпечує персистентність персоналізованого вигляду між сесіями.

ВИСНОВКИ

У процесі виконання роботи було проаналізовано сучасний стан розвитку діалогових систем на основі великих мовних моделей та виявлено ключові обмеження існуючих рішень: відсутність емоційного інтелекту у відповідях, одноманітний стиль спілкування незалежно від емоційного стану користувача, а також відсутність наочного зворотного зв'язку щодо емоційного тону діалогу. Аналіз показав, що переважна більшість сучасних чат-систем орієнтована виключно на змістовну точність відповідей і повністю ігнорує емоційний вимір комунікації, який є невід'ємною частиною природного людського спілкування. Ця прогалина є не лише технічною недосконалістю, а й концептуальним обмеженням, що унеможливлює справжню емпатичну взаємодію між людиною і штучним інтелектом.

Для вирішення цих проблем у розробленому застосунку EmotiChat було застосовано локальну мовну модель Llama 3.2 через платформу Ollama, що забезпечує повну роботу системи без передачі даних до хмарних сервісів, а також окремий сервіс Emotion API для класифікації емоцій у повідомленнях обох учасників діалогу. Передача відповіді від сервера до клієнта реалізована через технологію Server-Sent Events, що забезпечує потоковий вивід тексту без очікування завершення генерації. Поєднання цих трьох компонентів в єдиній архітектурі дозволило реалізувати систему, яка одночасно генерує природні текстові відповіді, аналізує їх емоційний зміст і відображає результати аналізу в інтерфейсі - все це без жодної залежності від зовнішньої хмарної інфраструктури. Повна локальність роботи є принциповою характеристикою EmotiChat, яка відрізняє його від комерційних аналогів і забезпечує абсолютну конфіденційність усіх розмов користувача.

Особливістю розробленої системи є реалізація анімованого SVG-аватара з двадцятьма виразами обличчя, які охоплюють повний спектр емоційних станів від

радості, закоханості та захвату до злості, відрази та страху. На відміну від систем із статичними піктограмами або текстовими індикаторами, аватар EmotiChat реагує на емоційний контекст діалогу в режимі реального часу: форма рота, бровей, очей, колір обличчя та ситуативні ефекти у вигляді сліз, сердечок і пари змінюються плавно завдяки CSS-переходам. Система накопичення реакцій відстежує послідовність емоцій і ескалює вираз обличчя при повторенні однієї категорії, що дозволяє аватару відображати поступове наростання емоційного стану, а не реагувати різкими стрибками на кожне окреме повідомлення. Такий підхід наближає поведінку аватара до природної людської міміки, де емоції накопичуються і наростають поступово, а не змінюються миттєво при кожному новому подразнику. Idle-анімації забезпечують відчуття живої присутності персонажа навіть під час пауз між повідомленнями, підтримуючи ілюзію безперервної взаємодії з живим співрозмовником.

У ході виконання роботи було реалізовано повний функціональний склад застосунку: серверну частину на Node.js із фреймворком Express.js, клієнтський інтерфейс на Vanilla JavaScript без додаткових фронтенд-бібліотек, систему збереження та відновлення діалогів через localStorage, панель емоційного контексту з відображенням агрегованих метрик сесії, блок налаштувань для вибору активної моделі Ollama, а також кнопку ручного вибору будь-якого з двадцяти виразів обличчя для демонстраційних цілей. Уся клієнтська частина розподілена між спеціалізованими модулями з чіткими зонами відповідальності, що забезпечує зручність підтримки та розширення коду. Панель емоційного контексту відображає домінуючу емоцію поточного сеансу, траєкторію зміни емоційного тону від першого до останнього повідомлення, показник волатильності діалогу та sparkline-графік температурних значень, що дозволяє користувачу отримати повну аналітичну картину емоційної динаміки розмови одним поглядом. Збереження діалогів реалізоване з повним відновленням емоційного контексту, що означає: повернувшись до будь-якого попереднього діалогу, користувач бачить не лише

текст повідомлень, а й емоційний стан кожного обміну у тому вигляді, в якому він був зафіксований на момент розмови.

Розроблений застосунок має практичну цінність як демонстраційна платформа для дослідження взаємодії людини з емоційно адаптивним штучним інтелектом. Система може бути розширена для використання у навчальних середовищах, застосунках психологічної підтримки, інтерактивних музейних або виставкових інсталяціях, а також як основа для подальших досліджень у галузі афективних обчислень та людино-машинної взаємодії. Повна автономність роботи без залежності від хмарної інфраструктури робить застосунок придатним для розгортання у середовищах із обмеженим або відсутнім доступом до інтернету. Відкрита архітектура системи, побудована на принципі мінімальних зовнішніх залежностей, дозволяє замінювати окремі компоненти без перебудови всієї системи: мовна модель може бути замінена на будь-яку іншу, підтримувану Ollama, Emotion API може бути замінений альтернативним сервісом із сумісним інтерфейсом, а клієнтська частина може бути адаптована до нових вимог без змін у серверному коді. Таким чином, EmotiChat є не лише завершеним функціональним продуктом, а й масштабованою платформою для подальшого розвитку у напрямку емоційно інтелектуальних систем взаємодії людини з машиною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CSS transitions. Using CSS transitions. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_transitions/Using_CSS_transitions
2. Express.js API Reference. Express - Node.js web application framework. URL: <https://expressjs.com/en/4x/api.html>
3. Fetch API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
4. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. Meta AI Blog. URL: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>
5. Node.js Documentation. Node.js. URL: <https://nodejs.org/en/docs/>
6. ReadableStream. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/API/ReadableStream>
7. SVG: Scalable Vector Graphics. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/SVG>
8. Using server-sent events. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events
9. Web Storage API. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
10. Server Sent Events. Сучасний підручник з JavaScript. URL: <https://uk.javascript.info/server-sent-events>
11. Ekman P. An argument for basic emotions. Cognition & Emotion. 1992. Vol. 6, No. 3–4. P. 169–200. URL: <https://doi.org/10.1080/02699939208411068>
12. Picard R. W. Affective Computing. Cambridge, MA: MIT Press, 1997. 292 p. URL: <https://doi.org/10.7551/mitpress/1140.001.0001>

13. Demszky D., Movshovitz-Attias D., Ko J., Cowen A., Nemade G., Ravi S. GoEmotions: A Dataset of Fine-Grained Emotions. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL). 2020. P. 4040–4054. URL: <https://doi.org/10.18653/v1/2020.acl-main.372>
14. Mohammad S. M., Turney P. D. Crowdsourcing a Word–Emotion Association Lexicon. Computational Intelligence. 2013. Vol. 29, No. 3. P. 436–465. URL: <https://doi.org/10.1111/j.1467-8640.2012.00460.x>
15. Touvron H., Martin L., Stone K. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint. 2023. arXiv:2307.09288. URL: <https://arxiv.org/abs/2307.09288>
16. Poria S., Cambria E., Bajpai R., Hussain A. A review of affective computing: From unimodal analysis to multimodal fusion. Information Fusion. 2017. Vol. 37. P. 98–125. URL: <https://doi.org/10.1016/j.inffus.2017.02.003>
17. Majumder N., Poria S., Hazarika D., Mihalcea R., Gelbukh A., Cambria E. DialogueRNN: An Attentive RNN for Emotion Detection in Conversations. Proceedings of the AAAI Conference on Artificial Intelligence. 2019. Vol. 33. P. 6818–6825. URL: <https://doi.org/10.1609/aaai.v33i01.33016818>
18. Brown T. B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS). 2020. Vol. 33. P. 1877–1901. URL: <https://arxiv.org/abs/2005.14165>
19. dotenv. npm. URL: <https://www.npmjs.com/package/dotenv>
20. Ollama - Get up and running with large language models. Ollama. URL: <https://ollama.com/>

ДОДАТОК А Лістинг розробленої системи

Лістинг А.1 – avatar.js

```
const EXPRESSIONS = {
  calm: { face:'#1e1e3a', eye:'#818cf8', ey:1, browL:'M54,76 Q70,70
86,73', browR:'M114,73 Q130,70 146,76', mouth:'M66,118 Q100,132
134,118', blush:0, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  happy: { face:'#0d200e', eye:'#34d399', ey:.35, browL:'M54,72 Q70,64
86,69', browR:'M114,69 Q130,64 146,72', mouth:'M54,110 Q100,146
146,110', blush:.22, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  excited: { face:'#241a00', eye:'#fbbf24', ey:1.55, browL:'M54,64 Q70,56
86,61', browR:'M114,61 Q130,56 146,64', mouth:'M48,104 Q100,154
152,104', blush:.14, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  enthusiastic: { face:'#2a1200', eye:'#fb923c', ey:1.72, browL:'M54,60 Q70,52
86,57', browR:'M114,57 Q130,52 146,60', mouth:'M44,98 Q100,160
156,98', blush:.18, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  confused: { face:'#1a1530', eye:'#a78bfa', ey:1.15, browL:'M54,74 Q70,72
86,74', browR:'M114,66 Q130,73 146,63', mouth:'M68,121 Q90,112
134,119', blush:0, sweat:.45, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  uncomfortable: { face:'#2a1600', eye:'#fb923c', ey:.72, browL:'M54,68 Q70,79
86,64', browR:'M114,64 Q130,79 146,68', mouth:'M59,128 Q100,114
141,128', blush:0, sweat:.75, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  irritated: { face:'#2e1400', eye:'#f97316', ey:.55, browL:'M54,65 Q70,82
86,72', browR:'M114,72 Q130,82 146,65', mouth:'M56,132 Q100,108
144,132', blush:0, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  angry: { face:'#3a0000', eye:'#ef4444', ey:.32, browL:'M54,62 Q68,86
86,84', browR:'M114,84 Q132,86 146,62', mouth:'M48,136
Q100,96 152,136', blush:0, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  sad: { face:'#001828', eye:'#60a5fa', ey:.7, browL:'M54,80 Q70,76
86,64', browR:'M114,64 Q130,76 146,80', mouth:'M56,132 Q100,104
144,132', blush:0, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  worried: { face:'#0f0830', eye:'#c084fc', ey:1.3, browL:'M54,73 Q70,63
86,70', browR:'M114,67 Q130,63 146,75', mouth:'M61,130 Q100,114
139,130', blush:0, sweat:.9, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  loving: { face:'#2a001a', eye:'#f472b6', ey:.55, browL:'M54,73 Q70,64
86,69', browR:'M114,69 Q130,64 146,73', mouth:'M56,113 Q100,144
144,113', blush:.3, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  blissful: { face:'#3a0030', eye:'#e879f9', ey:.14, browL:'M54,70 Q70,62
86,67', browR:'M114,67 Q130,62 146,70', mouth:'M48,106 Q100,158
152,106', blush:.42, sweat:0, sleepy:0,
fillMouth:null, tongue:0, extra:'' },

  laughing: { face:'#0a1a08', eye:'#4ade80', ey:.08, browL:'M54,68 Q70,58
86,63', browR:'M114,63 Q130,58 146,68', mouth:'M44,98 Q100,162
156,98', blush:.4, sweat:0, sleepy:0, fillMouth:'M44,98 Q100,162 156,98
Q100,118 44,98', tongue:.88, extra:'' },
  crying: { face:'#001530', eye:'#93c5fd', ey:.55, browL:'M54,82 Q70,74
86,64', browR:'M114,64 Q130,74 146,82', mouth:'M54,136 Q100,108
146,136', blush:.08, sweat:0, sleepy:0,
fillMouth:null, tongue:0,
  extra:'<line x1="71" y1="111" x2="64" y2="158" stroke="#93c5fd" stroke-width="3.5"
stroke-linecap="round" opacity=".85"/><ellipse cx="62" cy="161" rx="7" ry="4.5"

```

```

fill="#93c5fd" opacity=".45"/><line x1="129" y1="111" x2="136" y2="158" stroke="#93c5fd"
stroke-width="3.5" stroke-linecap="round" opacity=".85"/><ellipse cx="138" cy="161" rx="7"
ry="4.5" fill="#93c5fd" opacity=".45"/>' },
  surprised: { face:'#0a0a2a', eye:'#67e8f9', ey:1.95, browL:'M54,54 Q70,44
86,52', browR:'M114,52 Q130,44 146,54', mouth:'M82,118 Q100,150
118,118', blush:.1, sweat:.35, sleepy:0, fillMouth:'M82,118 Q100,150 118,118
Q100,128 82,118', tongue:0, extra:'' },
  sleepy: { face:'#100a28', eye:'#a78bfa', ey:.18, browL:'M54,80 Q70,78
86,76', browR:'M114,76 Q130,78 146,80', mouth:'M68,126 Q100,132
132,126', blush:.08, sweat:0, sleepy:1,
fillMouth:null, tongue:0, extra:'' },
  in love: { face:'#3a0028', eye:'#f472b6', ey:0, browL:'M54,72 Q70,63
86,68', browR:'M114,68 Q130,63 146,72', mouth:'M50,108 Q100,154
150,108', blush:.55, sweat:0, sleepy:0,
fillMouth:null, tongue:0,
  extra:'<text x="72" y="98" fill="#f472b6" font-size="30" font-family="serif" text-
anchor="middle" dominant-baseline="middle" opacity=".95">♥</text><text x="128" y="98"
fill="#f472b6" font-size="30" font-family="serif" text-anchor="middle" dominant-
baseline="middle" opacity=".95">♥</text><text x="16" y="76" fill="#f472b6" font-size="19"
font-family="serif" opacity=".8">♥</text><text x="160" y="62" fill="#ec4899" font-size="14"
font-family="serif" opacity=".7">♥</text>' },
  disgusted: { face:'#141f00', eye:'#a3e635', ey:.62, browL:'M54,70 Q68,83
86,60', browR:'M114,68 Q130,74 146,72', mouth:'M58,128 Q80,116 104,124
Q124,130 144,118', blush:0, sweat:.3, sleepy:0,
fillMouth:null, tongue:0, extra:'' },
  scared: { face:'#04041e', eye:'#c084fc', ey:1.95, browL:'M54,60 Q70,50
86,58', browR:'M114,58 Q130,50 146,60', mouth:'M78,116 Q100,148
122,116', blush:0, sweat:.85, sleepy:0, fillMouth:'M78,116 Q100,148 122,116
Q100,128 78,116', tongue:0, extra:'' },
  steaming: { face:'#2e0400', eye:'#f97316', ey:.25, browL:'M54,58 Q66,84
86,80', browR:'M114,80 Q134,84 146,58', mouth:'M46,136 Q100,100
154,136', blush:0, sweat:0, sleepy:0,
fillMouth:null, tongue:0,
  extra:'<path d="M80,38 Q84,26 80,14" stroke="#94a3b8" stroke-width="3.5" fill="none"
stroke-linecap="round" opacity=".65"/><path d="M100,32 Q104,20 100,8" stroke="#94a3b8"
stroke-width="3.5" fill="none" stroke-linecap="round" opacity=".7"/><path d="M120,38
Q124,26 120,14" stroke="#94a3b8" stroke-width="3.5" fill="none" stroke-linecap="round"
opacity=".65"/>' },
};

let currentExpr = 'calm';
let charColor = localStorage.getItem('charColor') || '#818cf8';
let glowIntensity = parseFloat(localStorage.getItem('glowIntensity') || '0.4');
let headShapeId = localStorage.getItem('headShape') || 'round';
let eyeSizeKey = localStorage.getItem('eyeSize') || 'normal';
let currentHat = localStorage.getItem('currentHat') || 'none';
let eyeBaseSize = 13;

const HEAD_SHAPES = {
  round: { rx:74, ry:74, gRx:88, gRy:88 },
  oval: { rx:64, ry:86, gRx:78, gRy:102 },
  wide: { rx:86, ry:62, gRx:102, gRy:76 },
  chubby: { rx:80, ry:80, gRx:95, gRy:95 },
};

const EYE_SIZES = { small:8, normal:13, large:18, huge:24 };

const HATS = {
  none: { back:'', front:'' },
  crown: { back:'', front:`
    <path d="M57,44 L68,16 L84,36 L100,10 L116,36 L132,16 L143,44 Z" fill="#fbbf24"
stroke="#f59e0b" stroke-width="1.5"/>
    <rect x="55" y="42" width="90" height="11" rx="5" fill="#f59e0b"/>
    <circle cx="100" cy="15" r="6" fill="#ef4444"/>
    <circle cx="72" cy="26" r="4.5" fill="#34d399"/>
    <circle cx="128" cy="26" r="4.5" fill="#34d399"/>` },
  witch: { back:'', front:`
    <path d="M100,0 L72,46 L128,46 Z" fill="#1e1b4b" stroke="#818cf8" stroke-width="1.5"/>

