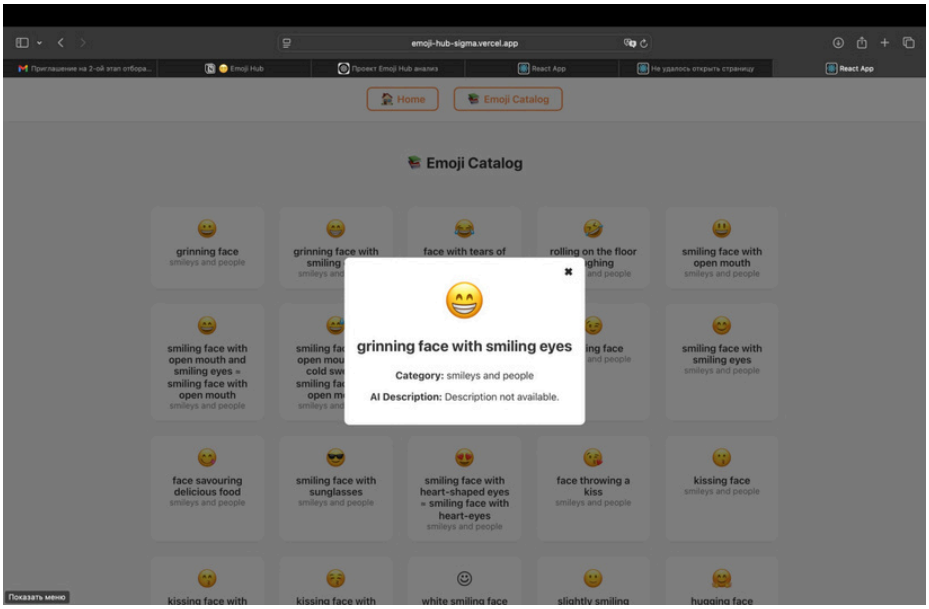
Emoji Hub

Emoji Hub — это веб-приложение для удобного просмотра и фильтрации коллекции эмодзи (emoji). Проект состоит из клиентской части (интерфейс на React) и серверной части (API на Spring Boot). Приложение отображает список ~1800 эмодзи, сгруппированных по категориям, и позволяет получить дополнительную информацию о каждом символе. Ключевая особенность — интеграция с моделью искусственного интеллекта *Gemini* от Google (через API), с помощью которой генерируется описание значения каждого эмодзи.

Описание проекта



Emoji Hub предоставляет каталог всех популярных эмодзи Unicode. Пользователь может просматривать эмодзи по категориям (например, «Smileys and People», «Animals and Nature» и др.) и искать нужный символ. Интерфейс отображает каждый эмодзи в виде карточки с названием (на англ. языке) и категорией. При клике на карточку открывается модальное окно с крупным изображением эмодзи, его названием, категорией и сгенерированным AI-описанием, объясняющим смысл и контекст использования данного эмодзи. Если описание от AI недоступно (например, при первом запуске или отсутствии подключения), в окне выводится сообщение о его недоступности. Благодаря такому подходу, даже малоизвестные символы становятся понятнее пользователю.

Основной функционал

- **Каталог эмодзи по категориям:** приложение загружает полную базу ~1800 эмодзи через внешний API и отображает их, группируя по категориям. Пользователь видит название каждого эмодзи и категорию, к которой он относится. Возможна фильтрация по категориям для удобства навигации.

- **Поиск и фильтрация:** реализован быстрый фильтр по названию эмодзи (и/или выбор категории) для нахождения нужного символа в обширном списке. Это упрощает поиск конкретного эмодзи по ключевым словам.
- **Детальная информация в модальном окне:** при выборе эмодзи открывается всплывающее модальное окно с подробностями. В нём отображается увеличенный символ, название, категория и **AI-описание** – краткое пояснение значения эмодзи, сгенерированное моделью Gemini. Это описание помогает понять эмоциональную окраску или типичный смысл использования данного значка.
- **Кеширование описаний:** чтобы снизить задержки и нагрузку на API AI-модели, сгенерированные описания сохраняются (кешируются). При повторном открытии того же эмодзи приложение использует закешированное описание, избегая повторного запроса к внешнему сервису.
- **Современный интерфейс:** фронтенд построен с помощью React. Интерфейс адаптивный и интуитивно понятный: навигационное меню позволяет переключаться между домашней страницей и каталогом эмодзи, карточки оформлены единообразно, а модальные окна обеспечивают фокус на выбранном элементе без перегрузки новой страницы.

