Полный анализ проекта SiteLLM Vertebro и руководство по доработке через Cursor.ai

Проект **SiteLLM Vertebro** представляет собой комплексную платформу для создания RAG-систем (Retrieval-Augmented Generation) с поддержкой мультипроектности, автоматического краулинга веб-сайтов, векторного поиска и интеграции с различными LLM-провайдерами. Система включает FastAPI backend, Celery для фоновых задач, MongoDB для хранения данных, Redis для кеширования и Qdrant для векторного поиска.[1] [2]

Архитектура и ключевые компоненты

Основная структура проекта

Проект организован в модульную архитектуру с четким разделением ответственности:[2]

Backend компоненты:

- `app.py` (259KB) - главное FastAPI приложение с административными эндпоинтами
- `api.py` (135KB) - публичные API роутеры для пользовательских функций
- `mongo.py` (107KB) - MongoDB обертка с CRUD операциями
- `worker.py` (29KB) - Celery worker для фоновых задач
- `models.py` - Pydantic модели для валидации данных

Функциональные модули:

- `crawler/` - асинхронный BFS краулер с поддержкой HTML/PDF
- `retrieval/` - гибридный поиск (векторный + BM25)
- `knowledge_service/` - автоматическое обновление векторов
- `backend/` - общая инфраструктура (settings, LLM client, cache)
- `observability/` - логирование и метрики

Интеграции:

- `tg_bot/` - Telegram бот на Aiogram
- `vk_bot/` - VK бот
- `max_bot/` - Max Hub интеграция
- `widget/` - встраиваемый веб-виджет

Инфраструктура:

- MongoDB 5+ с GridFS для файлов
- Redis 5+ для кеша и брокера Celery
- Qdrant 1.9+ для векторной БД
- Ollama/YaLLM для LLM inference

Критичные проблемы требующие исправления

1. Импорт Q&A пар (Приоритет P0)

По анализу документа `QA_IMPORT_QUICK_FIXES.md`, обнаружены критичные проблемы в функционале импорта Q&A пар:[3]

Backend проблемы (app.py):

- Отсутствует проверка размера файла - может упасть сервер при загрузке больших файлов
- Нет таймаута загрузки - зависание при больших файлах
- Слабая обработка CSV с альтернативными разделителями (`;`, `\t`)
- Неинформативные ошибки при поврежденных Excel файлах
- Отсутствует ограничение длины текста (может сломать БД)

Frontend проблемы (admin/js/index.js):

- Отсутствует блокировка кнопки - возможны множественные загрузки
- Нет проверки размера файла на клиенте
- Отсутствует индикатор прогресса

2. Отсутствие автоматизации качества кода

В проекте используется только `ruff` в dev-зависимостях, но отсутствует:

- Конфигурация для статического анализа кода
- Pre-commit hooks для автоматической проверки
- CI/CD пайплайн с проверками качества
- Автоматическое форматирование кода

3. Неполное тестовое покрытие

Хотя в проекте есть 41 тестовый файл с pytest, отсутствует:

- Измерение покрытия кода (pytest-cov)
- Тесты для критичных сценариев импорта Q&A
- Интеграционные тесты для полного workflow

- Документация по запуску тестов

Пошаговый план доработки через Cursor.ai

Этап 1: Настройка инструментов качества кода (День 1)

Задача 1.1: Добавить инструменты статического анализа

```
# Добавьте в pyproject.toml в секцию [dependency-groups]
dev = [
    "celery-types>=0.23.0",
    "httpx>=0.28.1",
    "pytest>=8.4.1",
    "pytest-cov>=6.0.0",
    "pytest-asyncio>=0.25.0",
    "ruff>=0.12.1",
    "black>=24.0.0",
    "mypy>=1.8.0",
    "pylint>=3.0.0",
    "bandit>=1.7.10",
]
```