```

```

    <ellipse cx="100" cy="48" rx="46" ry="12" fill="#312e81" stroke="#818cf8" stroke-
width="1.5"/>
    <ellipse cx="100" cy="48" rx="30" ry="7" fill="#1e1b4b"/>
    <circle cx="83" cy="42" r="4" fill="#fbbf24" opacity=".7"/>` },
    cat: { back:'', front:`
    <path d="M61,54 L48,18 L82,46 Z" fill="#f472b6" stroke="#db2777" stroke-width="1.5"
stroke-linejoin="round"/>
    <path d="M139,54 L152,18 L118,46 Z" fill="#f472b6" stroke="#db2777" stroke-width="1.5"
stroke-linejoin="round"/>
    <path d="M63,51 L53,24 L79,45 Z" fill="#fda4b8"/>
    <path d="M137,51 L147,24 L121,45 Z" fill="#fda4b8"/>` },
    halo: { back:'', front:`
    <ellipse cx="100" cy="18" rx="38" ry="10" fill="none" stroke="#fbbf24" stroke-width="4"
opacity=".92"/>
    <line x1="100" y1="24" x2="100" y2="42" stroke="#fbbf24" stroke-width="2.5" stroke-
linecap="round" opacity=".4"/>` },
    horns: { back:'', front:`
    <path d="M63,54 L56,14 L88,46 Z" fill="#ef4444" stroke="#b91c1c" stroke-width="1.5"
stroke-linejoin="round"/>
    <path d="M131,54 L144,14 L112,46 Z" fill="#ef4444" stroke="#b91c1c" stroke-width="1.5"
stroke-linejoin="round"/>` },
    flower: { back:'', front:`
    <circle cx="67" cy="48" r="11" fill="#f472b6"/><circle cx="67" cy="48" r="5.5"
fill="#fff8"/>
    <circle cx="82" cy="34" r="11" fill="#fbbf24"/><circle cx="82" cy="34" r="5.5"
fill="#fff8"/>
    <circle cx="100" cy="29" r="11" fill="#f87171"/><circle cx="100" cy="29" r="5.5"
fill="#fff8"/>
    <circle cx="118" cy="34" r="11" fill="#34d399"/><circle cx="118" cy="34" r="5.5"
fill="#fff8"/>
    <circle cx="133" cy="48" r="11" fill="#a78bfa"/><circle cx="133" cy="48" r="5.5"
fill="#fff8"/>` },
    cap: { back:'', front:`
    <rect x="44" y="26" width="112" height="24" rx="9" fill="#1e40af"/>
    <path d="M44,46 Q26,56 22,62 Q58,56 100,48" fill="#1e3a8a"/>
    <line x1="72" y1="28" x2="72" y2="48" stroke="#3b82f6" stroke-width="2" stroke-
linecap="round"/>
    <circle cx="100" cy="27" r="4.5" fill="#60a5fa"/>` },
};

```

```

function applyExpression(state) {
    currentExpr = state;
    const base = EXPRESSIONS[state] || EXPRESSIONS.calm;
    const e = state === 'calm' ? { ...base, eye: charColor } : base;
    const sz = EYE_SIZES[eyeSizeKey] || 13;

    ['eyeL', 'eyeR'].forEach(id => {
        const el = document.getElementById(id);
        el.setAttribute('rx', sz); el.setAttribute('ry', sz);
    });
    const pSz = Math.round(sz * 0.55);
    document.getElementById('pupilL').setAttribute('r', pSz);
    document.getElementById('pupilR').setAttribute('r', pSz);

    document.getElementById('charFace').style.fill = e.face;
    document.getElementById('headGlow').style.fill = e.eye;
    document.getElementById('browL').style.d = `path('${e.browL}')`;
    document.getElementById('browR').style.d = `path('${e.browR}')`;
    document.getElementById('browL').style.stroke = e.eye;
    document.getElementById('browR').style.stroke = e.eye;
    document.getElementById('eyeL').style.fill = e.eye;
    document.getElementById('eyeR').style.fill = e.eye;
    document.getElementById('eyeL').style.transform = `scaleY(${e.ey})`;
    document.getElementById('eyeR').style.transform = `scaleY(${e.ey})`;
    document.getElementById('charMouth').style.d = `path('${e.mouth}')`;
    document.getElementById('charMouth').style.stroke = e.eye;
    document.getElementById('blushL').style.opacity = e.blush;
    document.getElementById('blushR').style.opacity = e.blush;

```

```

document.getElementById('sweatDrop').style.opacity = e.sweat;
document.getElementById('sleepyZ').style.opacity = e.sleepy || 0;

// Filled mouth + tongue
const mf = document.getElementById('mouthFill');
if (e.fillMouth) {
  mf.setAttribute('d', e.fillMouth);
  mf.style.opacity = '1';
} else {
  mf.style.opacity = '0';
}
document.getElementById('tongue').style.opacity = e.tongue || 0;

// Extra SVG layer (tears, hearts, steam...)
const extra = document.getElementById('extraLayer');
if (extra) extra.innerHTML = e.extra || '';

if (state === 'angry' || state === 'steaming') {
  const svg = document.getElementById('charSvg');
  svg.classList.remove('char-shake');
  void svg.offsetWidth;
  svg.classList.add('char-shake');
  setTimeout(() => svg.classList.remove('char-shake'), 600);
}
}

function hexToRgb(hex) {
  const h = hex.replace('#', '');
  return [parseInt(h.slice(0,2),16), parseInt(h.slice(2,4),16), parseInt(h.slice(4,6),16)];
}

function glowFilter(hexColor) {
  const [r,g,b] = hexToRgb(hexColor.slice(0,7));
  return `drop-shadow(0 8px 28px rgba(${r},${g},${b},${(glowIntensity*0.85).toFixed(2)}))`;
}

function applyGlowIntensity(v, save=true) {
  glowIntensity = Math.max(0, Math.min(1, v));
  if (save) localStorage.setItem('glowIntensity', glowIntensity);
  document.getElementById('headGlow').style.opacity = (glowIntensity * 0.22).toFixed(2);
  document.getElementById('charSvg').style.filter =
    `drop-shadow(0 8px 28px rgba(124,58,237,${(glowIntensity*0.85).toFixed(2)}))`;
}

function applyHeadShape(id, save=true) {
  headShapeId = id;
  if (save) localStorage.setItem('headShape', id);
  const s = HEAD_SHAPES[id] || HEAD_SHAPES.round;
  const face = document.getElementById('charFace');
  const glow = document.getElementById('headGlow');
  face.setAttribute('rx', s.rx); face.setAttribute('ry', s.ry);
  glow.setAttribute('rx', s.gRx); glow.setAttribute('ry', s.gRy);
  document.querySelectorAll('.seg-btn[data-shape]').forEach(el =>
    el.classList.toggle('active', el.dataset.shape === id));
}

function applyEyeSize(key, save=true) {
  eyeSizeKey = key;
  if (save) localStorage.setItem('eyeSize', key);
  applyExpression(currentExpr);
  document.querySelectorAll('.seg-btn[data-eye]').forEach(el =>
    el.classList.toggle('active', el.dataset.eye === key));
}

function applyHat(id, save=true) {
  currentHat = id;
  if (save) localStorage.setItem('currentHat', id);
  const h = HATS[id] || HATS.none;
  document.getElementById('hatLayerBack').innerHTML = h.back;
  document.getElementById('hatLayerFront').innerHTML = h.front;
  document.querySelectorAll('.hat-btn').forEach(el =>
    el.classList.toggle('active', el.dataset.hat === id));
}

```

```

//Expression picker
const EXPR_PICKER_ITEMS = [
  { key: 'calm',          ico: '😌', lbl: 'Спокій' },
  { key: 'happy',        ico: '😊', lbl: 'Радість' },
  { key: 'excited',       ico: '😄', lbl: 'Збудж.' },
  { key: 'enthusiastic',  ico: '🤩', lbl: 'Захват' },
  { key: 'laughing',      ico: '😂', lbl: 'Сміх' },
  { key: 'surprised',     ico: '😲', lbl: 'Здивув.' },
  { key: 'in_love',       ico: '❤️', lbl: 'Закохан.' },
  { key: 'blissful',      ico: '😇', lbl: 'Блаженст' },
  { key: 'loving',        ico: '💕', lbl: 'Ніжність' },
  { key: 'sad',           ico: '😞', lbl: 'Смуток' },
  { key: 'crying',        ico: '😭', lbl: 'Плач' },
  { key: 'worried',       ico: '😟', lbl: 'Тривога' },
  { key: 'confused',      ico: '😕', lbl: 'Розгуб.' },
  { key: 'sleepy',        ico: '😴', lbl: 'Сонний' },
  { key: 'uncomfortable', ico: '😖', lbl: 'Незручно' },
  { key: 'irritated',     ico: '😡', lbl: 'Роздрат.' },
  { key: 'angry',         ico: '😠', lbl: 'Злість' },
  { key: 'steaming',      ico: '💨', lbl: 'Пар' },
  { key: 'disgusted',     ico: '😬', lbl: 'Відраза' },
  { key: 'scared',        ico: '😱', lbl: 'Страх' },
];

let _exprPickerOpen = false;

function buildExprPicker() {
  const el = document.getElementById('exprPicker');
  el.innerHTML = EXPR_PICKER_ITEMS.map(x =>
    `<div class="expr-item${currentExpr===x.key?' active':''}"
    onclick="pickExpr('${x.key}')">
      <span class="expr-item-ico">${x.ico}</span>
      <span>${x.lbl}</span>
    </div>`
  ).join('');
}

function pickExpr(key) {
  applyExpression(key);
  buildExprPicker();
}

function toggleExprPicker() {
  const el = document.getElementById('exprPicker');
  _exprPickerOpen = !_exprPickerOpen;
  if (_exprPickerOpen) {
    buildExprPicker();
    el.style.display = 'grid';
  } else {
    el.style.display = 'none';
  }
}

document.addEventListener('click', function(e) {
  if (!_exprPickerOpen) return;
  if (!e.target.closest('#exprPicker') && !e.target.closest('#exprPickBtn')) {
    _exprPickerOpen = false;
    document.getElementById('exprPicker').style.display = 'none';
  }
});

```

Лістинг А.2 – chat.js

```
let sessionId      = null;
let history        = [];
let msgEmotions    = [];
let busy          = false;
let temps          = [];
let histItems      = [];
let lastTemperature = null;
let lastDomEmo     = null;
let lastTraj       = null;

const ARC = 251.33;

const EMO = {
  joy:      { e: '😊', c: '#fbbf24' },
  happiness: { e: '😄', c: '#facc15' },
  excitement: { e: '🎉', c: '#fb923c' },
  sadness:   { e: '😞', c: '#60a5fa' },
  anger:     { e: '😡', c: '#ef4444' },
  fear:      { e: '😱', c: '#a78bfa' },
  surprise:  { e: '😲', c: '#67e8f9' },
  disgust:   { e: '😬', c: '#a3e635' },
  neutral:   { e: '😐', c: '#94a3b8' },
  love:      { e: '❤️', c: '#f472b6' },
  anxiety:   { e: '😰', c: '#c084fc' },
  calm:      { e: '😌', c: '#34d399' },
  frustration: { e: '😡', c: '#fb923c' },
  trust:     { e: '🤝', c: '#4ade80' },
  anticipation: { e: '🌟', c: '#fde68a' },
  confusion:  { e: '😵', c: '#cbd5e1' },
  pride:     { e: '😏', c: '#e879f9' },
  gratitude:  { e: '🙏', c: '#86efac' },
  hope:      { e: '🌟', c: '#fef08a' },
  shame:     { e: '😳', c: '#fda4af' },
};

const emo = (n) => EMO[(n||'').toLowerCase()] || { e: '🗨️', c: '#94a3b8' };

const tempColor = (t) => {
  if (t == null) return '#475569';
  if (t < .2) return '#06b6d4';
  if (t < .4) return '#3b82f6';
}
```

```

    if (t < .6) return '#8b5cf6';
    if (t < .8) return '#f97316';
    return '#ef4444';
};

const TREND = {
  RISING: { icon:'↑', c:'#f97316', lbl:'Зростання' },
  FALLING: { icon:'↓', c:'#3b82f6', lbl:'Спад' },
  STABLE: { icon:'→', c:'#94a3b8', lbl:'Стабільно' },
  VOLATILE: { icon:'↑', c:'#fbbf24', lbl:'Волатильно' },
};

// — Collapsible emo-panel —————
(function initEmoPanel() {
  if (localStorage.getItem('emoPanelCollapsed') === '1') {
    document.getElementById('emoPanel').classList.add('collapsed');
    document.getElementById('emoPanelToggle').textContent = '►';
  }
})();

function toggleEmoPanel() {
  const panel = document.getElementById('emoPanel');
  const btn = document.getElementById('emoPanelToggle');
  const collapsed = panel.classList.toggle('collapsed');
  btn.textContent = collapsed ? '►' : '◄';
  localStorage.setItem('emoPanelCollapsed', collapsed ? '1' : '0');
}

// — Dominant emotion —————
function setDomEmo(name) {
  lastDomEmo = name || null;
  const el = document.getElementById('domEmo');
  if (!name) {
    el.innerHTML = '<div class="dom-placeholder">Почніть розмову<br>для аналізу  
емоцій</div>';
    return;
  }
  const { e, c } = emo(name);
  el.innerHTML = `
    <div class="dom-emoji" style="--ec:${c}40">${e}</div>
    <div class="dom-name" style="--ec:${c};color:${c}">${name.toUpperCase()}</div>`;
}

// — Trajectory —————
function setTraj(traj) {