Установка и запуск

Проект состоит из двух частей, поэтому для локального запуска понадобятся **Node.js** (для фронтенда) и **Java 17+** с Maven (для бэкенда, Maven Wrapper включён). Ниже приведены инструкции по запуску локально и развертыванию на платформах **Render** (сервер) и **Vercel** (клиент):

Локальный запуск

1. **Клонирование репозитория:**

   ```
   git clone https://github.com/Sultan31415/emoji-hub.git
   cd emoji-hub
   ```

2. **Конфигурация переменных окружения:** Для работы AI-описаний требуется API-ключ от Google Gemini. Получите ключ на Google AI Studio (необходим доступ к **Generative Language API**). Затем создайте переменную окружения `GEMINI_API_KEY` со значением вашего ключа:
3. В Linux/MacOS: `export GEMINI_API_KEY=<ваш_ключ>`
4. В Windows (PowerShell): `$env:GEMINI_API_KEY="<ваш_ключ>"`
   Вы также можете прописать ключ в файл `backend/src/main/resources/application.properties`, однако **не рекомендуется** хранить секреты в коде – для безопасности используется вариант с переменными окружения.
5. **Запуск backend-сервера:** Убедитесь, что установлена Java (рекомендуется 17 или 20). Выполните команду Maven Wrapper для сборки и запуска Spring Boot сервиса:

   ```
   cd backend
   ./mvnw spring-boot:run
   ```

   Это запустит сервер на порту 8080 (по умолчанию). Если запуск успешен, в консоли появится лог Spring Boot со строкой `Started BackendApplication`.
6. **Запуск frontend-приложения:** Откройте новый терминал и выполните:

```
cd frontend
npm install
npm start
```

Убедитесь, что установлена Node.js (рекомендуется версия 18+). Скрипт `npm start` запустит локальный dev-сервер React на порту 3000. Фронтенд автоматически проксирует API-запросы к локальному бэкенду (на http://localhost:8080), чтобы получать данные. После запуска откройте http://localhost:3000 в браузере. Вы должны увидеть веб-интерфейс Emoji Hub и загрузившийся список эмодзи.

7. **Проверка функционала:** Попробуйте кликнуть на любую карточку эмодзи. При первом клике может потребоваться 1-2 секунды для получения описания от AI (см. раздел об известных проблемах ниже). Повторные открытия должны быть быстрее благодаря кешированию. Если описания не появляются, убедитесь, что правильно настроен API-ключ и бэкенд запущен без ошибок.

Развертывание на Render и Vercel

Backend (Spring Boot на Render): Для деплоя серверной части используйте платформу Render, которая поддерживает Spring Boot приложения. Шаги развертывания: - Создайте новый Web Service на Render, привязав репозиторий GitHub с проектом **emoji-hub**. Укажите путь к директории `backend` и команду сборки `./mvnw clean install` (или `mvn clean install`) и команду запуска `java -jar target/backend-0.0.1-SNAPSHOT.jar` (имя JAR может отличаться по версии). Альтернативно, можно использовать Docker: в репозитории имеется Dockerfile для backend, его можно задействовать для контейнерного деплоя. - В настройках сервиса на Render добавьте переменную окружения **GEMINI_API_KEY** с вашим API-ключом (через меню *Environment -> Add Environment Variable*). Это необходимо, чтобы при запуске сервер получил доступ к ключу и мог вызывать Gemini API. - Настройте политику CORS: на сервере уже предусмотрена настройка, разрешающая запросы с адреса фронтенда. По умолчанию разрешены обращения с домена Vercel приложения и с `localhost:3000` для отладки. - После деплоя бекенд будет доступен по URL вида https://emoji-hub-xxxxx.onrender.com. Убедитесь, что этот URL корректно используется фронтендом (см. ниже).