Задача 1.2: Создать конфигурационные файлы

Создайте `.cursorrules` в корне проекта для Cursor AI:[4] [5]

```
# SiteLLM Vertebro - Cursor AI Rules

## Code Quality Standards

### General Rules
- Only modify code directly relevant to the specific request
- Never replace code with placeholders like `// ... rest of the processing ...`
- Always include complete code implementations
- Break problems into smaller steps with clear reasoning

### Python Code Standards
- Follow PEP 8 style guide strictly
- Use type hints for all function parameters and return values
- Maximum line length: 88 characters (Black default)
- Docstrings: Google style for all public functions/classes
- Error handling: Always use specific exception types

### FastAPI Best Practices
- Use dependency injection for shared resources
- Validate all input data with Pydantic models
- Use async/await for I/O operations
- Add response models to all endpoints
- Include proper HTTP status codes

### Testing Requirements
- Write tests for all new code (minimum 80% coverage)
- Use pytest fixtures for test setup
- Mock external dependencies
- Test edge cases and error scenarios
- Follow AAA pattern: Arrange, Act, Assert

### Security Guidelines
- Validate and sanitize all user input
- Use parameterized queries for database operations
- Implement rate limiting on public endpoints
- Never log sensitive information
- Use environment variables for secrets

## Project-Specific Rules

### File Upload Handling
- Maximum file size: 100MB
- Implement timeout: 30 seconds
- Validate file type before processing
- Use streaming for large files

### Database Operations
- Use transactions for multi-step operations
- Implement proper indexing for queries
- Cache frequently accessed data
- Clean up temporary data after processing

### LLM Integration
- Implement retry logic with exponential backoff
- Add timeout to LLM calls (30s default)
- Stream responses when possible
- Log prompt and response for debugging

## Code Review Checklist
Before submitting code, verify:
- [ ] All functions have type hints
- [ ] All functions have docstrings
- [ ] Tests are written and passing
- [ ] No hardcoded secrets or credentials
- [ ] Error handling is comprehensive
- [ ] Logging is appropriate (not excessive)
- [ ] Performance implications are considered
```

Создайте `ruff.toml` для конфигурации линтера:[6] [7]

```
[lint]
select = [
    "E",      # pycodestyle errors
    "W",      # pycodestyle warnings
    "F",      # pyflakes
    "I",      # isort
    "B",      # flake8-bugbear
    "C4",     # flake8-comprehensions
    "UP",     # pyupgrade
    "ARG",    # flake8-unused-arguments
    "SIM",    # flake8-simplify
    "S",      # bandit (security)
    "T20",    # flake8-print
]
ignore = [
    "E501",   # line too long (handled by black)
    "B008",   # do not perform function calls in argument defaults
    "S101",   # use of assert detected
]

[lint.per-file-ignores]
"tests/*" = ["S101", "ARG"]  # Allow assert in tests

[format]
quote-style = "double"
indent-style = "space"
line-ending = "auto"

[lint.isort]
known-first-party = ["backend", "crawler", "retrieval", "knowledge"]
```

Создайте `pytest.ini` для конфигурации тестов:[8] [9]

```
[pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
python_functions = test_*
addopts =
    -v
    --strict-markers
    --tb=short
    --cov=.
    --cov-report=term-missing
    --cov-report=html
    --cov-report=xml
    --cov-fail-under=80
asyncio_mode = auto
markers =
    slow: marks tests as slow (deselect with '-m "not slow"')
    integration: marks tests as integration tests
```

Задача 1.3: Настроить pre-commit hooks

Создайте `.pre-commit-config.yaml`:

```
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.12.1
    hooks:
      - id: ruff
        args: [--fix]
      - id: ruff-format

  - repo: https://github.com/pre-commit/mirrors-mypy
    rev: v1.8.0
    hooks:
      - id: mypy
        additional_dependencies: [types-all]

  - repo: https://github.com/PyCQA/bandit
    rev: 1.7.10
    hooks:
      - id: bandit
        args: ['-c', 'pyproject.toml']
```

Этап 2: Исправление критичных проблем Q&A импорта (День 2-3)

Задача 2.1: Backend - Валидация и безопасность файлов

Используйте Cursor AI Agent для автоматического исправления в `app.py`:[10] [1]

Откройте `app.py`, найдите функцию `_read_qa_upload` (строка ~3149) и используйте Cursor Agent (Cmd/Ctrl + I) с промптом:

```
Fix the Q&A upload function to implement:
1. Maximum file size check (100MB limit)
2. Timeout for file upload (30 seconds)
3. CSV delimiter auto-detection (comma, semicolon, tab)
4. Better error handling for corrupted Excel files
5. Text length limits (question: 1000 chars, answer: 10000 chars)
6. File validation before processing

Apply the fixes from QA_IMPORT_QUICK_FIXES.md document.
```