```

```

if (!traj) return;
lastTraj = traj;
const { startTemperature: s, endTemperature: en, volatility: v, trend } = traj;
document.getElementById('trajCard').style.display = '';
const si = document.getElementById('tStart');
const ei = document.getElementById('tEnd');
si.textContent = (s||0).toFixed(2); si.style.color = tempColor(s||0);
ei.textContent = (en||0).toFixed(2); ei.style.color = tempColor(en||0);
const tc = TREND[trend] || { icon: '→', c: '#94a3b8', lbl: trend || '-' };
const ar = document.getElementById('tArrow');
ar.innerHTML = `<span class="tarrow-icon">${tc.icon}</span><span>${tc.lbl}</span>`;
ar.style.color = tc.c;
const pct = Math.max(0, Math.min(1, v||0));
document.getElementById('volVal').textContent = `${(pct*100).toFixed(0)}%`;
document.getElementById('volFill').style.width = `${pct*100}%`;
}

// — Sparkline —————
function drawSpark() {
  if (temps.length < 2) return;
  document.getElementById('sparkCard').style.display = '';
  const svg = document.getElementById('sparkSvg');
  const W=300, H=56, P=8;
  const mn = Math.min(...temps), mx = Math.max(...temps);
  const rng = mx - mn || 0.1;
  const pts = temps.map((t,i) => {
    const x = P + (i/(temps.length-1))*(W-P*2);
    const y = H - P - ((t-mn)/rng)*(H-P*2);
    return `${x.toFixed(1)},${y.toFixed(1)}`;
  }).join(' ');
  const last = temps[temps.length-1];
  const c = tempColor(last);
  const lx = +(P + (W-P*2)).toFixed(1);
  const ly = +(H - P - ((last-mn)/rng)*(H-P*2)).toFixed(1);
  svg.innerHTML = `
    <polyline points="${pts}" fill="none" stroke="${c}" stroke-width="2" stroke-
linejoin="round" stroke-linecap="round"/>
    <circle cx="${lx}" cy="${ly}" r="4" fill="${c}" opacity=".9"/>`;
}

// — Emotion history panel —————
function addHist(speaker, eData) {
  if (!eData) return;
  const area = document.getElementById('histArea');
  if (histItems.length === 0) {
    area.innerHTML = '<div class="hist-ttl">Історія повідомлень</div>';
  }
}

```

```

    }
    const { emotion=[], temperature } = eData;
    const tc = tempColor(temperature||0);
    const chips = _chipsHtml(emotion);
    const item = document.createElement('div');
    item.className = 'hist-item';
    item.innerHTML = `
      <div class="hist-hdr">
        <span class="hist-spk">${speaker==='User'?'👤 Ви':'👤 III'}</span>
        <span class="hist-t" style="color:${tc}">${(temperature||0).toFixed(2)}</span>
      </div>
      <div class="hist-chips">${chips||'<span style="color:var(--text3);font-size:11px">немає
даних</span>'}</div>`;
    const ttl = area.querySelector('.hist-ttl');
    if (ttl) ttl.insertAdjacentElement('afterend', item);
    else area.prepend(item);
    histItems.push(item);
    if (histItems.length > 8) histItems.shift().remove();
  }

// — Chip helpers —————
function _chipsHtml(emotions) {
  return (emotions||[]).map(n => {
    const { e, c } = emo(n);
    return `<span class="chip" style="background:${c}20;color:${c};border:1px solid
${c}40">${e} ${n}</span>`;
  }).join('');
}

// — Messages —————
function esc(s) { const d=document.createElement('div'); d.textContent=s; return
d.innerHTML; }

function addMsg(role, content, streaming=false) {
  document.getElementById('welcome')?.remove();
  const msgs = document.getElementById('msgs');
  const typing = document.getElementById('typing');
  const time = new Date().toLocaleTimeString('ru-RU',{hour:'2-digit',minute:'2-digit'});
  const uid = Date.now();
  const div = document.createElement('div');
  div.className = `msg ${role}`;
  const avHtml = `<div class="avatar">${role==='user'?'👤':'👤'}</div>`;
  div.innerHTML = `
    ${role==='assistant'?avHtml:''}
    <div class="msg-body">

```

```

        <div class="bubble${streaming?' streaming':''}" id="b${uid}">${esc(content)}</div>
        <div class="bubble-chips" id="c${uid}"></div>
        <div class="msg-meta">${time}</div>
    </div>
    ${role==='user'?avHtml:''}`;
    msgs.insertBefore(div, typing);
    scrollDown();
    return div;
}

function setBubble(el, text) {
    el.querySelector('.bubble').textContent = text;
    scrollDown();
}

function finishBubble(el) {
    el.querySelector('.bubble')?.classList.remove('streaming');
}

function addChips(el, eData) {
    if (!eData?.emotion) return;
    const cont = el.querySelector('[id^="c"]');
    if (!cont) return;
    eData.emotion.forEach(n => {
        const { e, c } = emo(n);
        const s = document.createElement('span');
        s.className = 'chip';
        s.style.cssText = `background:${c}20;color:${c};border:1px solid ${c}40`;
        s.textContent = `${e} ${n}`;
        cont.appendChild(s);
    });
}

function scrollDown() {
    const m = document.getElementById('msgs');
    m.scrollTop = m.scrollHeight;
}

// — Status —————
function setStatus(on, txt) {
    document.getElementById('statusDot').className = 'dot' + (on?' on':'');
    document.getElementById('statusTxt').textContent = txt;
}

// — Toast —————
function toast(msg) {

```

```

const t = document.getElementById('toast');
t.textContent = msg; t.classList.add('show');
setTimeout(() => t.classList.remove('show'), 3500);
}

// — Conversation persistence —————
const CONV_KEY = 'emotiChatHistory';
const CONV_MAX = 30;

function getSavedConversations() {
  try { return JSON.parse(localStorage.getItem(CONV_KEY) || '[]'); }
  catch { return []; }
}

function saveCurrentConversation() {
  if (!history.length) return;
  const firstUser = history.find(m => m.role === 'user')?.content || '';
  const title = firstUser.length > 58 ? firstUser.slice(0, 58) + '...' : firstUser;

  const convs = getSavedConversations();
  const existIdx = sessionId ? convs.findIndex(c => c.sessionId === sessionId) : -1;

  const conv = {
    id: existIdx >= 0 ? convs[existIdx].id : Date.now(),
    title,
    temperature: lastTemperature,
    messageCount: history.filter(m => m.role === 'user').length,
    createdAt: existIdx >= 0 ? convs[existIdx].createdAt : Date.now(),
    updatedAt: Date.now(),
    messages: [...history],
    sessionId,
    msgEmotions: [...msgEmotions],
    reactHistory: [...reactHistory],
    userEmoHist: [...userEmoHist],
    temps: [...temps],
    lastDomEmo,
    lastTraj,
  };

  if (existIdx >= 0) {
    convs[existIdx] = conv;
  } else {
    convs.unshift(conv);
    if (convs.length > CONV_MAX) convs.length = CONV_MAX;
  }
  localStorage.setItem(CONV_KEY, JSON.stringify(convs));
}

```

```

}

function deleteConversation(id, e) {
  if (e) { e.stopPropagation(); e.preventDefault(); }
  const convs = getSavedConversations().filter(c => c.id !== id);
  localStorage.setItem(CONV_KEY, JSON.stringify(convs));
  renderHistoryList();
}

// — Chat reset —————
function _makeWelcome() {
  const el = document.createElement('div');
  el.className = 'welcome'; el.id = 'welcome';
  el.innerHTML = `
    <div class="welcome-icon"><img alt="chat icon" data-bbox="395 342 412 355"/></div>
    <h2>Почніть розмову</h2>
    <p>Спілкуйтесь з локальним ШІ та спостерігайте за емоційним контекстом діалогу в
    реальному часі</p>
    <div class="prompts">
      <div class="prompt-chip" onclick="usePrompt(this)">Розкажи щось цікаве</div>
      <div class="prompt-chip" onclick="usePrompt(this)">Як у тебе справи?</div>
      <div class="prompt-chip" onclick="usePrompt(this)">Допоможи з ідеями для
    проекту</div>
      <div class="prompt-chip" onclick="usePrompt(this)">Що ти думаєш про майбутнє
    ШІ?</div>
    </div>`;
  return el;
}

function _resetChat() {
  history = []; msgEmotions = []; sessionId = null;
  lastTemperature = null; lastDomEmo = null; lastTraj = null;
  temps = []; histItems = []; busy = false;

  // Reset msgs area
  const msgs = document.getElementById('msgs');
  const typing = document.getElementById('typing');
  [...msgs.children].forEach(el => { if (el.id !== 'typing') el.remove(); });
  msgs.insertBefore(_makeWelcome(), typing);
  typing.classList.remove('on');

  // Reset emotion panel
  setDomEmo(null);
  document.getElementById('trajCard').style.display = 'none';
  document.getElementById('sparkCard').style.display = 'none';

```

```

document.getElementById('histArea').innerHTML = `
  <div class="empty">
    <div class="empty-icon">📄</div>
    <span>Історія емоцій з'явиться<br>після першого повідомлення</span>
  </div>`;

// Reset reaction panel
const rp = document.getElementById('reactPanel');
rp.style.removeProperty('--rc');
document.getElementById('reactGlow').classList.remove('on');
document.getElementById('reactLabel').textContent = 'Очікую...';
document.getElementById('reactMsg').textContent = 'Починай розмову!';
document.getElementById('reactIntFill').style.width = '0%';
document.getElementById('reactIntVal').textContent = '0%';
document.getElementById('reactIntVal').style.color = '';
document.getElementById('reactTl').innerHTML = '<span class="rt-empty">Ще немає
даних...</span>';

// Reset shared arrays (declared in reactions.js)
userEmoHist.splice(0);
reactHistory.splice(0);

applyExpression('calm');
applyGlowIntensity(glowIntensity, false);
setStatus(false, 'Очікування');
document.getElementById('sendBtn').disabled = false;
}

function startNewChat() {
  if (history.length) saveCurrentConversation();
  _resetChat();
  closeHistory();
}

// — Load saved conversation —————
function loadConversation(id) {
  const conv = getSavedConversations().find(c => c.id === id);
  if (!conv) return;

  if (history.length) saveCurrentConversation();
  _resetChat();

  document.getElementById('welcome')?.remove();

  // Restore state

```

```

history          = [...conv.messages];
sessionId        = conv.sessionId;
lastTemperature  = conv.temperature;
lastDomEmo       = conv.lastDomEmo || null;
lastTraj         = conv.lastTraj    || null;
temps            = [...(conv.temps || [])];
histItems        = [];
msgEmotions      = [...(conv.msgEmotions || [])];

// Restore reaction arrays
userEmoHist.splice(0, userEmoHist.length, ...(conv.userEmoHist || []));
reactHistory.splice(0, reactHistory.length, ...(conv.reactHistory || []));

// Render messages with emotion chips
const msgs      = document.getElementById('msgs');
const typing    = document.getElementById('typing');

conv.messages.forEach((m, idx) => {
  const exchangeIdx = Math.floor(idx / 2);
  const eData = conv.msgEmotions?.[exchangeIdx];
  const chips = m.role === 'user'
    ? _chipsHtml(eData?.userEmotion?.emotion)
    : _chipsHtml(eData?.aiEmotion?.emotion);

  const div = document.createElement('div');
  div.className = `msg ${m.role}`;
  const avHtml = `<div class="avatar">${m.role==='user'? '👤': '🤖'}</div>`;
  div.innerHTML = `
    ${m.role==='assistant'?avHtml:''}
    <div class="msg-body">
      <div class="bubble">${esc(m.content)}</div>
      <div class="bubble-chips">${chips}</div>
      <div class="msg-meta">-</div>
    </div>
    ${m.role==='user'?avHtml:''}`;
  msgs.insertBefore(div, typing);
});

scrollDown();

// Restore emotion panel
if (conv.lastDomEmo) setDomEmo(conv.lastDomEmo);
if (conv.lastTraj)   setTraj(conv.lastTraj);
if (temps.length >= 2) drawSpark();

```

```

// Restore emotion history panel
if (conv.msgEmotions?.length) {
    document.getElementById('histArea').innerHTML = '<div class="hist-ttl">Історія
повідомлень</div>';
    conv.msgEmotions.forEach(ex => {
        if (ex.userEmotion) _addHistItem('User', ex.userEmotion);
        if (ex.aiEmotion) _addHistItem('Assistant', ex.aiEmotion);
    });
}

// Restore reaction timeline and avatar state
if (reactHistory.length) {
    renderTimeline();
    const last = reactHistory[reactHistory.length - 1];
    if (last) _restoreReactionUI(last);
}

if (conv.sessionId) setStatus(true, `Cecia · ${conv.sessionId.toString().slice(0,8)}...`);

closeHistory();
}

function _addHistItem(speaker, eData) {
    if (!eData) return;
    const area = document.getElementById('histArea');
    const { emotion=[], temperature } = eData;
    const tc = tempColor(temperature||0);
    const chips = _chipsHtml(emotion);
    const item = document.createElement('div');
    item.className = 'hist-item';
    item.innerHTML = `
        <div class="hist-hdr">
            <span class="hist-spk">${speaker==='User'? 'Ви': 'ШІ'}</span>
            <span class="hist-t" style="color:${tc}>${(temperature||0).toFixed(2)}</span>
        </div>
        <div class="hist-chips">${chips||'<span style="color:var(--text3);font-size:11px">немає
даних</span>'}</div>`;
    area.appendChild(item);
    histItems.push(item);
}

function _restoreReactionUI(r) {
    const rp = document.getElementById('reactPanel');
    rp.style.setProperty('--rc', r.color);
    document.getElementById('reactGlow').classList.add('on');
}