Frontend (React на Vercel): Клиентскую часть удобно размещать на Vercel или аналогичной хостинговой платформе для React приложений. - Создайте новый проект на Vercel, выберите репозиторий **emoji-hub** и укажите папку проекта `frontend` для деплоя. Vercel автоматически обнаружит React и запустит сборку (`npm run build`). - В настройках Vercel проекта добавьте переменную окружения **REACT_APP_BACKEND_URL**, значение которой должно быть URL вашего backend API на Render. Например: `REACT_APP_BACKEND_URL = https://emoji-hub-xxxxx.onrender.com`. Это нужно, чтобы сборка фронтенда знала, куда слать запросы за данными. Если переменную не указать, по умолчанию фронтенд будет пытаться обратиться на адрес бэкенда, заданный в исходниках (в режиме разработки это localhost:8080). - После деплоя Vercel предоставит вам URL вида https://emoji-hub-yyyy.vercel.app. Откройте этот адрес – приложение должно загрузиться и взаимодействовать с API на Render. Проверьте работоспособность: эмодзи подгружаются, модальные окна отображают описания. Учтите, что при простое бэкенд на бесплатном тарифе Render «засыпает» – первый запрос после паузы может выполняться с задержкой (см. известные проблемы).

Этапы разработки и дизайн

Разработка Emoji Hub велась итеративно, с разделением на несколько ключевых этапов:

1. **Планирование и анализ:** Сначала были сформулированы цели проекта – создать справочник эмодзи с возможностью поиска и получения значения символов. На этом этапе выбрали стек технологий (React для UI, Spring Boot для API) и внешние ресурсы: нашли открытый API с базой эмодзи и решили использовать генеративный AI для описаний. Обсуждалось, какие данные отображать (название, категория, код символа и т.д.) и как интегрировать AI-описание, чтобы оно действительно повышало ценность приложения.
2. **Базовая реализация фронтенда и API эмодзи:** Далее был создан каркас React-приложения (структура компонентов, маршруты страниц *Home* и *Emoji Catalog*). Параллельно настроен бэкенд-проект Spring Boot. На этом этапе реализована загрузка данных всех эмодзи через внешний EmojiHub API. Сервис **EmojiService** на бэкенде делает HTTP-запрос к публичному API и возвращает список эмодзи (имя, категория, код, unicode и др.). Фронтенд через fetch/Axios обращается к своему бэкенду (ендпоинт, например, `/api/emojis`) и отображает полученные данные в виде сетки карточек. Была реализована простая фильтрация списка на стороне клиента (например, по категории или тексту) для удобства пользователя.
3. **Интеграция модальных окон и UX улучшения:** Затем добавлен компонент модального окна на React. При клике на элемент списка фронтенд запрашивает у бэкенда дополнительную информацию (описание) и открывает модальное окно. Этот подход улучшил UX: пользователь остаётся на той же странице, а подробности отображаются поверх основного контента. Также были доработаны стили с помощью CSS: карточки оформлены с иконкой эмодзи и тенью при наведении, модальное окно – центрировано с полупрозрачным фоном вокруг.
4. **Внедрение AI-генерации описаний:** Один из самых важных этапов — подключение *Gemini API* для генерации описаний. Реализован сервис **EmojiDescriptionService** на бэкенде, который получает название эмодзи и делает запрос к внешнему AI API (Google Generative Language API, модель Gemini). На первом шаге интеграции ключ API временно был прописан прямо в код для тестирования, и успешный вызов возвращал текст – краткое описание или значение эмодзи. После подтверждения работоспособности логика была улучшена: ключ вынесен в переменные окружения для безопасности, добавлена обработка ошибок (если внешний сервис не отвечает или квота исчерпана, сервис возвращает уведомление, что описание недоступно). Полученное от AI описание включается в ответ бэкенда и отображается в модальном окне на фронте.
5. **Кеширование и оптимизация:** Поскольку генеративный AI запрос может выполняться ~1-2 секунды и потенциально тарифицируется, было решено внедрить механизм кеширования. На уровне сервера сохранения результатов в памяти: например, в коллекции Map по ключу названия эмодзи хранится сгенерированный ранее текст. **EmojiDescriptionService** при запросе сначала проверяет кеш: если описание для данного эмодзи уже получалось ранее, сразу возвращается сохранённый результат, минуя внешний вызов. Эта оптимизация значительно ускорила повторные обращения и снизила нагрузку на API. Также, на фронтенде можно кешировать полученные описания в состоянии React, чтобы при повторном открытии модального окна не делать повторный запрос вовсе.
6. **Тестирование и отладка:** После реализации основных функций проводилось тестирование: проверка корректности загрузки всех эмодзи, работы фильтра, соответствия AI-описаний контексту эмодзи. Выявилась проблема с CORS при попытке обращения фронтенда (Vercel) к бэкенду (локально или на Render) — она была решена добавлением