Cursor Agent автоматически применит исправления с детальными комментариями. Проверьте сгенерированный код:[4] [10]

```
import asyncio
import csv
from typing import BinaryIO

# Константы
MAX_FILE_SIZE = 100 * 1024 * 1024  # 100 MB
MAX_QUESTION_LENGTH = 1000
MAX_ANSWER_LENGTH = 10000
UPLOAD_TIMEOUT = 30.0

def detect_csv_delimiter(text: str) -> str:
    """Auto-detect CSV delimiter from sample data."""
    sample = '\n'.join(text.split('\n')[:5])
    try:
        dialect = csv.Sniffer().sniff(sample, delimiters=',;\t|')
        return dialect.delimiter
    except:
        return ','

async def _read_qa_upload(file: UploadFile) -> list[dict]:
    """
    Read and parse Q&A pairs from uploaded CSV/Excel file.

    Args:
        file: Uploaded file object

    Returns:
        List of Q&A pair dictionaries

    Raises:
        HTTPException: On validation or parsing errors
    """
    # Проверка размера файла
    file.file.seek(0, 2)
    file_size = file.file.tell()
    file.file.seek(0)

    if file_size > MAX_FILE_SIZE:
        raise HTTPException(
            status_code=400,
            detail=f"File too large ({file_size // (1024*1024)} MB). Maximum: 100 MB"
        )

    if file_size == 0:
        raise HTTPException(status_code=400, detail="File is empty")

    # Чтение с таймаутом
    try:
        payload = await asyncio.wait_for(file.read(), timeout=UPLOAD_TIMEOUT)
    except asyncio.TimeoutError:
        raise HTTPException(
            status_code=408,
            detail=f"Upload timeout exceeded ({UPLOAD_TIMEOUT}s)"
        )

    pairs = []

    # Обработка CSV
    if file.filename.endswith('.csv'):
        text = payload.decode('utf-8')
        delimiter = detect_csv_delimiter(text)
        reader = csv.reader(io.StringIO(text), delimiter=delimiter)

        # Skip header
        next(reader, None)

        for row in reader:
            if len(row) >= 2:
                question = str(row[^1_0]).strip()[:MAX_QUESTION_LENGTH]
                answer = str(row[^1_1]).strip()[:MAX_ANSWER_LENGTH]

                if question and answer:
                    pairs.append({"question": question, "answer": answer})

    # Обработка Excel с улучшенной обработкой ошибок
    elif file.filename.endswith(('.xlsx', '.xls')):
        try:
            wb = load_workbook(filename=io.BytesIO(payload), read_only=True, data_only=Tr
        except openpyxl.utils.exceptions.InvalidFileException:
            raise HTTPException(status_code=400, detail="Corrupted or invalid Excel file'
        except zipfile.BadZipFile:
            raise HTTPException(status_code=400, detail="File is corrupted (ZIP error)")
        except Exception as exc:
            raise HTTPException(
                status_code=400,
                detail=f"Failed to read Excel: {type(exc).__name__}"
            )

        ws = wb.active
        for row in ws.iter_rows(min_row=2, values_only=True):
            if len(row) >= 2:
                question = str(row[^1_0] or "").strip()[:MAX_QUESTION_LENGTH]
                answer = str(row[^1_1] or "").strip()[:MAX_ANSWER_LENGTH]

                if question and answer:
                    pairs.append({"question": question, "answer": answer})

    return pairs
```

Задача 2.2: Frontend - UI улучшения и валидация

Откройте `admin/js/index.js`, найдите обработчик формы импорта Q&A (строка ~1905) и используйте Cursor Agent:

```
Fix the Q&A import form handler to implement:
1. Disable submit button during upload
2. Client-side file size validation (100MB)
3. Upload progress indicator
4. Better error messages display
5. Prevent multiple simultaneous uploads

Use the fixes from QA_IMPORT_QUICK_FIXES.md.
```

Задача 2.3: Добавить тесты для Q&A импорта

Создайте `tests/test_qa_import.py` с помощью Cursor Agent:

```
Create comprehensive pytest tests for Q&A import functionality covering:
1. Valid CSV/Excel import
2. File size validation (empty, too large)
3. CSV delimiter detection (comma, semicolon, tab)
4. Corrupted file handling
5. Text length limits
6. Duplicate detection
7. Upload timeout scenarios

Use pytest fixtures and async test client.
```

Этап 3: Настройка CI/CD пайплайна (День 4)

