SPEC-1-Автоматическая генерация учебных работ из Git-проекта

Background

Учебные заведения требуют формальные курсовые и дипломные работы по результатам практических проектов. Разработчики часто имеют реальный код в публичных Git-репозиториях, но тратят значительное время на подготовку академического текста: обоснование, описание архитектуры, функционала и выводов. Существующие генеративные инструменты склонны «галлюцинировать» и производить обобщённые тексты, не привязанные к фактам конкретного проекта.

Данный сервис решает проблему за счёт строгого разделения ролей: логика сервиса выполняет детерминированный анализ проекта и формирует структурированный набор фактов (facts.json), а LLM используется строго как генератор текста на основе предоставленных фактов, структуры документа и кратких резюме уже сгенерированных разделов. Модель не получает исходный код и не хранит состояние (stateless), что снижает расходы токенов и повышает воспроизводимость. Сервис также автоматизирует получение иллюстраций (скриншоты для web-проектов через headless-браузер; для других типов программ — ручная загрузка), а на выходе формирует DOCX с корректной нумерацией разделов, рисунков и ссылок и поддержкой частичной перегенерации.

Ключевые ограничения и предпосылки: - ИИ — только генератор академического текста; все факты должны подтверждаться анализом репозитория. - Минимизация токенов: в LLM передаются только структура, facts.json и краткие summaries разделов. - Детализация и воспроизводимость: единообразные пайплайны анализа, стабильные промпты, фиксированные версии инструментов. - Расширяемость: возможность добавлять новые языки/стеки анализаторами-плагинами.

Requirements

Assumptions - Язык итоговых документов: **ru-RU**. - Академический стандарт оформления будет добавлен позже (пока нейтральная вёрстка DOCX).

MoSCoW

Must Have

- **Ввод**: URL публичного Git-репозитория + параметры (тип работы, целевой объём страниц A4, язык ru-RU, опционально дисциплина). *Acceptance*: валидация URL, проверка доступности clone без auth.
- **Анализ репозитория**: clone → очистка по `.gitignore` и правилам «шаблонного мусора» → определение языков, фреймворков/стека, архитектурного типа, модулей и features. *Acceptance*: детерминированный пайплайн, повторный запуск на том же коммите даёт идентичный `facts.json`.

- **facts.json**: только подтверждённые факты с ссылками-доказательствами (файл/путь, строки, хеш коммита). *Acceptance*: JSON-схема v1 с валидацией; каждая сущность содержит `evidence[]`.
- **Генерация поэтапно**: разделы вызываются отдельными запросами к LLM с передачей глобального контекста, структуры и кратких summaries предыдущих разделов. *Acceptance*: API имеет шаги `outline → theory → practice → conclusion` и хранит state вне LLM.
- **Экономия токенов**: LLM не получает исходный код; передаются только `facts.json`, структура и summaries. *Acceptance*: метрики токенов на запрос/раздел логируются.
- **Иллюстрации**: web-проекты — автоскриншоты headless-браузером; иные — ручная загрузка. Текст ссылается на изображения независимо от наличия. *Acceptance*: плейсхолдеры и корректные подписи/нумерация.
- **Выход**: DOCX с корректной нумерацией разделов/рисунков/ссылок. *Acceptance*: автооглавление, перекрёстные ссылки, стили заголовков.
- **Перегенерация разделов**: возможность пересоздать один раздел без полной пересборки. *Acceptance*: id разделов, неизменность прочих частей.
- **Детерминизм**: фиксированные версии анализаторов и промптов; возможность зафиксировать seed/температуру. *Acceptance*: контрольные прогонки дают одинаковый результат.
- **Расширяемость**: плагинная система анализаторов под новые языки/типы проектов. *Acceptance*: контракт плагина и каталог discovery.

