Автоматическая классификация «болей писем»

1. Общее описание решения

Классификация делается как сопоставление пользовательского текста с заранее заданным набором тегов (болей): задача «смысл → один тег из списка». Система не придумывает новые категории, а только выбирает один наиболее подходящий тег из существующих в ДБ.

Цепочка: ввод текста → анализ смысла (LLM) → выбор одного тега из списка → проверка по словарю → возврат пользователю.

Подход к идеальному подбору тега опирается на пять элементов: строгий формат ответа (JSON, только tag_id), проверка по словарю (если tag_id не из списка -ответ считается ошибкой), при необходимости few-shot (несколько примеров «текст → правильный тег») и политика при низкой уверенности (уточнение у пользователя или возврат наиболее близкого тега с пометкой).

Основные компоненты и их роли

Компонент - Роль

Клиент - Показывает список болей по топикам, принимает свободный текст от пользователя, отправляет его на backend, получает и отображает выбранный тег.

API классификации - Принимает текст, вызывает логику анализа (обращение к LLM), проверяет ответ по словарю тегов, возвращает один тег (и при необходимости топик).

Модуль анализа текста - На основе LLM сопоставляет пользовательский текст со списком тегов: получает от модели один tag_id в строгом формате (JSON, без), проверяет по словарю, при необходимости использует few-shot и политику при низкой уверенности.

Хранилище топиков и тегов - Таблицы в Supabase с топиками и тегами, оттуда берётся словарь для проверки ответа LLM и при необходимости -описания тегов для подсказок или few-shot примеров.

2. Архитектура

Общая схема архитектуры

Архитектура строится вокруг Supabase как единой backend-платформы. Приложение обращается к одному API-методу (Supabase Edge Function):

1. Принимает свободный текст пользователя.
2. Загружает из ДБ список топиков и их тегов для подсказки LLM и проверки ответа.
3. Отправляет текст и список тегов во внешний LLM API с инструкцией «вернуть ровно один tag_id из списка», получает ответ в строгом формате.
4. Проверяет, что tag_id входит в словарь тегов, если не из списка -ответ считается ошибкой.
5. Возвращает один выбранный тег клиенту.

Все данные и логика классификации сосредоточены внутри Supabase. Вызов LLM выполняется из Edge Function, словарь тегов хранится в ДБ и не дублируется на клиенте для целей классификации.

Разделение на клиент и backend

На клиенте:

- UI, ввод текста, вызов API классификации, отображение результата
- Отдельные запросы для получения списка топиков и тегов

На backend:

- Управление и хранение топиков, тегов
- Классификация в Edge Function: приём текста → вызов LLM API → проверка ответа по словарю → возврат одного тега
- Интеграция с внешним LLM API

Роль Supabase

Предоставляет аутентификацию, доступ к ДБ и Edge Functions.

Является единственной точкой входа для мобильного клиента.

Хранит доменные данные: топики, теги, при необходимости -логи запросов классификации.

Вызов LLM из Edge Function

Edge Function формирует запрос к внешнему LLM API: пользовательский текст, список допустимых тегов (id и название/описание), инструкция «вернуть ровно один tag_id из списка», строгий формат ответа (JSON).

Ответ модели проверяется по словарю тегов из ДБ, если tag_id не из списка, ответ считается ошибочным

Таймауты и обработка недоступности LLM API обеспечиваются на стороне Edge Function.

3. Работа с данными

Структура данных

Топики:

- id -идентификатор топика
- name -название топика
- description -описание топика
- sort_order -порядок вывода топиков для пользователя
- created_at, updated_at (timestamptz) - тех. метаданные для отслеживания создания и изменения записей

Теги:

- id -идентификатор тега
- topic_id -к какому топику относится этот тег
- name -название тега
- description -описание тега
- sort_order -для сортировки внутри топика
- created_at, updated_at (timestamptz) - тех. метаданные для отслеживания изменений в справочнике

Использование данных в процессе классификации

Топики

Топики в момент выбора тега не участвуют: классификация идёт по тегам, а не по топикам. Топики нужны до и после.