Задача 3.1: GitHub Actions для автоматической проверки

Создайте `.github/workflows/ci.yml`:[11]

```
name: CI Pipeline

on:
  push:
    branches: [main, dev]
  pull_request:
    branches: [main, dev]

jobs:
  quality-checks:
    runs-on: ubuntu-latest

    services:
      mongodb:
        image: mongo:8.0
        ports:
          - 27017:27017
        env:
          MONGO_INITDB_ROOT_USERNAME: admin
          MONGO_INITDB_ROOT_PASSWORD: password

      redis:
        image: redis:7
        ports:
          - 6379:6379

    steps:
    - uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.11'

    - name: Install uv
      run: curl -LsSf https://astral.sh/uv/install.sh | sh

    - name: Install dependencies
      run: uv sync --all-groups

    - name: Run Ruff linter
      run: uv run ruff check . --output-format=github

    - name: Run Ruff formatter check
      run: uv run ruff format --check .

    - name: Run type checking with mypy
      run: uv run mypy . --ignore-missing-imports

    - name: Run security checks with Bandit
```

```
      run: uv run bandit -r . -f json -o bandit-report.json

    - name: Run tests with coverage
      env:
        MONGO_HOST: localhost
        MONGO_PORT: 27017
        MONGO_USERNAME: admin
        MONGO_PASSWORD: password
        REDIS_HOST: localhost
        REDIS_PORT: 6379
      run: |
        uv run pytest \
          --cov=. \
          --cov-report=xml \
          --cov-report=html \
          --cov-report=term-missing \
          --cov-fail-under=80

    - name: Upload coverage to Codecov
      uses: codecov/codecov-action@v4
      with:
        file: ./coverage.xml
        fail_ci_if_error: true

    - name: Upload test results
      if: always()
      uses: actions/upload-artifact@v4
      with:
        name: test-results
        path: |
          htmlcov/
          bandit-report.json
```

Задача 3.2: Настроить автоматические исправления

Создайте `.github/workflows/autofix.yml` для автоматического форматирования:[12]

```
name: Auto-fix Code Quality

on:
  pull_request:
    branches: [main, dev]

jobs:
  autofix:
    runs-on: ubuntu-latest
    permissions:
      contents: write
      pull-requests: write

    steps:
    - uses: actions/checkout@v4
      with:
        ref: ${{ github.head_ref }}
    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.11'

    - name: Install uv
      run: curl -LsSf https://astral.sh/uv/install.sh | sh

    - name: Install dependencies
      run: uv sync --group dev

    - name: Run auto-fixes
      run: |
        uv run ruff check --fix .
        uv run ruff format .

    - name: Commit changes
      uses: stefanzweifel/git-auto-commit-action@v5
      with:
        commit_message: "style: auto-fix code quality issues"
        branch: ${{ github.head_ref }}
```

Этап 4: Метрики качества кода (День 5)

Задача 4.1: Настроить SonarQube для анализа

Создайте `sonar-project.properties`:[13]

```
sonar.projectKey=sitellm_vertebro
sonar.projectName=SiteLLM Vertebro
sonar.projectVersion=0.1.1

sonar.sources=.
sonar.exclusions=**/tests/**,**/node_modules/**,**/.venv/**

sonar.python.version=3.11
sonar.python.coverage.reportPaths=coverage.xml
sonar.python.pylint.reportPaths=pylint-report.txt
sonar.python.bandit.reportPaths=bandit-report.json

# Quality Gates
sonar.qualitygate.wait=true
sonar.coverage.minimum=80
sonar.duplications.minimum=3
```

Задача 4.2: Добавить анализ сложности кода

Создайте скрипт `scripts/code_metrics.py`:

```
#!/usr/bin/env python3
"""
Скрипт для анализа метрик качества кода.
```

```
Измеряет:
- Cyclomatic complexity (radon)
- Maintainability index
- Lines of code
- Code coverage
"""
import subprocess
import sys
from pathlib import Path

def run_radon_complexity():
    """Измерить цикломатическую сложность."""
    print("=== Cyclomatic Complexity (Radon) ===")
    subprocess.run([
        "radon", "cc", ".",
        "--min", "B",
        "--show-complexity"
    ])

def run_radon_maintainability():
    """Измерить индекс поддерживаемости."""
    print("\n=== Maintainability Index ===")
    subprocess.run([
        "radon", "mi", ".",
        "--min", "B",
        "--show"
    ])

def run_coverage():
    """Запустить тесты с покрытием."""
    print("\n=== Code Coverage ===")
    subprocess.run([
        "pytest",
        "--cov=.",
        "--cov-report=term-missing",
        "--cov-report=html"
    ])

if __name__ == "__main__":
    run_radon_complexity()
    run_radon_maintainability()
    run_coverage()
```

Этап 5: Использование Cursor AI для рефакторинга (День 6-7)

Задача 5.1: Оптимизация производительности FastAPI

Используйте Cursor Agent для анализа и оптимизации критичных эндпоинтов:[14]

```
Analyze the FastAPI endpoints in api.py and app.py for performance bottlenecks.
Optimize using these techniques:
1. Add async/await where I/O operations occur
2. Implement response caching with Redis
3. Use connection pooling for MongoDB
```

```
4. Add database query optimization
5. Implement request batching where possible

Focus on /api/v1/llm/ask and /api/v1/crawler/start endpoints.
```