Should Have

- **Поддержка популярных хостингов**: GitHub, GitLab, Bitbucket. *Acceptance*: e2e-тесты для каждого.
- **Шаблоны структур** по типам работ (курсовая/диплом) и дисциплинам. *Acceptance*: каталог шаблонов + правила выбора.
- **Управление объёмом**: генерация с таргетом страниц (±10%). *Acceptance*: подсчёт по стилям DOCX (а не по символам).
- **Русская типографика**: неразрывные пробелы, кавычки-ёлочки и т. п. *Acceptance*: пост-процессор форматирования.
- **Учётные записи пользователей**: регистрация/логин, роли (user, staff), API-токены. *Acceptance*: CRUD профиля, выдача токена, ограничение доступа к чужим проектам.
- **Модуль оплаты**: создание заказа/подписки, обработка webhooks, страница статуса платежа. *Acceptance*: успешная оплата разблокирует генерацию/скачивание, корректные статусы транзакций.

Could Have

- **Экспорт в PDF/HTML** в дополнение к DOCX.
- **Проверка оригинальности**: отчёт о заимствованиях на уровне текста (исключая код).
- **Список литературы**: автогенерация и форматирование ссылок.
- **Локальная песочница рендеринга**: предпросмотр разделов в браузере.

Won’t Have (пока)

- **Автоматическая генерация исходного кода**.
- **Приватные репозитории/аутентификация**.

- **Полноценное распознавание UI вне web-скриншотов**.

(Далее будет раздел Method.)

Method

Архитектура (обзор)

- **Core API (Python 3.11+)** — REST/JSON (Django 5 + Django REST Framework).
- **Repo Analyzer (Celery worker)** — клон репозитория, очистка по `.gitignore`, извлечение фактов → `facts.json`.
- **LLM Orchestrator (через OpenAI API)** — генерация разделов по шагам на основе `facts.json`/структуры/summaries.
- **Renderer** — сборка DOCX, нумерации и TOC.
- **Screenshot Service** — Playwright для web-скриншотов.
- **Auth & Billing** — Django auth, токены для SPA; биллинг через платёжный провайдер, вебхуки.
- **Хранилища** — PostgreSQL (state/биллинг), Redis (очереди), MinIO (артефакты).

```
### Технологический стек (MVP)

#### Выбор backend-стека: **Django + DRF** vs **FastAPI**
| Критерий | Django + DRF | FastAPI |
|---|---|---|
| Админ-интерфейс/CRUD инструментарий | Есть из коробки (Django Admin) | Нет,
нужно писать отдельно |
| Организация проекта и экосистема | Зрелая, стандартизованная |
Минималистичная, больше решений «собрать самому» |
| Async/IO (git/сетевые вызовы/OpenAI) | Поддержка async-view в Django 5/DRF,
часть экосистемы ещё sync | Нативный async повсеместно |
| Celery/Redis интеграция | Отличная совместимость | Отличная |
| Генерация OpenAPI/схем | drf-spectacular/drf-yasg | Авто-OpenAPI из аннотаций
|
**Решение:** берём **Django 5 + Django REST Framework**. Нам критичны
стандартные админ-инструменты, ролевые модели и единообразный стек; IO-нагрузку
вынесем в Celery. Async-обработчики подключим точечно при необходимости.

#### Подсистемы и технологии
- **Backend API:** Django 5, **Django REST Framework** (DRF), **drf-
spectacular** для OpenAPI 3.
- **БД:** **PostgreSQL 15** (Django ORM, миграции `manage.py migrate`).
- **Кэш/очереди:** **Redis 7**; **Celery** для фоновых задач (анализ репо,
скриншоты, сборка DOCX, вызовы OpenAI API). Опционально `django-celery-beat` для
расписаний.
- **Артефакты:** MinIO (S3-совместимое) — хранение `facts.json`, скриншотов,
итоговых DOCX (альтернатива: локальный том в MVP).
- **Headless-браузер:** **Playwright (Chromium)** для web-скриншотов.
```

```
- **Документы:** `python-docx` + шаблон стилей; оглавление и подписи.
- **Интеграция с OpenAI:** только **OpenAI API** (без локального LLM), ключи в переменных окружения.
- **Frontend:** **React + TypeScript + TailwindCSS** (SPA) поверх DRF API.
- **Контейнеризация:** **Docker/Compose** сейчас; подготовка k8s-манифестов на будущее (Deployment, Service, Ingress, HPA). Пометка: «Kubernetes — vNext».

### Контракты данных

#### 0) Аутентификация и биллинг (новое)
**Модели (упрощённо):**
```sql
CREATE TABLE account (
  id UUID PRIMARY KEY,
  user_id UUID UNIQUE,            -- Django auth_user
  role TEXT CHECK (role IN ('user','staff')) DEFAULT 'user'
);