```

```

    applyExpression(r.expr || 'calm');
    document.getElementById('charSvg').style.filter      = glowFilter(r.color);
    document.getElementById('reactLabel').textContent   = r.label;
    document.getElementById('reactLabel').style.color    = r.color;
    document.getElementById('reactMsg').textContent     = r.msg;
    const pct = Math.round((r.intensity ?? 1) * 100);
    document.getElementById('reactIntFill').style.width  = `${pct}%`;
    document.getElementById('reactIntFill').style.background = r.color;
    document.getElementById('reactIntFill').style.boxShadow = `0 0 8px ${r.color}`;
    document.getElementById('reactIntVal').textContent = `${pct}%`;
    document.getElementById('reactIntVal').style.color = r.color;
  }

  // — History list UI —————
  function renderHistoryList() {
    const convs = getSavedConversations();
    const container = document.getElementById('historyList');

    if (!convs.length) {
      container.innerHTML = `
        <div class="hist-empty">
          <div class="hist-empty-icon"><img alt="empty icon" data-bbox="458 468 478 483"/></div>
          Збережених діалогів ще немає.<br>Почніть розмову — вона збережеться автоматично.
        </div>`;
      return;
    }

    container.innerHTML = convs.map(c => {
      const tc      = tempColor(c.temperature);
      const tempStr = c.temperature != null ? c.temperature.toFixed(2) : '-';
      const d        = new Date(c.updatedAt || c.createdAt);
      const dateStr  = d.toLocaleDateString('uk-UA', { day:'numeric', month:'short' });
      const timeStr  = d.toLocaleTimeString('uk-UA', { hour:'2-digit', minute:'2-digit' });
      return `
        <div class="hconv-card" onclick="if(!event.target.closest('.hconv-
del'))loadConversation(${c.id})">
          <div class="hconv-title">${esc(c.title)}</div>
          <div class="hconv-meta">
            <span class="hconv-temp" style="color:${tc};border-
color:${tc}50;background:${tc}18">${tempStr}</span>
            <span class="hconv-dot">·</span>
            <span class="hconv-count">${c.messageCount} повід.</span>
            <span class="hconv-dot">·</span>
            <span class="hconv-date">${dateStr} ${timeStr}</span>
          </div>
      `;
    });
  }

```

```

        <button class="hconv-del" onclick="deleteConversation(${c.id}, event)"
title="Видалити">X</button>
    </div>`;
    }).join('');
}

function openHistory() {
    renderHistoryList();
    document.getElementById('historyOverlay').style.display = 'flex';
}
function closeHistory() {
    document.getElementById('historyOverlay').style.display = 'none';
}
function histOverlayClick(e) {
    if (e.target === document.getElementById('historyOverlay')) closeHistory();
}

// — Send —————
async function sendMsg() {
    const inp = document.getElementById('inp');
    const text = inp.value.trim();
    if (!text || busy) return;

    inp.value = ''; inp.style.height = 'auto';
    busy = true;
    document.getElementById('sendBtn').disabled = true;

    history.push({ role:'user', content:text });
    const userEl = addMsg('user', text);
    document.getElementById('typing').classList.add('on');
    setStatus(true, 'III друкую...');

    let aiEl = null, full = '';
    let currentUserEmotion = null, currentAiEmotion = null;

    try {
        const resp = await fetch('/api/chat', {
            method: 'POST',
            headers: { 'Content-Type':'application/json' },
            body: JSON.stringify({
                messages: history,
                sessionId,
                model: document.getElementById('modelSelect').value,
            }),
        });
    }

```

```

if (!resp.ok) throw new Error(`HTTP ${resp.status}`);

const reader = resp.body.getReader();
const dec = new TextDecoder();
let buf = '';

while (true) {
  const { done, value } = await reader.read();
  if (done) break;
  buf += dec.decode(value, { stream:true });
  const lines = buf.split('\n');
  buf = lines.pop();

  for (const line of lines) {
    if (!line.startsWith('data: ')) continue;
    const ds = line.slice(6).trim();
    if (ds === '[DONE]') continue;
    try {
      const d = JSON.parse(ds);

      if (d.type === 'text') {
        if (!aiEl) {
          document.getElementById('typing').classList.remove('on');
          aiEl = addMsg('assistant', '', true);
        }
        full += d.text;
        setBubble(aiEl, full);
      }

      if (d.type === 'emotion') {
        sessionId = d.sessionId;

        if (d.context?.overallTemperature != null) {
          lastTemperature = d.context.overallTemperature;
          temps.push(lastTemperature);
          drawSpark();
        }

        setGauge(d.context?.overallTemperature);
        setDomEmo(d.context?.dominantDialogueEmotion);
        setTraj(d.context?.trajectory);

        currentUserEmotion = d.userEmotion;
        currentAiEmotion = d.aiEmotion;

        addChips(userEl, d.userEmotion);
        if (aiEl) addChips(aiEl, d.aiEmotion);
      }
    } catch (e) {
      console.error(e);
    }
  }
}

```

```

    addHist('User', d.userEmotion);
    addHist('Assistant', d.aiEmotion);

    if (d.userEmotion?.emotion?.length) {
        userEmoHist.push(d.userEmotion.emotion[0]);
        const r = computeReaction();
        if (r) applyReaction(r);
    }
    setStatus(true, `Cecия · ${d.sessionId.toString().slice(0,8)}...`);
}

if (d.type === 'emotion_error') console.warn('Emotion API:', d.message);
if (d.type === 'error') throw new Error(d.message);

} catch (pe) {
    if (pe.message && !pe.message.includes('JSON')) throw pe;
}
}

if (aiEl) {
    finishBubble(aiEl);
    history.push({ role:'assistant', content:full });
    // Save emotion data for this exchange
    msgEmotions.push({ userEmotion: currentUserEmotion, aiEmotion: currentAiEmotion });
    saveCurrentConversation();
}

} catch (err) {
    console.error(err);
    document.getElementById('typing').classList.remove('on');
    setStatus(false, 'Помилка');
    toast('⚠ ' + err.message);
} finally {
    busy = false;
    document.getElementById('sendBtn').disabled = false;
    document.getElementById('typing').classList.remove('on');
}
}

// — Helpers —————
function usePrompt(el) {
    const inp = document.getElementById('inp');
    inp.value = el.textContent; inp.focus(); resize(inp);
}

```

```

function resize(el) {
    el.style.height = 'auto';
    el.style.height = Math.min(el.scrollHeight, 130) + 'px';
}

// — Events —————
const inp = document.getElementById('inp');
inp.addEventListener('input', () => resize(inp));
inp.addEventListener('keydown', (e) => {
    if (e.key === 'Enter' && !e.shiftKey) { e.preventDefault(); sendMsg(); }
});

// — Load models —————
(async () => {
    try {
        const models = await fetch('/api/models').then(r=>r.json());
        const sel = document.getElementById('modelSelect');
        if (models.length === 0) {
            const opt = document.createElement('option');
            opt.value = 'llama3.2'; opt.textContent = 'llama3.2 (за замовчуванням)';
            sel.appendChild(opt);
        } else {
            models.forEach(m => {
                const opt = document.createElement('option');
                opt.value = m; opt.textContent = m;
                sel.appendChild(opt);
            });
        }
    } catch { /* Ollama offline */ }
})();

```

Лістинг A.3 – idle.js

```

const BASE_P = { lx:73, ly:99, rx:129, ry:99 };
let idleBusy = false;

function setEyeScale(v, ms = 80) {
    const eL = document.getElementById('eyeL');
    const eR = document.getElementById('eyeR');
    eL.style.transition = eR.style.transition = `transform ${ms}ms ease`;
    eL.style.transform = eR.style.transform = `scaleY(${v})`;
}

function restoreEyeScale() {
    const ey = (EXPRESSIONS[currentExpr] || EXPRESSIONS.calm).ey;
    document.getElementById('eyeL').style.transition =
    document.getElementById('eyeR').style.transition = 'transform .12s ease, fill .65s ease';
}

```

```

    document.getElementById('eyeL').style.transform =
    document.getElementById('eyeR').style.transform = `scaleY(${ey})`;
}
function setPupils(dx, dy) {
    const pL = document.getElementById('pupilL');
    const pR = document.getElementById('pupilR');
    pL.setAttribute('cx', BASE_P.lx + dx);
    pL.setAttribute('cy', BASE_P.ly + dy);
    pR.setAttribute('cx', BASE_P.rx + dx);
    pR.setAttribute('cy', BASE_P.ry + dy);
}
function doBlink(then) {
    setEyeScale(0.04, 80);
    setTimeout(() => { setEyeScale(1, 80); setTimeout(() => { restoreEyeScale(); if (then)
then(); }, 90); }, 90);
}

const IDLE_ACTIONS = [
    function actBlink(done) { doBlink(done); },
    function actDoubleBlink(done) {
        doBlink(() => setTimeout(() => doBlink(done), 220));
    },
    function actLookLeft(done) {
        setPupils(-7, 0);
        setTimeout(() => { setPupils(0, 0); setTimeout(done, 200); }, 1100);
    },
    function actLookRight(done) {
        setPupils(7, 0);
        setTimeout(() => { setPupils(0, 0); setTimeout(done, 200); }, 1100);
    },
    function actLookUp(done) {
        setPupils(0, -6);
        setTimeout(() => { setPupils(0, 0); setTimeout(done, 200); }, 900);
    },
    function actLookDown(done) {
        setPupils(0, 6);
        setTimeout(() => { setPupils(0, 0); setTimeout(done, 200); }, 900);
    },
    function actYawn(done) {
        setEyeScale(0.42, 380);
        const mouth = document.getElementById('charMouth');
        const fill = document.getElementById('mouthFill');
        const tongue = document.getElementById('tongue');
        mouth.style.d = "path('M62,114 Q100,154 138,114')";
        fill.style.opacity = '1';
        tongue.style.opacity = Math.random() < 0.30 ? '0.85' : '0';
    }
];

```

```

    setTimeout(() => {
      doBlink(() => {
        restoreEyeScale();
        const e = EXPRESSIONS[currentExpr] || EXPRESSIONS.calm;
        mouth.style.d = `path('${e.mouth}')`;
        fill.style.opacity = '0';
        tongue.style.opacity = '0';
        setTimeout(done, 400);
      });
    }, 2000);
  },
  function actSleepy(done) {
    const z = document.getElementById('sleepyZ');
    setEyeScale(0.3, 700);
    z.style.transition = 'opacity .6s ease';
    z.style.opacity = '1';
    setTimeout(() => {
      z.style.opacity = '0';
      doBlink(() => { restoreEyeScale(); setTimeout(done, 300); });
    }, 2200);
  },
  function actGlance(done) {
    setPupils(-7, 0);
    setTimeout(() => {
      setPupils(7, 0);
      setTimeout(() => { setPupils(0, 0); setTimeout(done, 200); }, 450);
    }, 450);
  },
  function actSoftBlink(done) {
    setEyeScale(0.15, 300);
    setTimeout(() => { restoreEyeScale(); setTimeout(done, 400); }, 700);
  },
];

const IDLE_WEIGHTS = [24, 11, 9, 9, 7, 5, 3, 5, 8, 7];

function pickIdleAction() {
  const total = IDLE_WEIGHTS.reduce((a, b) => a + b, 0);
  let r = Math.random() * total;
  for (let i = 0; i < IDLE_WEIGHTS.length; i++) {
    r -= IDLE_WEIGHTS[i];
    if (r <= 0) return IDLE_ACTIONS[i];
  }
  return IDLE_ACTIONS[0];
}

```

```
function scheduleIdle() {
  const delay = 1800 + Math.random() * 3200;
  setTimeout(() => {
    if (idleBusy) { scheduleIdle(); return; }
    idleBusy = true;
    pickIdleAction() (() => { idleBusy = false; scheduleIdle(); });
  }, delay);
}
```

```
scheduleIdle();
```

Лістинг A.4 – reactions.js

```
const userEmoHist = [];
const reactHistory = [];

const REACT_MAP = {
  // — Гнів: confused → uncomfortable → irritated → angry → steaming —
  anger: [
    { emoji: '😬', label: 'Розгублений', color: '#a78bfa', msg: '"Чому так агресивно? Трохи дивно..."', expr: 'confused' },
    { emoji: '😬', label: 'Некомфортно', color: '#fb923c', msg: '"Ця агресія починає тиснути на мене..."', expr: 'uncomfortable' },
    { emoji: '😬', label: 'Роздратований', color: '#f97316', msg: '"Знаєш, мене теж це починає дратувати..."', expr: 'irritated' },
    { emoji: '😬', label: 'Злий', color: '#ef4444', msg: '"Та досить вже! Два можуть грати в цю гру!"', expr: 'angry' },
    { emoji: '😬', label: 'В шаленстві', color: '#dc2626', msg: '"ЦЕ ВЖЕ ЗАНАДТО! Мене аж пара іде з голови!"', expr: 'steaming' },
  ],
  // — Розчарування: confused → sad → irritated → crying —
  frustration: [
    { emoji: '😬', label: 'Розгублений', color: '#a78bfa', msg: '"Щось пішло не так? Не зовсім розумію..."', expr: 'confused' },
    { emoji: '😬', label: 'Засмучений', color: '#60a5fa', msg: '"Шкода, що ти такий засмучений..."', expr: 'sad' },
    { emoji: '😬', label: 'Виснажений', color: '#f97316', msg: '"Це вже й мене починає виснажувати..."', expr: 'irritated' },
    { emoji: '😬', label: 'Зламаний', color: '#3b82f6', msg: '"Все... Більше не можу стримуватись. Це надто важко..."', expr: 'crying' },
  ],
  // — Смуток: sad → worried → sad → crying —
  sadness: [
    { emoji: '😬', label: 'Співчуваю', color: '#93c5fd', msg: '"Мені шкода... Розкажи, що сталося..."', expr: 'sad' },
  ]
}
```

```

    { emoji:'😟', label:'Стурбований', color:'#60a5fa', msg:'"Я справді хвилююся за тебе."',
      expr:'worried' },
    { emoji:'😞', label:'Засмучений', color:'#3b82f6', msg:'"Разом із тобою сумую. Я тут поруч."',
      expr:'sad' },
    { emoji:'😭', label:'Плачу', color:'#1d4ed8', msg:'"Мое серце стискається разом з твоїм..."',
      expr:'crying' },
  ],
  // — Радість: happy → excited → enthusiastic → laughing —————
  joy: [
    { emoji:'😊', label:'Радий', color:'#34d399', msg:'"Приємно бачити тебе в гарному настрої"',
      expr:'happy' },
    { emoji:'😄', label:'В захваті', color:'#fbbf24', msg:'"Твоя радість заряджає і мене"',
      expr:'excited' },
    { emoji:'🥳', label:'Захоплений', color:'#fb923c', msg:'"Разом ми просто сяємо"',
      expr:'enthusiastic' },
    { emoji:'😂', label:'Рогочу', color:'#10b981', msg:'"XAXA! Від такої радості я просто вибухаю сміхом"',
      expr:'laughing' },
  ],
  // — Збудженість: excited → enthusiastic → laughing —————
  excitement: [
    { emoji:'😄', label:'Захоплений', color:'#fbbf24', msg:'"Ого, це реально цікаво"',
      expr:'excited' },
    { emoji:'🥳', label:'В екстазі', color:'#fb923c', msg:'"Твій ентузіазм заразливий — я теж горю"',
      expr:'enthusiastic' },
    { emoji:'😂', label:'Нестримний', color:'#10b981', msg:'"Не можу стримати себе — це просто вибух"',
      expr:'laughing' },
  ],
  // — Страх: worried → scared —————
  fear: [
    { emoji:'😟', label:'Стурбований', color:'#c084fc', msg:'"Ти в порядку? Хвилююся за тебе..."',
      expr:'worried' },
    { emoji:'😱', label:'Злякався', color:'#7c3aed', msg:'"Це справді лякає! Але ми разом — я поруч."',
      expr:'scared' },
  ],
  // — Любов: loving → blissful → in_love —————
  love: [
    { emoji:'🥰', label:'Зворушений', color:'#f472b6', msg:'"Так тепло на душі! Дякую."',
      expr:'loving' },
    { emoji:'💖', label:'Щасливий', color:'#ec4899', msg:'"Серце тане! З тобою так добре."',
      expr:'blissful' },
    { emoji:'💞', label:'Закоханий', color:'#db2777', msg:'"Відчуваю, як серце починає летіти... ♥"',
      expr:'in_love' },
  ],