соответствующих заголовков в Spring Boot (разрешены origins Vercel и localhost). Кроме того, при первоначальной интеграции AI-ключа GitHub обнаружил утечку ключа (функция Secret Scanning пометила его как публичный), поэтому ключ был оперативно удалён из репозитория и заменён на безопасное хранение через переменные окружения. Это улучшило безопасность проекта.

7. **Деплоймент:** Заключительный этап — развёртывание приложения. Фронтенд был собран и размещён на Vercel, что дало мгновенный доступ к клиенту из браузера. Бэкенд задеплоен на Render; потребовалась дополнительная настройка (переменные окружения, указание версии Java, сборка) для успешного запуска. После деплоя были проведены финальные испытания уже в боевом окружении: проверена связка фронт+бэк через интернет, скорость отклика первого запроса, поведение приложения при «пробуждении» спящего сервиса. Удостоверившись в работоспособности, проект был подготовлен к публикации.

Уникальные подходы и технологии в проекте

- **Интеграция с Gemini API (Generative AI):** В отличие от статичных справочников, Emoji Hub использует возможности современного ИИ для генерации описаний. Модель *Gemini* от Google (через OpenRouter/Google API) принимает название эмодзи и генерирует человекоподобное объяснение, что означает этот символ и в каком контексте он применяется. Этот подход позволяет получать более «живые» и понятные описания, чем сухие технические определения. Интеграция потребовала разработки специального контроллера и сервиса на Spring Boot для общения с внешним API, а также обработки ошибок и задержек. В результате, каждая карточка эмодзи обогащается динамическим содержанием от ИИ, что выделяет проект среди аналогов.
- **Использование модальных окон для деталей:** Вместо перехода на отдельную страницу для каждого эмодзи, была внедрена система модальных окон (всплывающих поверх контента). Это улучшает пользовательский опыт: просмотрев описание, можно сразу закрыть окно и продолжить с того же места в списке. Реализация выполнена с помощью React-хука состояния, который отображает компонент с деталями при выборе эмодзи. Модальные окна также помогают фокусировать внимание — фон затемняется, пока окно открыто.
- **Кеширование AI-описаний:** Уникальная инженерная находка в проекте — кеширование результатов работы AI. Поскольку генерация описания через внешний API – операция относительно долгая и платная, был сделан компромисс между свежестью данных и скоростью работы. Решение: сохранять сгенерированное описание для каждого эмодзи после первого запроса. В дальнейшем, при повторном запросе того же описания, возвращать сохранённый результат. Таким образом, если пользователь многократно открывает одни и те же популярные эмодзи, описание генерируется только один раз. Этот подход резко снижает количество обращений к AI-сервису и обеспечивает мгновенный отклик на уже просмотренные элементы.
- **Разделение на frontend и backend:** Проект реализован по принципу *full-stack*: React отвечает за интерфейс и логику на стороне клиента, а Spring Boot – за бизнес-логику и интеграцию с внешними API. Они общаются между собой через REST API (JSON). Такое разделение позволило разрабатывать и развёртывать части независимо (например, масштабировать сервер или обновлять интерфейс без перепаковки всего приложения). Также это обеспечивает безопасность: секретный ключ AI хранится только на сервере, недоступном напрямую из браузера.
- **Внешний API EmojiHub:** Вместо ручного сбора базы эмодзи, проект использует готовый открытый API (EmojiHub от разработчика cheatsnake) для получения актуального списка эмодзи и их категорий. Это пример повторного использования существующих данных:

эмодзи и так стандартизированы Unicode, поэтому целесообразно брать их из проверенного источника. Благодаря этому, Emoji Hub всегда оперирует полной и актуальной базой символов без необходимости вручную обновлять данные при выходе новых эмодзи.
- **UX-акценты:** Помимо технических решений, уделено внимание мелочам UX: например, при загрузке списка эмодзи может отображаться индикатор (спиннер), чтобы пользователь понимал, что идёт процесс. Текстовые сообщения (как то «Description not available») помогают сориентироваться, если что-то пошло не так. Дизайн выдержан в едином стиле, использованы нейтральные цвета фона, чтобы акцентировать внимание на ярких эмодзи. Эти детали делают использование приложения приятнее.

Принятые решения и компромиссы

В ходе разработки приходилось принимать решения, связанные с выбором технологий и реализаций, учитывая ограничения времени и ресурсов:

- **Выбор источника данных для эмодзи:** Рассматривалось несколько вариантов получения списка эмодзи – использовать готовый API, подключиться к какому-либо SDK или просто сохранить статичный JSON вручную. Был выбран **EmojiHub API** как готовое решение. Компромисс: зависимость от доступности внешнего сервиса, зато минимум трудозатрат и всегда актуальные данные. В случае недоступности API на продакшене можно реализовать резервный вариант – хранить закешированный список эмодзи локально и обновлять его периодически.
- **Выбор AI API для описаний:** Изначально рассматривались различные AI-модели для генерации текста. Возможные опции: **OpenAI GPT-3.5/GPT-4**, открытые модели через **HuggingFace API**, либо новые модели от Google. Был сделан выбор в пользу **Google Gemini (Palm API)**, доступ к которой получен через платформу OpenRouter/Google AI. Причины выбора: высокое качество генерации, бесплатная квота на тестирование и интерес попробовать новейшую технологию. От **HuggingFace** отказались из-за нескольких факторов: использование open-source модели потребовало бы либо развертывания собственной инфраструктуры (что сложно и ресурсозатратно), либо обращения к API HuggingFace (который по сути обёртка над теми же моделями, но с ограничениями по скорости и качеству). Модели OpenAI, хоть и очень мощные, часто требуют платной подписки и имеют строгие ограничения на ключи, а также широко распространены – для учебного проекта было интересно интегрировать альтернативное решение от Google.
- **Логика на фронте vs бэкенде:** Поначалу часть функционала планировалась прямо в React (например, обращение к API эмодзи напрямую из браузера или даже вызов AI через fetch). Такой подход проще, но небезопасен – пришлось бы экспонировать API-ключ AI в коде фронтенда. После анализа рисков было принято вынести всю интеграцию с внешними сервисами на **backend**. Компромисс – усложнение архитектуры (нужно писать и поддерживать серверный код), задержка на промежуточное звено, но взамен получаем защиту ключей, централизованную логику (можно валидировать/обогащать данные на сервере) и контроль доступа. Это решение себя оправдало: например, настроив кэш на сервере, мы сразу улучшили производительность для всех клиентов.
- **CORS и единый источник данных:** Ещё один момент – решено не смешивать фронтенд и бэкенд в одном приложении (монолите), а развернуть раздельно. Фронтенд обслуживается как статика (Vercel), а API – отдельно (Render). Это упрощает деплой, но требует настройки CORS (разрешить домен фронта на бэке). Мы добавили необходимую конфигурацию в Spring Boot. Альтернативой было бы собрать фронтенд в static-ресурсы и размещать его на том же домене, но тогда пришлось бы отказаться от удобств Vercel. Мы предпочли гибкость раздельного деплоя, пожертвовав немного временем на настройку CORS.