CREATE TABLE subscription (
  id UUID PRIMARY KEY,
  account_id UUID REFERENCES account(id),
  plan TEXT,                      -- e.g. free, pro
  status TEXT CHECK (status IN ('inactive','active','past_due','canceled')),
  valid_until TIMESTAMPTZ
);

CREATE TABLE payment (
  id UUID PRIMARY KEY,
  account_id UUID REFERENCES account(id),
  provider TEXT,                  -- stripe|yookassa|cloudpayments
  provider_payment_id TEXT,
  amount_cents INT,
  currency CHAR(3) DEFAULT 'RUB',
  status TEXT CHECK (status IN ('created','paid','failed','refunded')),
  raw JSONB,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

Поток: `POST /billing/checkout` → редирект/линк на провайдера → webhook `/billing/webhook` → обновление `payment/subscription` → разблокировка функционала.

1) Входные параметры

```
{
  "repo_url": "https://github.com/org/repo",
  "work_type": "course|diploma",
```
```

```
  "target_pages": 40,
  "language": "ru-RU",
  "discipline": "optional string"
}
```

2) facts.json (v1)

```
{
  "schema": "facts.v1",
  "repo": {
    "url": "...",
    "commit": "<sha>",
    "detected_at": "2025-12-31T12:00:00Z"
  },
  "languages": [
    {"name": "Python", "ratio": 0.85, "evidence": [{"path": "*.py"}]}
  ],
  "frameworks": [
    {"name": "FastAPI", "type": "web", "evidence": [{"path": "pyproject.toml",
"lines": ["fastapi==..."]}]}
  ],
  "architecture": {
    "type": "web|desktop|api|cli|client-server|library",
    "layers": ["api", "domain", "infra"],
    "evidence": [{"path": "docker-compose.yml", "lines": ["uvicorn"]}]
  },
  "modules": [
    {"name": "users", "role": "auth", "evidence": [{"path": "app/users/"}]}
  ],
  "features": [
    {"id": "auth_login", "summary": "Авторизация", "evidence": [{"path": "app/
users/routes.py", "lines": [10, 80]}]}
  ],
  "runtime": {
    "dependencies": [
      {"name": "fastapi", "version": "*", "evidence": [{"path":
"pyproject.toml"}]}
    ],
    "build_files": ["Dockerfile", "docker-compose.yml"],
    "entrypoints": ["app/main.py"]
  }
}
```

3) Внутреннее состояние (PostgreSQL)

```
CREATE TABLE project (
  id UUID PRIMARY KEY,
  repo_url TEXT NOT NULL,
  default_branch TEXT,
  commit_sha TEXT,
  work_type TEXT CHECK (work_type IN ('course','diploma')),
  target_pages INT,
  language TEXT DEFAULT 'ru-RU',
  discipline TEXT,
  created_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE section (
  id UUID PRIMARY KEY,
  project_id UUID REFERENCES project(id),
  code TEXT CHECK (code IN ('outline','theory','practice','conclusion')),
  status TEXT CHECK (status IN ('pending','ready','error')),
  summary TEXT,
  content TEXT,
  version INT DEFAULT 1,
  updated_at TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE artifact (
  id UUID PRIMARY KEY,
  project_id UUID REFERENCES project(id),
  kind TEXT CHECK (kind IN ('facts','screenshot','docx')),
  uri TEXT,
  meta JSONB,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

Алгоритмы

A1. Клонирование и очистка

1. `git ls-remote` для проверки доступа, затем `git clone --depth=1`.
2. Считать `.gitignore` (включая глобальные) → скомпилировать `pathspec`.
3. Пройтись по дереву, удалить/исключить файлы по `pathspec` и правилам «шаблонного мусора» (e.g., `README-templates`, лицензии не учитываются как шум, а отмечаются как метаданные).