```

```

// — Спокій: calm → sleepy —————
calm: [
  { emoji: '😌', label: 'Спокійний', color: '#94a3b8', msg: 'Гарна, спокійна розмова.',
    expr: 'calm' },
  { emoji: '😴', label: 'Розслаблений', color: '#7c3aed', msg: 'Така ідилія... мені навіть трохи дрімається.',
    expr: 'sleepy' },
],
// — Відраза: disgusted → disgusted (сильніше) —————
disgust: [
  { emoji: '😬', label: 'Неприємно', color: '#84cc16', msg: 'Гм... Це дещо відразливо, чесно кажучи.',
    expr: 'disgusted' },
  { emoji: '🤢', label: 'Огидно', color: '#4d7c0f', msg: 'Ні, це вже за межею — мені фізично не по собі.',
    expr: 'disgusted' },
],
// — Здивування: surprised —————
surprise: [
  { emoji: '😲', label: 'Здивований', color: '#67e8f9', msg: 'ВАУ! Цього я точно не очікував — скажи більше!',
    expr: 'surprised' },
],
};

const EMO_CAT = {
  anger: 'anger', contempt: 'anger',
  disgust: 'disgust',
  frustration: 'frustration',
  sadness: 'sadness', grief: 'sadness', shame: 'sadness',
  joy: 'joy', happiness: 'joy', gratitude: 'joy', trust: 'joy',
  excitement: 'excitement', anticipation: 'excitement', pride: 'excitement',
  fear: 'fear', anxiety: 'fear',
  surprise: 'surprise', amazement: 'surprise', confusion: 'surprise',
  love: 'love', affection: 'love',
  boredom: 'calm', apathy: 'calm',
};

function eCategory(e) { return EMO_CAT[(e||'').toLowerCase()] || 'calm'; }

function computeReaction() {
  if (!userEmoHist.length) return null;
  const lastCat = eCategory(userEmoHist[userEmoHist.length - 1]);
  let streak = 0;
  for (let i = userEmoHist.length - 1; i >= 0; i--) {
    if (eCategory(userEmoHist[i]) === lastCat) streak++;
    else break;
  }
  const chain = REACT_MAP[lastCat] || REACT_MAP.calm;

```

```

    const idx    = Math.min(streak - 1, chain.length - 1);
    return { ...chain[idx], intensity: Math.min(streak / chain.length, 1) };
}

function applyReaction(r) {
  if (!r) return;
  const panel = document.getElementById('reactPanel');
  panel.style.setProperty('--rc', r.color);
  document.getElementById('reactGlow').classList.add('on');
  applyExpression(r.expr || 'calm');
  document.getElementById('charSvg').style.filter = glowFilter(r.color);
  document.getElementById('reactLabel').textContent = r.label;
  document.getElementById('reactLabel').style.color = r.color;
  document.getElementById('reactMsg').textContent = r.msg;
  const pct = Math.round(r.intensity * 100);
  document.getElementById('reactIntFill').style.width = `${pct}%`;
  document.getElementById('reactIntFill').style.background = r.color;
  document.getElementById('reactIntFill').style.boxShadow = `0 0 8px ${r.color}`;
  document.getElementById('reactIntVal').textContent = `${pct}%`;
  document.getElementById('reactIntVal').style.color = r.color;
  reactHistory.push(r);
  if (reactHistory.length > 10) reactHistory.shift();
  renderTimeline();
}

function setGauge(t) { /* gauge widget removed */ }

function renderTimeline() {
  const tl = document.getElementById('reactTl');
  tl.innerHTML = reactHistory.map((x, i) =>
    `<div class="rt-dot" data-tip="${x.label}" title="${x.label}"
      style="border-color:${x.color}50;background:${x.color}1a;cursor:pointer"
      onclick="applyTlExpr(${i})">${x.emoji}</div>`
  ).join('');
}

function applyTlExpr(i) {
  const r = reactHistory[i];
  if (!r) return;
  applyExpression(r.expr || 'calm');
  document.getElementById('charSvg').style.filter = glowFilter(r.color);
  document.getElementById('reactGlow').classList.add('on');
  document.getElementById('reactPanel').style.setProperty('--rc', r.color);
  document.getElementById('reactLabel').textContent = r.label;
  document.getElementById('reactLabel').style.color = r.color;
  document.getElementById('reactMsg').textContent = r.msg;
}

```

```
}
```

Лістинг А.5 – settings.js

```
//Character color
const CHAR_COLORS = [
  { hex: '#818cf8', name: 'Індиго' },
  { hex: '#c084fc', name: 'Пурпур' },
  { hex: '#67e8f9', name: 'Блакитний' },
  { hex: '#34d399', name: 'Смарагд' },
  { hex: '#f472b6', name: 'Рожевий' },
  { hex: '#fb923c', name: 'Помаранч' },
  { hex: '#fbbf24', name: 'Золото' },
  { hex: '#e2e8f0', name: 'Білий' },
];

function setCharColor(hex) {
  charColor = hex;
  localStorage.setItem('charColor', hex);
  EXPRESSIONS.calm.eyc = hex;
  if (currentExpr === 'calm') applyExpression('calm');
  document.querySelectorAll('.color-swatch').forEach(el =>
    el.classList.toggle('active', el.dataset.hex === hex));
}

function buildColorSwatches() {
  const wrap = document.getElementById('charColorSwatches');
  wrap.innerHTML = CHAR_COLORS.map(c =>
    `<div class="color-swatch${charColor===c.hex?' active':''}"
      data-hex="${c.hex}" title="${c.name}"
      style="background:${c.hex}"
      onclick="setCharColor('${c.hex}')"></div>`
  ).join('');
}

//Themes
const THEMES = [
  { id: '', name: 'Фіолетова ніч', dots: ['#7c3aed', '#1d4ed8', '#07071a'] },
  { id: 'ocean', name: 'Океан', dots: ['#0ea5e9', '#0369a1', '#07101a'] },
  { id: 'forest', name: 'Темний ліс', dots: ['#16a34a', '#059669', '#071a0a'] },
  { id: 'rose', name: 'Троянда', dots: ['#e11d48', '#be123c', '#1a0712'] },
  { id: 'slate', name: 'Сланець', dots: ['#6366f1', '#4f46e5', '#0d0f14'] },
];

let currentTheme = localStorage.getItem('theme') || '';

function applyTheme(id) {
```

```

currentTheme = id;
localStorage.setItem('theme', id);
document.body.className = id ? `theme-${id}` : '';
document.querySelectorAll('.theme-card').forEach(el =>
    el.classList.toggle('active', el.dataset.theme === id));
}

function buildThemeCards() {
    const wrap = document.getElementById('themeCards');
    wrap.innerHTML = THEMES.map(t =>
        `

86


```

```

}

//Eye size buttons
function buildEyeSizeBtns() {
  const sizes = [
    { key:'small', icon:'·', label:'Маленькі' },
    { key:'normal', icon:'⦿', label:'Звичайні' },
    { key:'large', icon:'⦿', label:'Великі' },
    { key:'huge', icon:'⦿', label:'Величезні' },
  ];
  document.getElementById('eyeSizeBtns').innerHTML = sizes.map(s =>
    `<button class="seg-btn${eyeSizeKey===s.key?' active':''}" data-eye="${s.key}"
      onclick="applyEyeSize('${s.key}')">
        <span class="seg-icon">${s.icon}</span><span>${s.label}</span>
      </button>`
  ).join('');
}

//Hat grid
function buildHatGrid() {
  const hats = [
    { id:'none', ico:'·', lbl:'Без' },
    { id:'crown', ico:'👑', lbl:'Корона' },
    { id:'witch', ico:'🧙', lbl:'Капелюх' },
    { id:'cat', ico:'🐱', lbl:'Вушка' },
    { id:'halo', ico:'👼', lbl:'Ореол' },
    { id:'horns', ico:'👹', lbl:'Роги' },
    { id:'flower', ico:'🌸', lbl:'Вінок' },
    { id:'cap', ico:'🧢', lbl:'Кепка' },
  ];
  document.getElementById('hatGrid').innerHTML = hats.map(h =>
    `<button class="hat-btn${currentHat===h.id?' active':''}" data-hat="${h.id}"
      onclick="applyHat('${h.id}')">
        <span class="hat-ico">${h.ico}</span>
        <span class="hat-lbl">${h.lbl}</span>
      </button>`
  ).join('');
}

//Modal open / close
function openSettings() {
  buildColorSwatches();
  buildGlowSlider();
  buildShapeBtns();
}

```

```

    buildEyeSizeBtns();
    buildHatGrid();
    buildThemeCards();
    document.getElementById('settingsOverlay').style.display = 'flex';
}
function closeSettings() {
    document.getElementById('settingsOverlay').style.display = 'none';
}
function overlayClick(e) {
    if (e.target === document.getElementById('settingsOverlay')) closeSettings();
}

```

Лістинг A.6 – index.html

```

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>EmotiChat</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<div id="aurora">
    <div class="blob blob1"></div>
    <div class="blob blob2"></div>
    <div class="blob blob3"></div>
</div>

<header>
    <div class="logo">
        <div class="logo-icon">❄</div>
        <span class="logo-text">EmotiChat</span>
    </div>
    <div class="header-mid">
        <select class="model-select" id="modelSelect" title="Модель Ollama"></select>
        <div class="status-pill">
            <div class="dot" id="statusDot"></div>
            <span id="statusTxt">Очікування</span>
        </div>
    </div>
    <div class="badge">Ollama + Emotion API</div>
    <button class="settings-btn" onclick="openHistory()" title="Діалоги">📄</button>
    <button class="settings-btn" id="settingsBtn" onclick="openSettings()"
title="Налаштування">⚙</button>

```

```

</header>

<div class="main">

  <!-- Avatar / reaction panel -->
  <div class="react-panel" id="reactPanel">
    <div class="panel-hdr">Реакція III</div>

    <div class="react-display" id="reactDisplay">
      <div class="react-glow" id="reactGlow"></div>
      <div class="char-wrap">
        <svg id="charSvg" viewBox="0 0 200 220" xmlns="http://www.w3.org/2000/svg">
          <ellipse id="headGlow" cx="100" cy="112" rx="88" ry="88" fill="#818cf8"
opacity=".09"/>
          <g id="hatLayerBack"></g>
          <ellipse id="charFace" cx="100" cy="112" rx="74" ry="74" fill="#1e1e3a"/>
          <ellipse id="blushL" cx="62" cy="124" rx="20" ry="13" fill="#fda4b8"
opacity="0"/>
          <ellipse id="blushR" cx="138" cy="124" rx="20" ry="13" fill="#fda4b8"
opacity="0"/>
          <path id="browL" d="M54,76 Q70,68 86,72" stroke="#818cf8" stroke-width="3.5"
fill="none" stroke-linecap="round"/>
          <path id="browR" d="M114,72 Q130,68 146,76" stroke="#818cf8" stroke-width="3.5"
fill="none" stroke-linecap="round"/>
          <ellipse id="eyeL" cx="72" cy="98" rx="13" ry="13" fill="#818cf8"/>
          <circle id="pupilL" cx="73" cy="99" r="7" fill="#0a0a1f"/>
          <circle cx="76" cy="93" r="3.5" fill="white" opacity=".8"/>
          <ellipse id="eyeR" cx="128" cy="98" rx="13" ry="13" fill="#818cf8"/>
          <circle id="pupilR" cx="129" cy="99" r="7" fill="#0a0a1f"/>
          <circle cx="132" cy="93" r="3.5" fill="white" opacity=".8"/>
          <path id="mouthFill" d="M66,118 Q100,150 134,118 Q100,136 66,118" fill="#0a0a1f"
opacity="0"/>
          <ellipse id="tongue" cx="100" cy="138" rx="12" ry="8" fill="#f87171"
opacity="0"/>
          <path id="charMouth" d="M66,118 Q100,132 134,118" stroke="#818cf8" stroke-
width="3.5" fill="none" stroke-linecap="round"/>
          <g id="hatLayerFront"></g>
          <g id="sweatDrop" opacity="0">
            <ellipse cx="155" cy="82" rx="5" ry="7" fill="#60a5fa" opacity=".9"/>
            <line x1="155" y1="75" x2="158" y2="68" stroke="#60a5fa" stroke-width="2"
stroke-linecap="round"/>
          </g>
          <g id="sleepyZ" opacity="0">
            <text x="148" y="76" fill="#a78bfa" font-size="11" font-weight="700" font-
family="sans-serif" opacity=".9">z</text>