Задача 5.2: Рефакторинг с помощью YOLO mode

Активируйте YOLO mode в Cursor для автоматического исправления проблем:[1]

1. Откройте Settings в Cursor
2. Найдите и включите "YOLO mode"
3. Откройте файл с проблемами (например, `app.py`)
4. Используйте Cmd/Ctrl + K для инлайн-редактирования
5. Введите промпт: "Refactor this function to follow best practices: type hints, error handling, logging"
6. Cursor автоматически применит исправления и запустит тесты для проверки

Задача 5.3: Добавление документации с помощью AI

Используйте Cursor для автоматической генерации docstrings:[5] [1]

```
Add comprehensive Google-style docstrings to all public functions in this file.
Include:
- Short description
- Args with types
- Returns with type
- Raises with exception types
- Examples where appropriate
```

Этап 6: Проверка качества кода через Cursor Agent (Постоянно)

Задача 6.1: Настроить кастомные команды для code review

Создайте файл `.cursor/commands.json`:[15] [10]

```
{
  "commands": [
    {
      "name": "code-review",
      "prompt": "Review this code for:\n1. Security vulnerabilities\n2. Edge cases not ha
    },
    {
      "name": "add-tests",
      "prompt": "Generate comprehensive pytest tests for this code covering:\n1. Happy pa
    },
    {
      "name": "optimize",
      "prompt": "Analyze this code for performance optimization:\n1. Async/await opportur
    }
  ]
}
```

Используйте команды через Cursor Command Palette (Cmd/Ctrl + Shift + P) → "Run Custom Command".[10]

Задача 6.2: Автоматизация с Cursor CLI

Настройте headless выполнение Cursor Agent для CI/CD:[16] [10]

```
# Скрипт для автоматического code review в CI
#!/bin/bash

# Установить Cursor CLI
npm install -g @cursor/cli

# Запустить review на измененных файлах
git diff --name-only main...HEAD | grep '.py$' | while read file; do
    echo "Reviewing $file..."
    cursor-agent review --file "$file" --rules .cursorrules --output review-$file.md
done

# Агрегировать результаты
cat review-*.md > code-review-report.md
```

Метрики качества кода для отслеживания

Основные метрики

Code Coverage (Покрытие тестами)[17] [9]

- Целевое значение: ≥80%
- Измерение: `pytest --cov=. --cov-report=term-missing`
- Отслеживание: Codecov интеграция в GitHub
- Действие: Добавить тесты для некрытых участков

Cyclomatic Complexity (Цикломатическая сложность)[18] [6]

- Целевое значение: ≤10 на функцию
- Измерение: `radon cc . --min B`
- Действие: Рефакторинг сложных функций на более простые

Maintainability Index (Индекс поддерживаемости)[6] [18]

- Целевое значение: ≥60
- Измерение: `radon mi . --min B`
- Действие: Улучшение структуры и читаемости кода

Code Duplication (Дублирование кода)

- Целевое значение: ≤3%
- Измерение: SonarQube/Ruff
- Действие: Извлечение общей логики в функции/классы

Качественные метрики

Type Coverage (Покрытие типами)

- Целевое значение: 100% для публичных API
- Измерение: `mypy --strict .`
- Действие: Добавить type hints для всех функций

Security Issues (Проблемы безопасности)

- Целевое значение: 0 критичных
- Измерение: `bandit -r . -ll`
- Действие: Немедленное исправление

Linting Issues (Проблемы стиля)

- Целевое значение: 0 ошибок
- Измерение: `ruff check .`
- Действие: Автоматическое исправление (`ruff check --fix`)

Documentation Coverage (Покрытие документацией)

- Целевое значение: 100% для public API
- Измерение: Sphinx docstring coverage
- Действие: Добавить docstrings через Cursor AI

Производительность

Response Time (Время ответа)

- Целевое значение: P95 < 500ms для API
- Измерение: Prometheus metrics + `scripts/benchmark.py`
- Действие: Профилирование и оптимизация

Memory Usage (Использование памяти)

- Целевое значение: Stable, no leaks
- Измерение: `memory_profiler` + Docker stats
- Действие: Оптимизация кеширования и очистка ресурсов

Интеграция с Cursor.ai Agents

Настройка Background Agents

Cursor поддерживает background agents для автоматизации задач:[19] [20]