A2. Определение языков/стека/архитектуры

- Языки: запустить `enry --json` по корню, агрегировать проценты. Исключить `vendor/`, бинарники.
- Фреймворки/зависимости: эвристики по lock-файлам (`package.json`, `requirements.txt`, `pyproject.toml`, `go.mod`, `pom.xml`) и по импортам в entrypoints.
- Архитектурный тип: правила:
- наличие `app/(routes|controllers)` и веб-зависимостей → `web/api`;
- GUI-ресурсы (`.ui`, `.qml`, `.xib`) → `desktop`;
- только `cli` / `main.go|py` без веб-зависимостей → `cli`.
- Модули/роли: по директориям верхнего уровня и именованию (users, auth, billing, core, infra, domain). Каждому модулю — `evidence`.
- Features: из роутинга (маршруты, описания handler'ов), CLI-команд, публичных API-ендпоинтов.

A3. Построение структуры документа (outline)

- Шаблоны: `course` vs `diploma` (различные глубины и объемы) → дерево разделов с требуемыми целями по страницам.
- Генерация: LLM получает `facts.json` + тип работы → возвращает заголовки и ожидаемые подпункты. Сохраняется как `section.outline`.

A4. Генерация теоретической части

- Ввод: `facts.json`, `outline`, summaries предыдущих частей.
- Вывод: введение, актуальность, обоснование выбранного стека (каждое — отдельная подчасть). Обязательная привязка к фактам (`[см. Feature: auth_login]`).

A5. Генерация практической части

- Описание архитектуры (по A2), ключевые функции через пользовательские сценарии (use cases), ссылки на изображения: `fig:web-home`, `fig:login-flow`.

A6. Заключение

- Краткие выводы, ограничения, будущие работы.

A7. Скриншоты (web)

- Playwright: автозапуск контейнера, `page.goto()`, `page.screenshot()` для маршрутов, найденных в роутинге. Имена файлов связаны с плейсхолдерами в тексте.

Промпты и детерминизм

- Параметры LLM: `temperature=0`, фиксированный `seed` (если поддерживается), строгие инструкции: «не выдумывать факты; использовать только данные из facts.json; ссылаться на features/модули».
- На вход не передаётся исходный код; только извлечённые факты и краткие summaries.

- Каждый раздел — отдельный вызов, в `summary` сохраняется 200–300 слов для передачи в следующий шаг.

DOCX-рендеринг

- Стили: `Heading 1/2/3`, `Caption`, `Normal`.
- Оглавление: вставка поля `TOC \o \"1-3\" \h \z \u`.
- Подписи рисунков: «Рисунок N — Описание», кросс-ссылки в тексте вида «см. рис. N» (MVP — простая нумерация; перекрёстные ссылки-поля возможны в v2).
- Управление объёмом: таргет страниц ±10% через регулировку плотности иллюстраций/диаграмм, длины теории и практики.

API (основные эндпоинты)

```
POST /auth/login
POST /auth/logout
POST /auth/register            # vNext при включении self-signup
GET  /auth/me

POST /projects
GET  /projects/{id}
POST /projects/{id}/sections/{code}/regenerate
GET  /projects/{id}/artifacts/docx

POST /billing/checkout        # создать платёж/подписку
POST /billing/webhook         # обработка провайдерских событий
GET  /billing/status/{id}
```

POST /projects GET /projects/{id} POST /projects/{id}/sections/{code}/regenerate GET /projects/{id}/artifacts/docx ```

Развёртывание

- Docker Compose: `web`, `worker`, `beat`, `redis`, `postgres`, `minio`, `playwright`.
- Логи и аудит: idempotent-ключи задач, фиксация версий инструментов в метаданных artifacts.
- **Auth**: Django Admin для staff; SPA авторизация по токену (SimpleJWT или drf-token).
- **Billing**: провайдер-адаптер (на выбор: Stripe/ЮKassa/CloudPayments) как сервис; защищённый webhook.

(Если ок — план расширений отражён в Milestones.)