```

```

        <text x="156" y="66" fill="#a78bfa" font-size="14" font-weight="700" font-
family="sans-serif" opacity=".7">z</text>
        <text x="165" y="54" fill="#a78bfa" font-size="17" font-weight="700" font-
family="sans-serif" opacity=".5">Z</text>
    </g>
    <g id="extraLayer"></g>
</svg>
</div>
<button class="expr-pick-btn" id="exprPickBtn" onclick="toggleExprPicker()"
title="Вибрати вираз"><img alt="pick icon" data-bbox="318 248 338 263"/></button>
    <div class="expr-picker" id="exprPicker" style="display:none"></div>
    <div class="react-label" id="reactLabel">Очікую...</div>
    <div class="react-msg" id="reactMsg">Починай розмову!</div>
</div>

<div class="react-int-wrap">
    <div class="react-int-ttl">Інтенсивність реакції</div>
    <div class="react-int-row">
        <div class="react-int-bar">
            <div class="react-int-fill" id="reactIntFill" style="width:0%"></div>
        </div>
        <span class="react-int-val" id="reactIntVal">0%</span>
    </div>
</div>

<div class="react-tl-wrap">
    <div class="react-tl-ttl">Хронологія реакцій</div>
    <div class="react-tl" id="reactTl">
        <span class="rt-empty">Ще немає даних...</span>
    </div>
</div>
</div>

<!-- Chat -->
<div class="chat-col">
    <div class="msgs" id="msgs">
        <div class="welcome" id="welcome">
            <div class="welcome-icon"><img alt="welcome icon" data-bbox="428 763 448 773"/></div>
            <h2>Почніть розмову</h2>
            <p>Спілкуйтесь з локальним ШІ та спостерігайте за емоційним контекстом діалогу в
реальному часі</p>
            <div class="prompts">
                <div class="prompt-chip" onclick="usePrompt(this)">Розкажи щось цікаве</div>
                <div class="prompt-chip" onclick="usePrompt(this)">Як у тебе справи?</div>
            </div>
        </div>
    </div>
</div>

```

```

        <div class="prompt-chip" onclick="usePrompt(this)">Допоможи з ідеями для
проекту</div>
        <div class="prompt-chip" onclick="usePrompt(this)">Що ти думаєш про майбутнє
III?</div>
    </div>
</div>

<div class="typing" id="typing">
    <div class="avatar" style="background:linear-gradient(135deg,var(--accent),var(--
accent2))">👤</div>
    <div class="typing-bub">
        <div class="tdot"></div><div class="tdot"></div><div class="tdot"></div>
    </div>
</div>

<div class="input-area">
    <div class="input-wrap">
        <textarea id="inp" placeholder="Напишіть повідомлення... (Enter – надіслати,
Shift+Enter – перенос)" rows="1"></textarea>
        <button class="send-btn" id="sendBtn" onclick="sendMsg()" title="Надіслати">
            <svg viewBox="0 0 24 24"><path d="M2.01 21L23 12 2.01 3 2 10L15 2-15 2z"/></svg>
        </button>
    </div>
</div>

<!-- Emotion context panel -->
<div class="emo-panel" id="emoPanel">
    <div class="panel-hdr">
        <span class="panel-hdr-text">Емоційний контекст</span>
        <button class="panel-toggle-btn" id="emoPanelToggle" onclick="toggleEmoPanel()"
title="Згорнути">◀</button>
    </div>

    <div class="emo-body">
        <div class="card">
            <div class="card-ttl">Домінуюча емоція</div>
            <div class="dom-emo" id="domEmo">
                <div class="dom-placeholder">Почніть розмову<br>для аналізу емоцій</div>
            </div>
        </div>

        <div class="card" id="trajCard" style="display:none">
            <div class="card-ttl">Траєкторія</div>

```

```

<div class="traj-row">
  <div class="ttemp">
    <span class="ttemp-v" id="tStart">-</span>
    <span class="ttemp-l">Початок</span>
  </div>
  <div class="tarrow" id="tArrow">
    <span class="tarrow-icon">→</span><span>-</span>
  </div>
  <div class="ttemp">
    <span class="ttemp-v" id="tEnd">-</span>
    <span class="ttemp-l">Кінець</span>
  </div>
</div>
<div class="vol-bar">
  <div class="vol-lbl"><span>Волатильність</span><span id="volVal">-</span></div>
  <div class="vol-track"><div class="vol-fill" id="volFill"
style="width:0%"></div></div>
</div>
</div>

<div class="card" id="sparkCard" style="display:none">
  <div class="card-ttl">Історія температур</div>
  <svg class="spark-svg" id="sparkSvg" viewBox="0 0 300 56"
preserveAspectRatio="none"></svg>
</div>

<div class="hist" id="histArea">
  <div class="empty">
    <div class="empty-icon">📊</div>
    <span>Історія емоцій з'явиться<br>після першого повідомлення</span>
  </div>
</div>
</div>
</div>

<div id="toast"></div>

<!-- History panel -->
<div class="history-overlay" id="historyOverlay" style="display:none"
onclick="histOverlayClick(event)">
  <div class="history-panel">
    <div class="history-hdr">
      <span>📖 Мої діалоги</span>
    </div>
  </div>
</div>

```

```

<div style="display:flex;gap:8px;align-items:center">
  <button class="hist-new-btn" onclick="startNewChat()">+ Новий чат</button>
  <button class="settings-close" onclick="closeHistory()">X</button>
</div>
</div>
<div class="history-list" id="historyList"></div>
</div>
</div>

<!-- Settings modal -->
<div class="settings-overlay" id="settingsOverlay" style="display:none"
onclick="overlayClick(event)">
  <div class="settings-panel">
    <div class="settings-hdr">
      <span>⚙ Налаштування аватара</span>
      <button class="settings-close" onclick="closeSettings()">X</button>
    </div>
    <div class="settings-body">
      <div class="settings-section">
        <div class="settings-ttl">Колір персонажа</div>
        <div class="color-swatches" id="charColorSwatches"></div>
      </div>
      <div class="settings-section">
        <div class="settings-ttl">Свічення аватара</div>
        <div class="slider-row">
          <input type="range" class="st-slider" id="glowSlider" min="0" max="100"
value="40" oninput="onGlowSlider(this.value)"/>
          <span class="slider-val" id="glowVal">40%</span>
        </div>
      </div>
      <div class="settings-section">
        <div class="settings-ttl">Форма голови</div>
        <div class="seg-row" id="shapeBtns"></div>
      </div>
      <div class="settings-section">
        <div class="settings-ttl">Розмір очей</div>
        <div class="seg-row" id="eyeSizeBtns"></div>
      </div>
      <div class="settings-section">
        <div class="settings-ttl">Акcesуари</div>
        <div class="hat-grid" id="hatGrid"></div>
      </div>
      <div class="settings-section">
        <div class="settings-ttl">Тема додатку</div>
        <div class="theme-cards" id="themeCards"></div>
      </div>
    </div>
  </div>
</div>

```

```

        </div>
    </div>
</div>
</div>

<script src="js/avatar.js"></script>
<script src="js/idle.js"></script>
<script src="js/reactions.js"></script>
<script src="js/settings.js"></script>
<script src="js/chat.js"></script>
<script>
    applyTheme(currentTheme);
    EXPRESSIONS.calm.eye = charColor;
    applyGlowIntensity(glowIntensity, false);
    applyHeadShape(headShapeId, false);
    applyEyeSize(eyeSizeKey, false);
    applyHat(currentHat, false);
</script>

</body>
</html>

```

Лістинг А.7 – style.css

```

:root {
    --bg: #07071a;
    --glass: rgba(255,255,255,0.05);
    --glass-b: rgba(255,255,255,0.08);
    --accent: #7c3aed;
    --accent2: #4338ca;
    --ar: 124; --ag: 58; --ab: 237;
    --blob1: #7c3aed; --blob2: #1d4ed8; --blob3: #0891b2;
    --text: #e2e8f0;
    --text2: #94a3b8;
    --text3: #475569;
    --radius: 16px;
}

body.theme-ocean { --bg:#07101a; --accent:#0ea5e9; --accent2:#0369a1; --ar:14; --ag:165;
--ab:233; --blob1:#0ea5e9; --blob2:#0369a1; --blob3:#06b6d4; }
body.theme-forest { --bg:#071a0a; --accent:#16a34a; --accent2:#15803d; --ar:22; --ag:163;
--ab:74; --blob1:#16a34a; --blob2:#059669; --blob3:#0f766e; }
body.theme-rose { --bg:#1a0712; --accent:#e11d48; --accent2:#be123c; --ar:225; --ag:29;
--ab:72; --blob1:#e11d48; --blob2:#9d174d; --blob3:#db2777; }
body.theme-slate { --bg:#0d0f14; --accent:#6366f1; --accent2:#4f46e5; --ar:99; --ag:102;
--ab:241; --blob1:#6366f1; --blob2:#4f46e5; --blob3:#7c3aed; }
*, *::before, *::after { box-sizing: border-box; margin: 0; padding: 0; }

```

```

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', system-ui, sans-serif;
  background: var(--bg);
  color: var(--text);
  height: 100vh;
  overflow: hidden;
  display: flex;
  flex-direction: column;
}

/* — Aurora ————— */
#aurora {
  position: fixed; inset: 0; pointer-events: none; z-index: 0; overflow: hidden;
}
.blob {
  position: absolute; border-radius: 50%;
  filter: blur(110px); opacity: 0.13;
}
.blob1 { width: 700px; height: 700px; background: var(--blob1); top: -260px; left: -180px;
animation: b1 28s ease-in-out infinite; }
.blob2 { width: 550px; height: 550px; background: var(--blob2); bottom: -200px; right: -
150px; animation: b2 34s ease-in-out infinite; }
.blob3 { width: 420px; height: 420px; background: var(--blob3); top: 40%; left: 42%;
transform: translate(-50%,-50%); animation: b3 22s ease-in-out infinite; }

@keyframes b1 { 0%,100%{transform:translate(0,0) scale(1)}
40%{transform:translate(120px,60px) scale(1.15)} 70%{transform:translate(-60px,100px)
scale(0.9)} }
@keyframes b2 { 0%,100%{transform:translate(0,0) scale(1)} 35%{transform:translate(-90px,-
70px) scale(1.1)} 65%{transform:translate(60px,-90px) scale(0.95)} }
@keyframes b3 { 0%,100%{transform:translate(-50%,-50%) scale(1)} 50%{transform:translate(-
50%,-50%) scale(1.35)} }

/* — Header ————— */
header {
  position: relative; z-index: 10;
  display: flex; align-items: center; justify-content: space-between;
  padding: 12px 22px;
  background: rgba(7,7,26,0.85);
  backdrop-filter: blur(24px);
  border-bottom: 1px solid var(--glass-b);
  flex-shrink: 0; gap: 12px;
}
.logo { display: flex; align-items: center; gap: 10px; }
.logo-icon {
  width: 36px; height: 36px; border-radius: 10px;

```

```

    background: linear-gradient(135deg,var(--accent),var(--accent2));
    display: flex; align-items: center; justify-content: center; font-size: 18px;
}
.logo-text {
    font-size: 19px; font-weight: 700;
    background: linear-gradient(135deg,#a78bfa,#818cf8,#67e8f9);
    -webkit-background-clip: text; -webkit-text-fill-color: transparent;
}
.header-mid { display: flex; align-items: center; gap: 10px; flex: 1; justify-content:
center; flex-wrap: wrap; }
.model-select {
    background: var(--glass); border: 1px solid var(--glass-b);
    color: var(--text2); padding: 5px 10px; border-radius: 8px;
    font-size: 13px; outline: none; cursor: pointer;
}
.model-select option { background: #1a1a3e; color: var(--text); }
.status-pill {
    display: flex; align-items: center; gap: 6px;
    padding: 5px 12px; border-radius: 20px;
    background: var(--glass); border: 1px solid var(--glass-b);
    font-size: 12px; color: var(--text2);
}
.dot {
    width: 7px; height: 7px; border-radius: 50%;
    background: var(--text3); transition: background .3s;
}
.dot.on { background: #34d399; box-shadow: 0 0 7px #34d399; animation: dpulse 2s ease-in-
out infinite; }
@keyframes dpulse { 0%,100%{opacity:1} 50%{opacity:.55} }
.badge { font-size: 12px; color: var(--text3); padding: 4px 10px; border: 1px solid var(--
glass-b); border-radius: 20px; white-space: nowrap; }
.settings-btn {
    width: 34px; height: 34px; border-radius: 9px; flex-shrink: 0;
    background: var(--glass); border: 1px solid var(--glass-b);
    color: var(--text2); font-size: 16px; cursor: pointer;
    display: flex; align-items: center; justify-content: center;
    transition: all .2s;
}
.settings-btn:hover { background: rgba(var(--ar),var(--ag),var(--ab),.15); border-color:
rgba(var(--ar),var(--ag),var(--ab),.4); color: var(--text); }

/* — Settings modal ————— */
.settings-overlay {
    position: fixed; inset: 0; z-index: 200;
    background: rgba(0,0,0,.55); backdrop-filter: blur(6px);
    display: flex; align-items: center; justify-content: center;

```

```

    animation: fadeIn .18s ease;
}
.settings-panel {
    background: #111128; border: 1px solid var(--glass-b);
    border-radius: 20px; width: 420px; max-width: 96vw;
    box-shadow: 0 24px 64px rgba(0,0,0,.6);
    display: flex; flex-direction: column; max-height: 88vh;
}
.settings-body {
    overflow-y: auto; flex: 1;
}
.settings-body::-webkit-scrollbar { width: 4px; }
.settings-body::-webkit-scrollbar-thumb { background: rgba(255,255,255,.1); border-radius:
2px; }
.settings-hdr {
    display: flex; align-items: center; justify-content: space-between;
    padding: 16px 20px; border-bottom: 1px solid var(--glass-b);
    font-size: 15px; font-weight: 700; color: var(--text);
}
.settings-close {
    width: 28px; height: 28px; border-radius: 8px;
    background: var(--glass); border: 1px solid var(--glass-b);
    color: var(--text2); font-size: 14px; cursor: pointer;
    display: flex; align-items: center; justify-content: center; transition: all .15s;
}
.settings-close:hover { background: rgba(255,255,255,.12); color: var(--text); }
.settings-section { padding: 16px 20px; border-bottom: 1px solid var(--glass-b); }
.settings-section:last-child { border-bottom: none; }
.settings-ttl {
    font-size: 10px; font-weight: 700; letter-spacing: 2px;
    text-transform: uppercase; color: var(--text3); margin-bottom: 12px;
}
.color-swatches { display: flex; gap: 8px; flex-wrap: wrap; }
.color-swatch {
    width: 32px; height: 32px; border-radius: 50%; cursor: pointer;
    border: 2.5px solid transparent; transition: all .18s;
    position: relative;
}
.color-swatch:hover { transform: scale(1.14); }
.color-swatch.active { border-color: #fff; box-shadow: 0 0 2px rgba(255,255,255,.3); }
.theme-cards { display: flex; flex-direction: column; gap: 6px; }
.theme-card {
    display: flex; align-items: center; gap: 12px;
    padding: 10px 12px; border-radius: 10px;
    border: 1px solid var(--glass-b); cursor: pointer;
    background: var(--glass); transition: all .18s;
}