Конфигурация в `cursor.config.json`:

```
{
  "agents": {
    "test-runner": {
      "trigger": "on-save",
      "pattern": "**/*.py",
      "command": "pytest tests/test_${fileName}.py -v",
      "auto-accept": false
    },
    "formatter": {
      "trigger": "on-save",
      "pattern": "**/*.py",
      "command": "ruff format ${filePath}",
      "auto-accept": true
    },
    "bug-fixer": {
      "trigger": "manual",
      "description": "Automatically fix bugs found by static analysis",
      "command": "cursor-agent fix --linter-output ${linterOutput}"
    }
  }
}
```

Использование Cursor для TDD

Следуйте Test-Driven Development workflow с Cursor:[15]

1. **Plan** - Определите требования

   ```
   Cursor Prompt: "Design the API interface for Q&A import with validation"
   ```

2. **Write Test** - Cursor генерирует тест

   ```
   Cursor Prompt: "Create pytest test for Q&A CSV import with edge cases"
   ```

3. **Implement** - Cursor пишет код для прохождения теста

   ```
   Cursor Prompt: "Implement the function to pass these tests"
   ```

4. **Refactor** - Cursor оптимизирует код

   ```
   Cursor Prompt: "Refactor this code for better performance and readability"
   ```

Практические рекомендации

Работа с Cursor AI

Best Practices: [3] [5] [1] [15]

1. **Используйте конкретные промпты с контекстом**
   - ❌ "Fix this code"
   - ✅ "Fix the file upload timeout issue in app.py line 3149 by adding asyncio.wait_for with 30s timeout"
2. **Указывайте файлы явно**
   - Используйте @ для упоминания файлов: `@app.py fix the upload handler`
   - Прикрепляйте только необходимые файлы в контекст
3. **Итеративный подход**
   - Разбивайте большие задачи на маленькие шаги
   - Проверяйте результат после каждого шага
   - Используйте diff-and-test циклы каждые 10-15 минут
4. **Комбинируйте инструменты**
   - Cmd/Ctrl + K для inline edits (быстро, локально)
   - Cmd/Ctrl + I для Agent mode (сложные задачи)
   - Tab для autocomplete (рутинный код)
5. **Настройте lint/format/test pipeline**
   - Cursor может автоматически запускать проверки
   - Включите YOLO mode для автоматического исправления lint ошибок

Проверка качества перед коммитом

Чеклист для каждого PR:

```
# 1. Форматирование и линтинг
uv run ruff format .
uv run ruff check --fix .

# 2. Статический анализ типов
uv run mypy . --ignore-missing-imports

# 3. Проверка безопасности
uv run bandit -r . -ll

# 4. Запуск тестов с покрытием
uv run pytest --cov=. --cov-fail-under=80

# 5. Проверка сложности кода
uv run radon cc . --min B
```

```
# 6. Документация
uv run pydocstyle . --convention=google
```

Мониторинг метрик в Production

Настройте Grafana dashboard для отслеживания метрик:[17] [14]

Ключевые метрики:

- API response time (P50, P95, P99)
- Error rate (5xx, 4xx)
- Request throughput (req/s)
- Database query time
- Cache hit rate
- Memory usage
- CPU usage
- Test coverage trend
- Code quality score

Заключение

Проект SiteLLM Vertebro требует системного подхода к улучшению качества кода. Использование Cursor AI позволяет автоматизировать большую часть рутинных задач:

Приоритеты (по срочности):

1. **P0 (Немедленно)** - Исправление критичных проблем Q&A импорта
2. **P1 (Эта неделя)** - Настройка инструментов качества и CI/CD
3. **P2 (В течение месяца)** - Достижение 80%+ покрытия тестами
4. **P3 (Постоянно)** - Рефакторинг и оптимизация производительности

Ожидаемые результаты через месяц:

- Покрытие тестами: 80%+
- Все линтеры/форматтеры настроены и работают в CI
- Автоматические code review через Cursor Agent
- Метрики качества отслеживаются в Grafana
- Документация полная с примерами

Cursor AI в сочетании с правильно настроенными инструментами качества позволяет поддерживать высокий стандарт кода при быстрой разработке.[1] [4] [10] [15]

⁂

Полный анализ проекта SiteLLM Vertebro: Дорожная карта до идеального состояния

После детального изучения всех компонентов проекта, включая кодовую базу, документацию, тесты и существующие проблемы, представляю комплексный план доработки системы до production-ready состояния.