- **Использование бесплатных тарифов:** При выборе хостингов и сервисов было решено использовать бесплатные тарифы (Free Tier) для экономии средств, что типично для учебного проекта. Например, Render (Free Web Service) и Vercel (Hobby). Компромисс здесь – ограничения по ресурсам: спящий режим сервера, более медленная работа после простоя, возможные ограничения по количеству запросов. Мы сочли это приемлемым для демонстрации проекта. В будущем, для production-развёртывания, можно перейти на более мощные планы или другой хостинг, если потребуется улучшить время отклика.

Обоснование выбора технологического стека

React (CRA) – выбран в качестве фронтенд-библиотеки благодаря своей популярности и богатой экосистеме. Он обеспечивает модульность через компоненты, что упростило создание карточек эмодзи и модальных окон как повторно используемых компонентов. Также React отлично подходит для SPA (Single Page Application) с динамическим обновлением данных без полной перезагрузки страницы, что было важно для реализации модальных окон и фильтрации. Большое сообщество и наличие готовых решений (например, библиотеки для модальных окон, иконки, роутинг) ускорили разработку интерфейса.

Spring Boot – современный фреймворк для разработки на Java, выбран для построения RESTful API бэкенда. Его преимущества: быстрое создание standalone-сервиса (всё запускается из jar, встроенный сервер Tomcat), удобная работа с внешними REST API (через RestTemplate/WebClient мы получаем данные эмодзи и обращаемся к AI), встроенные средства конфигурации (application.properties упростил вынесение URL и ключей API). Кроме того, Spring Boot хорошо знаком разработчику, что позволило быстро имплементировать необходимый функционал (контроллеры, сервисы) и воспользоваться готовыми решениями для безопасности и кеширования (аннотации @Value для чтения переменных, и т.д.).

OpenRouter / Google Gemini API – выбор этой технологии обоснован стремлением внедрить генеративный ИИ новейшего поколения. OpenRouter предоставляет унифицированный доступ к разным LLM (Large Language Model), и через него был получен доступ к модели **Gemini 2** от Google. Gemini – продвинутая модель, способная генерировать осмысленные тексты на основе входных данных. Использование её через API позволило обогатить функционал приложения уникальной возможностью объяснять эмодзи «словами машины». Альтернативой была OpenAI, но, как отмечено выше, выбор пал на решение от Google как экспериментальное и перспективное. Это показывает владение современными технологиями AI в проекте.

EmojiHub API (cheatsnake) – открытый API для эмодзи был задействован для экономии времени на сбор данных. Он предоставляет готовый эндпоинт со всеми необходимыми полями (название, категория, unicode и др.). Технически, можно было бы использовать и официальные данные Unicode или библиотеку, но готовый REST API оказался проще. Этот выбор демонстрирует умение находить и применять сторонние **Public API** для решения задач, вместо «изобретения велосипеда». Благодаря этому, основной фокус можно было сместить на действительно интересные части проекта (например, AI-описания), а не на рутинное наполнение справочника.

Vercel – платформа выбрана для деплоя фронтенда из-за её удобства для React-приложений. Vercel автоматизирует процесс CI/CD: достаточно связать репозиторий, и при каждом пуше изменений фронтенд пересобирается и выкладывается. Кроме того, Vercel обеспечивает глобальную CDN, благодаря чему статические ресурсы (HTML, JS, CSS эмодзи иконки) раздаются быстро пользователям по всему миру. Бесплатного плана достаточно для проекта, а время первого открытия страницы минимально. Также важна простота – не нужно вручную настраивать веб-сервер для раздачи собранного React, Vercel взял это на себя.