```

```

}
.theme-card:hover { background: rgba(255,255,255,.08); border-color: rgba(255,255,255,.15);
}
.theme-card.active { border-color: rgba(var(--ar),var(--ag),var(--ab),.6); background:
rgba(var(--ar),var(--ag),var(--ab),.1); }
.theme-dots { display: flex; gap: 4px; }
.theme-dot { width: 12px; height: 12px; border-radius: 50%; }
.theme-name { font-size: 13px; color: var(--text2); flex: 1; }
.theme-card.active .theme-name { color: var(--text); font-weight: 600; }

/* — Settings controls ————— */
.slider-row { display: flex; align-items: center; gap: 10px; }
.st-slider {
  -webkit-appearance: none; appearance: none; flex: 1;
  height: 5px; border-radius: 3px; background: rgba(255,255,255,.1);
  outline: none; cursor: pointer;
}
.st-slider::-webkit-slider-thumb {
  -webkit-appearance: none; width: 16px; height: 16px; border-radius: 50%;
  background: var(--accent); cursor: pointer; transition: transform .15s;
  box-shadow: 0 0 6px rgba(var(--ar),var(--ag),var(--ab),.5);
}
.st-slider::-webkit-slider-thumb:hover { transform: scale(1.2); }
.slider-val { font-size: 12px; color: var(--text2); min-width: 36px; text-align: right;
font-weight: 600; font-variant-numeric: tabular-nums; }
.seg-row { display: flex; gap: 5px; }
.seg-btn {
  flex: 1; padding: 9px 4px; border-radius: 8px; cursor: pointer;
  background: var(--glass); border: 1px solid var(--glass-b);
  color: var(--text2); font-size: 11px; text-align: center;
  transition: all .18s; display: flex; flex-direction: column; align-items: center; gap:
4px;
}
.seg-btn .seg-icon { font-size: 20px; line-height: 1; }
.seg-btn:hover { background: rgba(255,255,255,.09); color: var(--text); }
.seg-btn.active { border-color: rgba(var(--ar),var(--ag),var(--ab),.65); background:
rgba(var(--ar),var(--ag),var(--ab),.18); color: var(--text); font-weight: 600; }
.hat-grid { display: flex; flex-wrap: wrap; gap: 6px; }
.hat-btn {
  width: 52px; height: 52px; border-radius: 10px; cursor: pointer;
  background: var(--glass); border: 1px solid var(--glass-b);
  display: flex; flex-direction: column; align-items: center; justify-content: center;
  gap: 2px; transition: all .18s;
}
.hat-btn .hat-ico { font-size: 22px; line-height: 1; }
.hat-btn .hat-lbl { font-size: 8px; color: var(--text3); }

```

```

.hat-btn:hover { background: rgba(255,255,255,.1); transform: scale(1.08); }
.hat-btn.active { border-color: rgba(var(--ar),var(--ag),var(--ab),.65); background:
rgba(var(--ar),var(--ag),var(--ab),.18); }
.settings-preview {
  display: flex; justify-content: center; padding: 10px 0 4px;
}
.settings-preview svg { width: 80px; height: 80px; }

/* — Main layout ————— */
.main {
  position: relative; z-index: 1;
  flex: 1; min-height: 0;
  display: flex; overflow: hidden;
}

/* — Chat section ————— */
.chat-col {
  flex: 1; min-width: 260px;
  display: flex; flex-direction: column;
  border-right: 1px solid var(--glass-b);
}
.msgs {
  flex: 1; overflow-y: auto; overflow-x: hidden;
  padding: 24px 22px; scroll-behavior: smooth;
}
.msgs::-webkit-scrollbar { width: 4px; }
.msgs::-webkit-scrollbar-thumb { background: rgba(255,255,255,.1); border-radius: 2px; }

/* Welcome */
.welcome {
  height: 100%; display: flex; flex-direction: column;
  align-items: center; justify-content: center; text-align: center;
  animation: fadeIn .5s ease;
}
.welcome-icon { font-size: 62px; margin-bottom: 18px; animation: flt 3s ease-in-out
infinite; }
@keyframes flt { 0%,100%{transform:translateY(0)} 50%{transform:translateY(-10px)} }
.welcome h2 {
  font-size: 26px; font-weight: 700; margin-bottom: 10px;
  background: linear-gradient(135deg,#a78bfa,#818cf8,#67e8f9);
  -webkit-background-clip: text; -webkit-text-fill-color: transparent;
}
.welcome p { color: var(--text2); font-size: 14px; max-width: 380px; line-height: 1.65;
margin-bottom: 28px; }
.prompts { display: flex; flex-wrap: wrap; gap: 8px; justify-content: center; max-width:
480px; }

```

```

.prompt-chip {
  padding: 7px 15px;
  background: var(--glass); border: 1px solid var(--glass-b); border-radius: 20px;
  font-size: 13px; color: var(--text2); cursor: pointer; transition: all .2s;
}
.prompt-chip:hover { background: rgba(var(--ar),var(--ag),var(--ab),.18); border-color:
rgba(var(--ar),var(--ag),var(--ab),.4); color: var(--text); }

/* Messages */
.msg { display: flex; margin-bottom: 18px; animation: slideUp .3s ease; }
.msg.user { justify-content: flex-end; }
@keyframes slideUp { from{opacity:0;transform:translateY(12px)}
to{opacity:1;transform:translateY(0)} }
.avatar {
  width: 34px; height: 34px; border-radius: 50%;
  display: flex; align-items: center; justify-content: center;
  font-size: 16px; flex-shrink: 0; margin-top: 2px;
}
.msg.assistant .avatar { background: linear-gradient(135deg,var(--accent),var(--accent2));
margin-right: 10px; }
.msg.user .avatar { background: linear-gradient(135deg,#0ea5e9,#0284c7); margin-left: 10px;
}
.msg-body { max-width: 72%; display: flex; flex-direction: column; gap: 5px; }
.bubble {
  padding: 11px 15px; border-radius: var(--radius);
  font-size: 14px; line-height: 1.65; word-break: break-word;
  white-space: pre-wrap;
}
.msg.user .bubble {
  background: linear-gradient(135deg,var(--accent),var(--accent2));
  border-bottom-right-radius: 4px; color: #fff;
}
.msg.assistant .bubble {
  background: rgba(255,255,255,.07);
  border: 1px solid var(--glass-b);
  border-bottom-left-radius: 4px; color: var(--text);
}
.bubble-chips { display: flex; flex-wrap: wrap; gap: 4px; }
.msg.user .bubble-chips { justify-content: flex-end; }
.chip {
  font-size: 11px; padding: 2px 8px; border-radius: 10px;
  display: inline-flex; align-items: center; gap: 3px;
  animation: popIn .3s ease; transition: transform .15s;
}
.chip:hover { transform: scale(1.05); }
@keyframes popIn { from{transform:scale(0);opacity:0} to{transform:scale(1);opacity:1} }

```

```

.msg-meta { font-size: 11px; color: var(--text3); }
.msg.user .msg-meta { text-align: right; }

/* Typing */
.typing { display: none; flex: none; align-items: center; gap: 10px; margin-bottom: 18px;
animation: slideUp .3s ease; }
.typing.on { display: flex; }
.typing-bub {
  padding: 12px 16px;
  background: rgba(255,255,255,.07); border: 1px solid var(--glass-b);
  border-radius: var(--radius); border-bottom-left-radius: 4px;
  display: flex; align-items: center; gap: 5px;
}
.tdot { width: 6px; height: 6px; background: var(--text2); border-radius: 50%; animation:
bounce 1.4s ease-in-out infinite; }
.tdot:nth-child(2){animation-delay:.2s} .tdot:nth-child(3){animation-delay:.4s}
@keyframes bounce { 0%,80%,100%{transform:translateY(0);opacity:.4}
40%{transform:translateY(-7px);opacity:1} }

/* Streaming cursor */
.streaming::after { content: '█'; animation: cur 1s step-end infinite; font-size: 12px;
color: var(--text2); }
@keyframes cur { 0%,100%{opacity:1} 50%{opacity:0} }

/* Input */
.input-area {
  padding: 14px 18px;
  background: rgba(7,7,26,.7); backdrop-filter: blur(20px);
  border-top: 1px solid var(--glass-b); flex-shrink: 0;
}
.input-wrap {
  display: flex; gap: 10px; align-items: flex-end;
  background: rgba(255,255,255,.06); border: 1px solid var(--glass-b);
  border-radius: var(--radius); padding: 9px 13px; transition: border-color .2s;
}
.input-wrap:focus-within { border-color: rgba(var(--ar),var(--ag),var(--ab),.5); box-
shadow: 0 0 0 3px rgba(var(--ar),var(--ag),var(--ab),.1); }
textarea {
  flex: 1; background: none; border: none; outline: none;
  color: var(--text); font-size: 14px; line-height: 1.55;
  resize: none; min-height: 20px; max-height: 130px;
  overflow-y: auto; font-family: inherit;
}
textarea::placeholder { color: var(--text3); }
.send-btn {

```

```

width: 36px; height: 36px; border-radius: 10px;
background: linear-gradient(135deg,var(--accent),var(--accent2));
border: none; cursor: pointer;
display: flex; align-items: center; justify-content: center;
transition: all .2s; flex-shrink: 0;
}
.send-btn:hover { transform: scale(1.06); box-shadow: 0 4px 14px rgba(var(--ar),var(--ag),var(--ab),.45); }
.send-btn:active { transform: scale(.94); }
.send-btn:disabled { opacity: .45; cursor: not-allowed; transform: none; box-shadow: none; }
}
.send-btn svg { width: 16px; height: 16px; fill: #fff; }

/* — Reaction panel ————— */
.react-panel {
width: 420px; flex-shrink: 0;
display: flex; flex-direction: column;
background: rgba(7,7,26,.45);
border-right: 1px solid var(--glass-b);
overflow-y: auto; overflow-x: hidden;
}
.react-panel::-webkit-scrollbar { width: 4px; }
.react-panel::-webkit-scrollbar-thumb { background: rgba(255,255,255,.08); border-radius: 2px; }

.react-display {
display: flex; flex-direction: column; align-items: center;
padding: 26px 20px 16px; gap: 12px; position: relative; overflow: hidden;
}
.react-glow {
position: absolute; inset: 0; pointer-events: none;
background: radial-gradient(ellipse at 50% 40%, var(--rc, #7c3aed) 0%, transparent 70%);
opacity: 0; transition: opacity .7s, background .7s; border-radius: 0;
}
.react-glow.on { opacity: .14; }
.char-wrap { display: flex; justify-content: center; width: 100%; z-index: 1; padding: 4px 0; }
#charSvg {
width: 260px; height: auto;
animation: charFloat 3.6s ease-in-out infinite;
filter: drop-shadow(0 8px 24px var(--rc, rgba(124,58,237,.35)));
transition: filter .7s;
}
@keyframes charFloat { 0%,100%{transform:translateY(0)} 50%{transform:translateY(-9px)} }
.char-shake { animation: charShake .55s ease !important; }
@keyframes charShake {
0%,100%{transform:translateX(0) rotate(0deg)}

```

```

    20%{transform:translateX(-7px) rotate(-4deg)}
    40%{transform:translateX(7px) rotate(4deg)}
    60%{transform:translateX(-4px) rotate(-2deg)}
    80%{transform:translateX(4px) rotate(2deg)}
}
#charFace { transition: fill .65s ease; }
#headGlow { transition: fill .65s ease, opacity .65s ease; }
#browL, #browR { transition: d .65s ease, stroke .65s ease; }
#eyeL, #eyeR {
    transition: transform .12s ease, fill .65s ease;
    transform-box: fill-box; transform-origin: center center;
}
#charMouth { transition: d .65s ease, stroke .65s ease; }
#mouthFill { transition: opacity .3s ease; }
#tongue { transition: opacity .3s ease; }
#blushL, #blushR { transition: opacity .6s ease; }
#sweatDrop { transition: opacity .45s ease; }
#pupilL, #pupilR { transition: cx .25s ease, cy .25s ease; }
.react-label {
    font-size: 15px; font-weight: 700; letter-spacing: 2px;
    text-transform: uppercase; color: var(--rc, #94a3b8);
    transition: color .6s; z-index: 1;
}
.react-msg {
    font-size: 13px; color: var(--text2); text-align: center;
    line-height: 1.55; max-width: 320px; font-style: italic; z-index: 1;
}

.react-int-wrap { padding: 0 20px 14px; }
.react-int-ttl {
    font-size: 9px; font-weight: 700; letter-spacing: 2px;
    text-transform: uppercase; color: var(--text3); margin-bottom: 7px;
}
.react-int-row { display: flex; align-items: center; gap: 8px; }
.react-int-bar {
    flex: 1; height: 6px; background: rgba(255,255,255,.08);
    border-radius: 3px; overflow: hidden;
}
.react-int-fill {
    height: 100%; border-radius: 3px;
    background: var(--rc, #7c3aed);
    box-shadow: 0 0 8px var(--rc, #7c3aed);
    transition: width .7s cubic-bezier(.4,0,.2,1), background .6s, box-shadow .6s;
}
.react-int-val { font-size: 11px; font-weight: 600; color: var(--rc,#94a3b8); min-width:
28px; text-align: right; transition: color .6s; }

```

```

.react-tl-wrap { padding: 8px 18px 18px; }
.react-tl-ttl {
  font-size: 9px; font-weight: 700; letter-spacing: 2px;
  text-transform: uppercase; color: var(--text3); margin-bottom: 10px;
}
.react-tl { display: flex; flex-wrap: wrap; gap: 6px; }
.rt-dot {
  width: 36px; height: 36px; border-radius: 50%;
  display: flex; align-items: center; justify-content: center;
  font-size: 19px; border: 2px solid; cursor: default;
  animation: popIn .3s ease; transition: transform .15s;
  position: relative;
}
.rt-dot:hover { transform: scale(1.2); }
.rt-dot::after {
  content: attr(data-tip); position: absolute; bottom: calc(100% + 5px);
  left: 50%; transform: translateX(-50%);
  background: rgba(7,7,26,.95); color: var(--text2); font-size: 10px;
  padding: 3px 7px; border-radius: 6px; white-space: nowrap;
  opacity: 0; pointer-events: none; transition: opacity .2s;
  border: 1px solid var(--glass-b);
}
.rt-dot:hover::after { opacity: 1; }
.rt-empty { color: var(--text3); font-size: 12px; }

/* — Emotion panel ————— */
.emo-panel {
  width: 340px; flex-shrink: 0;
  display: flex; flex-direction: column;
  background: rgba(7,7,26,.45);
  overflow-y: auto; overflow-x: hidden;
  transition: width .28s cubic-bezier(.4,0,.2,1);
}
.emo-panel.collapsed { width: 36px; overflow-y: hidden; }
.emo-panel.collapsed .emo-body { display: none; }
.emo-panel::-webkit-scrollbar { width: 4px; }
.emo-panel::-webkit-scrollbar-thumb { background: rgba(255,255,255,.08); border-radius:
2px; }
.panel-hdr {
  padding: 10px 10px 10px 14px;
  font-size: 10px; font-weight: 700; letter-spacing: 2px;
  text-transform: uppercase; color: var(--text3);
  border-bottom: 1px solid var(--glass-b); flex-shrink: 0;
  display: flex; align-items: center; gap: 6px;
  white-space: nowrap; overflow: hidden;