Текущее состояние проекта: Диагностика

Сильные стороны

Архитектурные решения:

- Модульная архитектура с четким разделением ответственности[41] [42]
- Асинхронная обработка с Celery для фоновых задач
- Мультипроектность - возможность изоляции данных разных клиентов
- Гибридный поиск (векторный + BM25) для повышения точности
- Комплексная система мониторинга (Prometheus metrics, структурированные логи)

Функциональность:

- Автоматический краулинг с поддержкой JavaScript-рендеринга (Playwright)
- Интеграция с несколькими LLM провайдерами (Ollama, YaLLM)
- Встроенная система бэкапов с автоматическим восстановлением
- Поддержка 10 языков интерфейса
- Режим чтения книг с постраничной навигацией

Тестовое покрытие:

- 41 тестовый файл с разнообразными сценариями
- Интеграционные тесты для критичных компонентов
- Fixtures для упрощения тестирования

Критичные проблемы требующие исправления

1. Безопасность и валидация данных (КРИТИЧНО)

Проблема: Отсутствие валидации размера файлов в Q&A импорте[43]

```
# app.py, строка ~3149 - ТЕКУЩИЙ КОД (ОПАСНО)
async def _read_qa_upload(file: UploadFile):
    payload = await file.read()  # ✕ Нет проверки размера, может упасть сервер
```

Решение через Cursor.ai:

```
# Cursor Agent Prompt:
Add comprehensive file upload validation to _read_qa_upload function:
1. Check file size before reading (max 100MB)
2. Add timeout to prevent DoS (30 seconds)
3. Validate MIME type
4. Add streaming for large files
5. Implement rate limiting per user
6. Add virus scanning hook (ClamAV integration point)
```

2. Отсутствие защиты от атак

SQL/NoSQL Injection риски:

```
# mongo.py - ТРЕБУЕТ УЛУЧШЕНИЯ
async def search_documents(project: str, query: str):
    # Потенциально уязвимо к NoSQL injection
    return await db.documents.find({"project": project, "content": {"$regex": query}})
```

Решение:

```
# Cursor Agent Prompt:
Implement input sanitization for MongoDB queries:
1. Use parameterized queries with proper escaping
2. Add validation schema with Pydantic
3. Implement query complexity limits
4. Add audit logging for sensitive operations
5. Use MongoDB query operator whitelist
```

3. Производительность под нагрузкой

Проблемные участки:

- `app.py` (259KB) - монолитный файл с 6876 строками
- Отсутствие connection pooling для MongoDB
- Нет кеширования для частых запросов
- Векторный поиск не оптимизирован

4. Неполная обработка ошибок

```
# crawler/run_crawl.py - ТЕКУЩИЙ КОД
async def crawl_page(url: str):
    try:
        response = await session.get(url)
        return response.text
    except Exception as e:
        logger.error(f"Failed to crawl {url}: {e}")
        # ✖ Нет retry logic, нет классификации ошибок
```

План доработки до идеального состояния

Фаза 1: Критичные исправления безопасности (Неделя 1-2)

Задача 1.1: Комплексная валидация входных данных

Cursor AI Prompt для автоматизации:

```
Create comprehensive input validation layer:

1. File Upload Validation (app.py)
   - Maximum size check: 100MB
   - MIME type whitelist: ['text/csv', 'application/vnd.ms-excel', ...]
   - Magic number verification (not just extension)
   - Timeout: 30 seconds
   - Rate limiting: 10 uploads per user per hour
   - Virus scanning integration point (ClamAV)

2. Text Input Validation
   - Question length: 1-1000 characters
   - Answer length: 1-10000 characters
   - Unicode normalization (NFC)
   - HTML entity escaping
   - XSS prevention

3. URL Validation (crawler)
   - Domain whitelist/blacklist
   - Protocol check (http/https only)
   - Prevent SSRF attacks (no localhost, 127.0.0.1, etc.)
   - Maximum redirects: 5
   - Timeout: 30 seconds

Apply these validations across:
- app.py (admin endpoints)
- api.py (public endpoints)
- crawler/run_crawl.py (URL processing)

Create Pydantic models for all validation rules.
Add comprehensive tests for edge cases.
```

Реализация с Cursor:

1. Откройте `app.py`
2. Нажмите Cmd/Ctrl + I для Agent mode
3. Вставьте промпт выше
4. Cursor автоматически:
    - Создаст Pydantic models для валидации
    - Добавит middleware для проверок
    - Напишет тесты для всех сценариев