До запроса пользователь видит весь список болей, структурированный по топикам.

После выбора тега система возвращает один наиболее подходящий тег, топик можно подставить по topic_id для обогащения ответа.

Сами топики используются для отображения справочника и для обогащения ответа после того, как тег уже выбран.

Теги

Классификация строится только на тегах: словарь тегов загружается из ДБ и передаётся в LLM.

При запросе классификации Edge Function загружает из таблицы тегов список допустимых тегов (id, name, при необходимости description) для формирования промпта и проверки ответа.

Пользовательский текст приходит от клиента в Edge Function и вместе со списком тегов отправляется во внешний LLM API. Ответ «tag_id» проверяется по загруженному списку id, если значения нет в списке - ответ считается ошибочным.

При использовании few-shot примеры «текст → тег» могут храниться отдельно или формироваться из тех же полей тегов (name, description).

4. Подход к анализу текста

Метод анализа текста

Предлагается использование языковой модели (LLM) для задачи «смысл → один тег из списка»: модель получает свободный текст и перечень допустимых тегов (id, название, при необходимости описание) и возвращает ровно один идентификатор тега. Ответ оформляется в строгом формате: JSON, tag_id.

Так же, ответ модели проверяется по словарю тегов из ДБ: если возвращённый tag_id не входит в список допустимых, ответ считается ошибочным. При необходимости в промпт добавляют few-shot и задают политику при низкой уверенности: если модель не уверена, запрашивать уточнение у пользователя или отдавать наиболее близкий тег с пометкой. Регулярное тестирование на маленьком наборе сложных кейсов позволяет контролировать качество.

Как происходит сопоставление пользовательского текста с существующими тегами

В промпт передаётся пользовательский текст и список тегов (id, name, при необходимости description), загруженный из ДБ. В инструкции явно задаётся: выбрать один наиболее подходящий тег из списка и вернуть только его tag_id.

Модель возвращает структурированный ответ, парсинг происходит на стороне Edge Function. Полученный tag_id проверяется на вхождение в загруженный список id, если не из списка, ответ считается ошибкой. Пользователю возвращается один выбранный тег.

Почему выбран именно этот подход (сравнение)

Keywords - задание предполагает произвольное описание проблемы свободным текстом, поиск по ключевым словам недостаточен для надёжного сопоставления по смыслу.

TF-IDF - сравнение идёт по словам, а не по смыслу, при другой формулировке той же проблемы векторы могут сильно отличаться, качество классификации падает. Для свободного текста нужен учёт контекста и смысла.

Эмбеддинги + поиск ближайшего соседа - дешевле и стабильнее на больших объёмах, однако задача «смысл → один тег из списка» при близких по смыслу тегах лучше решается LLM с явным перечислением категорий. Строгий формат ответа и обязательная проверка по словарю гарантируют, что система всегда возвращает ровно один тег из существующего набора и не выдумывает новые категории.

Что ближе всего к идеальному подбору тега

1. LLM + строгий формат ответа - JSON, tag_id.
2. LLM + проверка по словарю - если tag_id не из списка, ответ считается ошибкой.
3. LLM + few-shot - несколько примеров «текст → правильный тег».
4. LLM + политика при низкой уверенности - если модель не уверена, запрашивать уточнение у пользователя или отдавать наиболее близкий тег с пометкой.
5. Маленький набор сложных кейсов - регулярное тестирование на текстах, где теги похожи.

Рассматриваемые LLM-модели

Ниже перечислены модели, которые целесообразно рассмотреть для задачи «смысл → один тег из списка».

1. **OpenAI GPT-4.1**

   Плюсы: очень хорошее понимание смысла, стабильный structured output (JSON), высокая точность в выборе «один тег из списка».

Минусы: дороже, чем более лёгкие модели.

Искользование: если нужен максимум качества, сложные тексты, близкие по смыслу теги.