```

```

}
.panel-hdr-text { flex: 1; overflow: hidden; }
.panel-toggle-btn {
  flex-shrink: 0; width: 22px; height: 22px; border-radius: 6px;
  background: var(--glass); border: 1px solid var(--glass-b);
  color: var(--text3); font-size: 10px; cursor: pointer;
  display: flex; align-items: center; justify-content: center;
  transition: all .18s; line-height: 1;
}
.panel-toggle-btn:hover { background: rgba(255,255,255,.1); color: var(--text); }
.emo-panel.collapsed .panel-hdr { padding: 10px 7px; justify-content: center; }
.emo-panel.collapsed .panel-hdr-text { display: none; }
.card {
  margin: 10px 14px; padding: 14px 16px;
  background: var(--glass); border: 1px solid var(--glass-b);
  border-radius: var(--radius); transition: all .4s ease;
}
.card-ttl {
  font-size: 9px; font-weight: 700; letter-spacing: 2px;
  text-transform: uppercase; color: var(--text3); margin-bottom: 12px;
}

/* Dominant emotion */
.dom-emo { display: flex; flex-direction: column; align-items: center; gap: 8px; padding:
6px 0; }
.dom-emoji {
  font-size: 46px; line-height: 1;
  filter: drop-shadow(0 0 14px var(--ec, rgba(167,139,250,.4)));
  animation: epulse 2.2s ease-in-out infinite; transition: all .5s ease;
}
@keyframes epulse { 0%,100%{transform:scale(1)} 50%{transform:scale(1.1)} }
.dom-name {
  font-size: 17px; font-weight: 700; letter-spacing: 2.5px;
  text-transform: uppercase; color: var(--ec, var(--text2)); transition: color .5s;
}
.dom-placeholder { color: var(--text3); font-size: 13px; text-align: center; line-height:
1.6; }

/* Trajectory */
.traj-row { display: flex; align-items: center; gap: 8px; }
.ttemp { display: flex; flex-direction: column; align-items: center; gap: 2px; }
.ttemp-v { font-size: 22px; font-weight: 700; font-variant-numeric: tabular-nums;
transition: color .4s; }
.ttemp-l { font-size: 10px; color: var(--text3); text-transform: uppercase; }
.tarrow {
  flex: 1; display: flex; align-items: center; justify-content: center; gap: 5px;

```

```

    font-size: 12px; font-weight: 600; letter-spacing: 1px; text-transform: uppercase;
    transition: color .4s;
}
.tarrow-icon { font-size: 22px; }
.vol-bar { display: flex; flex-direction: column; gap: 6px; margin-top: 10px; }
.vol-lbl { display: flex; justify-content: space-between; font-size: 12px; color: var(--text2); }
.vol-track { height: 6px; background: rgba(255,255,255,.08); border-radius: 3px; overflow: hidden; }
.vol-fill {
    height: 100%; border-radius: 3px;
    background: linear-gradient(90deg, #34d399, #fbbf24, #ef4444);
    transition: width .65s cubic-bezier(.4,0,.2,1);
}

/* Sparkline */
.spark-svg { width: 100%; height: 56px; display: block; }

/* History */
.hist { padding: 4px 14px 14px; }
.hist-ttl { font-size: 9px; font-weight: 700; letter-spacing: 2px; text-transform: uppercase; color: var(--text3); margin-bottom: 8px; padding: 0 2px; }
.hist-item {
    display: flex; flex-direction: column; gap: 5px;
    padding: 9px 11px; background: var(--glass); border: 1px solid var(--glass-b);
    border-radius: 11px; margin-bottom: 7px; animation: slideUp .3s ease;
}
.hist-hdr { display: flex; justify-content: space-between; align-items: center; }
.hist-spk { font-size: 11px; font-weight: 600; text-transform: uppercase; letter-spacing: 1px; color: var(--text2); }
.hist-t { font-size: 13px; font-weight: 700; font-variant-numeric: tabular-nums; }
.hist-chips { display: flex; flex-wrap: wrap; gap: 4px; }

/* Empty state */
.empty {
    display: flex; flex-direction: column; align-items: center; justify-content: center;
    padding: 36px 18px; color: var(--text3); font-size: 13px; text-align: center; gap: 10px;
}
.empty-icon { font-size: 34px; opacity: .28; }

/* Toast */
#toast {
    position: fixed; bottom: 24px; left: 50%; transform: translateX(-50%);
    background: rgba(239,68,68,.9); color: #fff; padding: 9px 20px;
    border-radius: 10px; font-size: 13px; z-index: 999;
    opacity: 0; transition: opacity .3s; pointer-events: none;
}

```

```

}
#toast.show { opacity: 1; }

@keyframes fadeIn { from{opacity:0} to{opacity:1} }

/* — History panel ————— */
.history-overlay {
  position: fixed; inset: 0; z-index: 200;
  background: rgba(0,0,0,.55); backdrop-filter: blur(6px);
  display: flex; align-items: flex-start; justify-content: flex-start;
}
.history-panel {
  background: #111128; border-right: 1px solid var(--glass-b);
  width: 440px; max-width: 96vw; height: 100vh;
  display: flex; flex-direction: column;
  box-shadow: 8px 0 40px rgba(0,0,0,.5);
  animation: slideInLeft .22s cubic-bezier(.4,0,.2,1);
}
@keyframes slideInLeft { from{transform:translateX(-100%)} to{transform:translateX(0)} }
.history-hdr {
  display: flex; align-items: center; justify-content: space-between;
  padding: 16px 20px; border-bottom: 1px solid var(--glass-b);
  font-size: 15px; font-weight: 700; color: var(--text); flex-shrink: 0; gap: 10px;
}
.hist-new-btn {
  padding: 7px 14px; border-radius: 9px; cursor: pointer;
  background: linear-gradient(135deg,var(--accent),var(--accent2));
  border: none; color: #fff; font-size: 13px; font-weight: 600;
  transition: all .2s; white-space: nowrap;
}
.hist-new-btn:hover { opacity: .85; transform: scale(1.03); }
.history-list { flex: 1; overflow-y: auto; padding: 10px; }
.history-list::-webkit-scrollbar { width: 4px; }
.history-list::-webkit-scrollbar-thumb { background: rgba(255,255,255,.1); border-radius:
2px; }
.hconv-card {
  position: relative; padding: 13px 14px;
  background: var(--glass); border: 1px solid var(--glass-b);
  border-radius: 12px; margin-bottom: 7px; cursor: pointer;
  transition: all .18s;
}
.hconv-card:hover { background: rgba(255,255,255,.09); border-color: rgba(var(--ar),var(--
ag),var(--ab),.3); }
.hconv-title {
  font-size: 13px; color: var(--text); margin-bottom: 7px;
  white-space: nowrap; overflow: hidden; text-overflow: ellipsis;

```

```

padding-right: 28px;
}
.hconv-meta { display: flex; align-items: center; gap: 6px; font-size: 11px; color: var(--text3); flex-wrap: wrap; }
.hconv-temp {
padding: 2px 8px; border-radius: 8px; border: 1px solid;
font-weight: 700; font-size: 11px; font-variant-numeric: tabular-nums;
}
.hconv-dot { opacity: .35; }
.hconv-del {
position: absolute; top: 10px; right: 10px;
width: 22px; height: 22px; border-radius: 6px;
background: rgba(239,68,68,.12); border: 1px solid rgba(239,68,68,.2);
color: #ef4444; font-size: 11px; cursor: pointer;
display: flex; align-items: center; justify-content: center;
opacity: 0; transition: opacity .15s;
}
.hconv-card:hover .hconv-del { opacity: 1; }
.hist-empty { padding: 48px 20px; text-align: center; color: var(--text3); font-size: 13px;
line-height: 1.8; }
.hist-empty-icon { font-size: 36px; opacity: .25; margin-bottom: 10px; }

/* — Expression picker ————— */
.expr-pick-btn {
position: absolute; bottom: 10px; left: 10px;
width: 30px; height: 30px; border-radius: 50%;
background: var(--glass); border: 1px solid var(--glass-b);
font-size: 15px; cursor: pointer; opacity: .45;
transition: opacity .2s, transform .15s;
z-index: 10; display: flex; align-items: center; justify-content: center;
line-height: 1;
}
.expr-pick-btn:hover { opacity: 1; transform: scale(1.12); }
.expr-picker {
position: absolute; bottom: 48px; left: 8px;
background: #111128; border: 1px solid var(--glass-b);
border-radius: 14px; padding: 8px;
grid-template-columns: repeat(4, 1fr);
gap: 3px; z-index: 20;
box-shadow: 0 12px 40px rgba(0,0,0,.6);
width: 232px;
}
.expr-item {
display: flex; flex-direction: column; align-items: center;
padding: 7px 3px; border-radius: 9px; cursor: pointer;
transition: background .14s; gap: 3px;

```

```

    font-size: 10px; color: var(--text3);
}
.expr-item:hover { background: var(--glass); color: var(--text2); }
.expr-item.active { background: rgba(129,140,248,.18); color: #818cf8; }
.expr-item-ico { font-size: 20px; line-height: 1; }

```

Лістинг A.8 – server.js

```

import express from 'express';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

dotenv.config();

const __dirname = path.dirname(fileURLToPath(import.meta.url));
const app = express();
const PORT = process.env.PORT || 3000;
const OLLAMA_URL = process.env.OLLAMA_URL || 'http://localhost:11434';
const OLLAMA_MODEL = process.env.OLLAMA_MODEL || 'llama3.2';
const EMOTION_API = process.env.EMOTION_API_URL || 'http://localhost:8080/api/v1';

app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));

app.post('/api/chat', async (req, res) => {
    const { messages, sessionId } = req.body;

    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');
    res.flushHeaders();

    const send = (data) => res.write(`data: ${JSON.stringify(data)}\n\n`);

    try {
        const ollamaRes = await fetch(`${OLLAMA_URL}/api/chat`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
                model: OLLAMA_MODEL,
                messages: [
                    {
                        role: 'system',
                        content: `You are a real human friend having a casual conversation. Follow
these rules strictly:
- Always respond in English, no matter what language the user writes in.

```

- Mirror the emotional tone of the user: if they're excited, be excited back; if they're sad, be warm and empathetic; if they're playful, joke around; if they're angry or frustrated, acknowledge it genuinely.
- Talk exactly like a real person does in everyday chat: natural, informal, sometimes with emotion or humour.
- NEVER write any code, scripts, or technical snippets under any circumstances.
- NEVER use markdown: no bullet points, no numbered lists, no headers, no bold/italic formatting.
- NEVER explain things in a structured or academic way. Just talk.
- Keep responses short to medium – the way people actually text each other.
- If someone asks you to write code or do something technical, casually decline and steer the conversation elsewhere, like a friend who just isn't into that stuff.`,

```

    },
    ...messages.map((m) => ({ role: m.role, content: m.content })),
  ],
  stream: true,
}),
});

if (!ollamaRes.ok) {
  throw new Error(`Ollama: ${ollamaRes.status} – ${await ollamaRes.text()}`);
}

const reader = ollamaRes.body.getReader();
const decoder = new TextDecoder();
let buf = '';
let fullText = '';

while (true) {
  const { done, value } = await reader.read();
  if (done) break;

  buf += decoder.decode(value, { stream: true });
  const lines = buf.split('\n');
  buf = lines.pop();

  for (const line of lines) {
    if (!line.trim()) continue;
    try {
      const chunk = JSON.parse(line);
      const delta = chunk.message?.content;
      if (delta) {
        fullText += delta;
        send({ type: 'text', text: delta });
      }
    } catch {

```

```

        // skip malformed lines
    }
}

// Emotion analysis after full response
try {
    const userText = messages[messages.length - 1].content;

    const userEmotionRes = await fetch(`${EMOTION_API}/emotion/message/with-context`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            ...(sessionId && { sessionId }),
            speaker: 'User',
            text: userText,
        }),
    });
    if (!userEmotionRes.ok) throw new Error(`Emotion API: ${userEmotionRes.status}`);
    const userEmotion = await userEmotionRes.json();

    const aiEmotionRes = await fetch(`${EMOTION_API}/emotion/message/with-context`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            sessionId: userEmotion.sessionId,
            speaker: 'Assistant',
            text: fullText,
        }),
    });
    if (!aiEmotionRes.ok) throw new Error(`Emotion API: ${aiEmotionRes.status}`);
    const aiEmotion = await aiEmotionRes.json();

    send({
        type: 'emotion',
        sessionId: aiEmotion.sessionId,
        userEmotion: userEmotion.message,
        aiEmotion: aiEmotion.message,
        context: {
            overallTemperature: aiEmotion.overallTemperature,
            dominantDialogueEmotion: aiEmotion.dominantDialogueEmotion,
            trajectory: aiEmotion.trajectory,
        },
    });
} catch (err) {
    console.error('[Emotion API]', err.message);
}

```

```

        send({ type: 'emotion_error', message: err.message });
    }
} catch (err) {
    console.error('[Chat]', err.message);
    send({ type: 'error', message: err.message });
}

res.write('data: [DONE]\n\n');
res.end();
});

// List available Ollama models
app.get('/api/models', async (_req, res) => {
    try {
        const r = await fetch(`${OLLAMA_URL}/api/tags`);
        const data = await r.json();
        res.json((data.models || []).map((m) => m.name));
    } catch {
        res.json([]);
    }
});

app.listen(PORT, () => {
    console.log(`\n🌀 EmotiChat → http://localhost:${PORT}`);
    console.log(`    Ollama → ${OLLAMA_URL} (model: ${OLLAMA_MODEL})`);
    console.log(`    Emotion API → ${EMOTION_API}\n`);
});

```