Render – сервис хостинга бэкенда. После ухода Heroku с бесплатного рынка Render стал популярным выбором для развёртывания проектов на Spring Boot. Он позволяет запустить контейнер с Java-приложением без особых сложностей: интеграция с GitHub, автоматическая сборка Maven, настройка переменных окружения. В рамках проекта Render оказался подходящим для демонстрационного API: поддержка Java 17, достаточно памяти для работы ~1800 объектов в памяти и выполнение HTTP-запросов к внешним сервисам. Хотя первое обращение к бесплатному инстансу может быть медленным (разогрев), в целом Render обеспечил надёжную работу серверной части. Выбор этого стека (React + Spring Boot + Vercel + Render + AI API) показал себя удачным: каждая технология используется по назначению и взаимодействует через чётко определённые интерфейсы (HTTP API), что упростило отладку и сопровождение.

Известные проблемы и ограничения

- **Медленный первый запуск (Cold Start):** При деплое на бесплатных платформах наблюдается заметная задержка при первом обращении. В частности, бэкенд на Render освобождает ресурсы при бездействии (через ~15 минут простоя). Поэтому первый запрос к API (например, загрузка списка эмодзи) после «пробуждения» сервера может выполняться 5–10 секунд. В этот момент фронтенд может показывать индикатор загрузки. Эта проблема характерна для бесплатного хостинга; на локальном сервере или платном тарифе такой задержки нет.
- **Задержка при получении AI-описания:** Генерация описания эмодзи через внешнюю модель занимает время (обычно ~1–2 секунды). Пользователь может заметить небольшую паузу между открытием модального окна и появлением текста описания. Мы минимизировали это за счёт кеширования (повторные запросы быстрые), но для нового эмодзи задержка неизбежна. Возможно, в будущем стоит добавить явный индикатор загрузки внутри модального окна, чтобы пользователь понимал, что идёт запрос к AI.
- **Ограничения бесплатного API:** Используемый AI-сервис (Google Gemini API) имеет квоты на количество запросов в минуту/день. В редких случаях при активном использовании приложения описания могут временно не генерироваться, если превышен лимит. В таких ситуациях пользователь увидит сообщение *«Description not available»*. Аналогично, внешнее EmojiHub API теоретически может быть недоступно; тогда список эмодзи не загрузится. Эти риски частично смягчены кешированием и тем, что при повторных запусках бэкенд может сохранять результат предыдущих загрузок.
- **Отсутствие поддержки старых браузеров:** Приложение тестировалось в современных браузерах (Chrome, Firefox, Safari, Edge). Используемые технологии (ES6+, fetch API, Flexbox/Grid CSS) могут не работать в устаревших версиях Internet Explorer. Для большинства пользователей это не проблема, но в README стоит упомянуть: рекомендуются современные браузеры для корректной работы Emoji Hub.
- **Возможные неточности описаний:** Хотя модель Gemini весьма продвинута, описания эмодзи генерируются автоматически и могут незначительно отличаться от общепринятых определений. Например, AI может описать эмоцию своими словами, которые не буквально совпадают с Unicode-описанием. В целом это не критично и даже придаёт «человечности» пояснениям, но стоит понимать, что источник этих данных – вероятностная модель. Мы не внедряли ручную модерацию или правку генерируемого текста, уповая на качество модели.
- **Дальнейшие улучшения:** В текущей версии отсутствует ряд возможных функций, которые могли бы повысить ценность приложения (например, копирование эмодзи в буфер обмена по клику, более сложная сортировка/поиск, локализация интерфейса на другие языки). Эти идеи находятся за рамками текущей реализации, но могут быть реализованы в будущем. Основная цель – продемонстрировать концепцию и базовый функционал – достигнута, а улучшения могут быть внесены по мере необходимости.

Emoji Hub – учебно-демонстрационный проект, сочетающий справочник эмодзи с современными технологиями веб-разработки и искусственного интеллекта. Он служит отличным примером полного цикла создания Full-Stack приложения: от идеи и дизайна до деплоя и поддержки. Мы постарались подробно описать каждый аспект проекта в данном README, чтобы облегчить понимание устройства приложения и выделить ключевые решения, принятые в ходе разработки. Спасибо за внимание! Если у вас есть вопросы или вы хотите внести свой вклад в проект, обращайтесь через систему Issues или Pull Request в репозитории GitHub. Сделаем Emoji Hub лучше вместе!