2. **Claude 3.5 Sonnet**

   Плюсы: аккуратно следует инструкции «строго один из списка», устойчивый вывод, хорошая семантика.

   Минусы: цена.

   Использование: если важно строгое соблюдение формата и стабильность.

3. **GPT-4o mini**

   Плюсы: дешёвый и быстрый, легко интегрируется.

   Минусы: при похожих тегах ошибается чаще.

   Использование: если много запросов и нужно снизить цену.

4. **Gemini 1.5 Pro**

   Плюсы: большой контекст, хорошая работа с длинными списками тегов.

   Минусы: качество языка иногда уступает топ-моделям OpenAI/Anthropic.

   Использованиеь: если список тегов очень большой или если стек завязан на Google.

5. **Mistral Large**

   Плюсы: дешевле GPT.

   Минусы: по качеству рассуждений и точности обычно уступает другим LLM.

   Использование: если нужен компромисс цена/качество, без максимальных требований.

Выбор конкретной модели зависит от приоритетов: максимум качества и сложные кейсы, строгий формат и стабильность, снижение стоимости при большом числе запросов, длинна списока тегов, компромисс цена/качество.

5. Технологический стек

Backend

Supabase используется как основная база данных, в ней хранятся топики и теги. Логика классификации реализуется в Supabase Edge Function, приём текста от

клиента → загрузка списка тегов из ДБ → вызов внешнего LLM API → проверка ответа по словарю → возврат одного тега клиенту.

Анализ текста

Для анализа текста используется внешний LLM API. Конкретная модель выбирается из рассмотренных выше в зависимости от приоритетов по качеству, стоимости и латентности. Edge Function формирует промпт, отправляет запрос в API выбранной модели, получает структурированный ответ и проверяет его по словарю тегов из ДБ. Такой подход даёт смысловое сопоставление текста с тегами и позволяет опираться на строгий формат ответа и проверку по словарю.

Интеграция с Supabase

Клиент общается только с Supabase, через него он вызывает Edge Function для классификации и отдельные запросы к таблицам топиков и тегов для отображения справочника. Вся логика (загрузка тегов из ДБ, вызов LLM API, проверка tag_id по словарю, возврат тега) выполняется на стороне Supabase.

Рассмотренные и отброшенные альтернативы

Вместо Supabase можно было использовать свой backend и отдельную ДБ, однако задание предполагает Supabase как backend-платформу, единая точка входа и хранение топиков/тегов в Supabase упрощают интеграцию с приложением.

Вместо LLM для анализа текста рассматривались ключевые слова, TF-IDF и эмбеддинги с поиском ближайшего соседа. Эмбеддинги дешевле и стабильнее, но при необходимости максимального качества и тонкого различения похожих тегов LLM предпочтительнее. Выбор конкретной модели определяется бюджетом и требованиями к точности.

6. Тестирование

Unit-тесты

Проверка отдельных функций на backend: правильная сборка промпта (пользовательский текст, список тегов, инструкция «вернуть ровно один tag_id»), парсинг структурированного ответа (JSON, tag_id), проверка tag_id по словарю допустимых id. Внешний LLM API подменяется заглушкой с фиксированным ответом (например, один и тот же tag_id), чтобы проверять логику Edge Function без вызова реального API.

Интеграционные тесты

Запускается код вместе с реальной ДБ Supabase (topics, tags). Проверяется полный путь: загрузили теги из ДБ → вызвали Edge Function (с заглушкой или тестовым вызовом LLM API) → получили ожидаемый тег. При необходимости проверяется, что при изменении списка тегов в ДБ классификация использует обновлённый словарь.

E2E-тесты

Имитация действий пользователя: ввод текста в приложении → вызов API классификации → отображение тега. Проверяется, что интерфейс показывает тот тег, который вернул backend, и корректно обрабатывает ошибки.

Обязательные проверки

Разные формулировки одной и той же проблемы (синонимы, другой порядок слов) в идеале приводят к одному и тому же тегу, для контроля качества поддерживается набор сложных кейсов, который регулярно прогоняется через систему.

Пустой или очень короткий текст не ломает систему: API возвращает понятную ошибку или обработанный ответ, а не падает.

При изменении описания или состава тегов в ДБ классификация при следующем запросе использует обновлённый список тегов.

Критичные части

Логика проверки ответа LLM: парсинг JSON с tag_id и валидация по загруженному списку id, обработка случая, когда модель вернула tag_id не из списка.

Обработка ошибок LLM API: таймауты, недоступность сервиса, невалидный или пустой ответ Edge Function должен возвращать клиенту понятный результат.

7. Ограничения и риски

Ограничения подхода

Система всегда выбирает только из существующих тегов и не умеет придумывать новые категории. Если в словаре нет по-настоящему подходящего тега, результат будет условно «наиболее близкий», но не идеальный.

Качество классификации зависит от промпта и от описаний тегов в ДБ. Если названия или описания тегов короткие, нечёткие или сильно похожи друг на друга,

модель может чаще ошибаться. Для хорошей работы нужно аккуратно вести справочник тегов.

Если внешний LLM API недоступен или сильно тормозит, классификация может работать медленно или не работать вовсе. Необходимы таймауты, ретраи и явная обработка сбоев.

Качество и стабильность зависят от выбранной модели и от того, насколько хорошо она поддерживает язык пользовательских текстов. Более дешёвые модели при похожих тегах ошибаются чаще.

Стоимость и латентность запроса выше, чем при подходе на эмбеддингах: каждый запрос классификации - это вызов LLM API. Модель может вернуть tag_id, не входящий в список, проверка по словарю обязательна и такой ответ считается ошибочным.

Основные риски

Плохое качество распознавания и классификации свободного текста (особенно при похожих тегах).

Решение: поддерживать набор типичных и сложных пользовательских фраз, для каждой вручную задать «правильный» тег, регулярно прогонять набор через систему, при необходимости добавлять few-shot примеры и политику при низкой уверенности, при недостаточном качестве рассмотреть более сильную модель.

Медленные ответы или недоступность LLM API.

Решение: таймауты и ретраи на вызов LLM API, при падении или таймауте возвращать клиенту понятную ошибку или fallback, при необходимости кэшировать результат для повторяющихся текстов.

Перегрузка или рост затрат на внешний LLM API.

Решение: ограничить длину пользовательского текста, логировать количество запросов, при необходимости ввести лимиты по пользователю или по приложению, выбор более дешёвой при приемлемом качестве.

8. Вопросы и допущения

Допущения

Пользователь вводит текст на том же языке, на котором описаны теги, либо выбранная LLM поддерживает нужные языки. Справочник тегов и модель предполагаются согласованными по языку.

Словарь тегов задан заранее и хранится в Supabase, новые теги не создаются пользователем в момент классификации.

Внешний LLM API доступен из Supabase Edge Function.

Объём текста от пользователя ограничен. Слишком длинные тексты можно обрезать или суммаризировать до одного запроса к LLM.

Supabase остаётся единой backend-платформой: аутентификация, ДБ и Edge Functions. Выбранная модель поддерживает structured output или его эмуляцию через промпт.

Вопросы для уточнения перед реализацией

Какую именно LLM использовать в проде. От этого зависят стоимость, латентность и качество. Возможен старт с более дешёвой модели с последующим переходом на более точную при необходимости.

Как поступать, если модель вернула tag_id не из списка. Считать ли ответ ошибочным и вернуть ошибку клиенту, повторить запрос, вернуть наиболее близкий тег с пометкой «низкая уверенность».

Как поступать при низкой уверенности модели. Возвращать наиболее близкий тег с пометкой, запрашивать у пользователя уточнение или ручной выбор тега.

Нужно ли логировать каждый запрос классификации для аналитики и отладки, и где хранить эти логи.