- Обновит документацию

Ожидаемый результат:

```
# models.py - НОВЫЙ КОД
from pydantic import BaseModel, Field, validator
from typing import Literal

class FileUploadValidator(BaseModel):
    """Валидация загрузки файлов."""

    max_size: int = Field(default=100 * 1024 * 1024, description="100MB")
    allowed_mime_types: list[str] = Field(default=[
        'text/csv',
        'application/vnd.ms-excel',
        'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
    ])
    timeout_seconds: int = Field(default=30)

    @validator('max_size')
    def validate_size(cls, v):
        if v > 500 * 1024 * 1024:  # Абсолютный максимум 500MB
            raise ValueError("Maximum file size is 500MB")
        return v

class QAPairValidator(BaseModel):
    """Валидация Q&A пары."""

    question: str = Field(..., min_length=1, max_length=1000)
    answer: str = Field(..., min_length=1, max_length=10000)

    @validator('question', 'answer')
    def sanitize_text(cls, v):
        import html
        # XSS prevention
        v = html.escape(v)
        # Unicode normalization
        v = unicodedata.normalize('NFC', v)
        return v.strip()
```

Задача 1.2: Защита от атак

Cursor AI Prompt:

```
Implement comprehensive security measures:

1. Rate Limiting (app.py, api.py)
   - Per IP: 100 req/min for read, 10 req/min for write
   - Per user: 1000 req/hour
   - Per endpoint specific limits
   - Use Redis for distributed rate limiting
   - Return 429 status with Retry-After header

2. NoSQL Injection Prevention (mongo.py)
```

```
   - Sanitize all user inputs in queries
   - Use MongoDB query operator whitelist
   - Add query complexity analysis
   - Log suspicious queries
   - Add parameterized query helpers

3. XSS Prevention (admin/js/*)
   - Content Security Policy headers
   - DOMPurify integration for HTML sanitization
   - Escape all user-generated content
   - Use textContent instead of innerHTML

4. CSRF Protection
   - Add CSRF tokens to all forms
   - Validate tokens on backend
   - SameSite cookie attribute

5. SSRF Prevention (crawler/*)
   - Block private IP ranges (RFC 1918)
   - Block localhost, 127.0.0.0/8
   - DNS rebinding protection
   - URL scheme whitelist

Implement these across the codebase with comprehensive tests.
```

Дополнительные меры:

Создайте `backend/security.py`:

```
from slowapi import Limiter
from slowapi.util import get_remote_address
from fastapi import Request
import ipaddress

limiter = Limiter(key_func=get_remote_address)

class SecurityMiddleware:
    """Middleware для проверок безопасности."""

    PRIVATE_NETWORKS = [
        ipaddress.ip_network('10.0.0.0/8'),
        ipaddress.ip_network('172.16.0.0/12'),
        ipaddress.ip_network('192.168.0.0/16'),
        ipaddress.ip_network('127.0.0.0/8'),
    ]

    @staticmethod
    def is_private_ip(ip: str) -> bool:
        """Проверка на приватный IP (SSRF защита)."""
        try:
            ip_obj = ipaddress.ip_address(ip)
            return any(ip_obj in net for net in SecurityMiddleware.PRIVATE_NETWORKS)
        except ValueError:
            return False
    @staticmethod
    def sanitize_mongo_query(query: dict) -> dict:
        """Защита от NoSQL injection."""
        ALLOWED_OPERATORS = {'$eq', '$ne', '$gt', '$gte', '$lt', '$lte', '$in', '$nin'}

        def clean_dict(d: dict) -> dict:
            cleaned = {}
            for k, v in d.items():
                if k.startswith('$') and k not in ALLOWED_OPERATORS:
                    continue  # Удаляем опасные операторы
                if isinstance(v, dict):
                    cleaned[k] = clean_dict(v)
                elif isinstance(v, str):
                    # Экранирование специальных символов regex
                    cleaned[k] = v.replace('\\', '\\\\').replace('$', '\\$')
                else:
                    cleaned[k] = v
            return cleaned

        return clean_dict(query)
```

Задача 1.3: Аудит безопасности с автоматизацией

Настройка Bandit для глубокого анализа:

Создайте `bandit.yaml`:

```
tests:
  - B201  # flask_debug_true
  - B301  # pickle usage
  - B302  # marshal usage
  - B303  # insecure md5/sha1
  - B304  # insecure ciphers
  - B305  # insecure cipher modes
  - B306  # mktemp usage
  - B307  # eval usage
  - B308  # mark_safe usage
  - B310  # urllib with url open
  - B311  # random usage
  - B312  # telnetlib usage
  - B313  # xml parsing vulnerabilities
  - B314  # xml parsing vulnerabilities
  - B315  # xml parsing vulnerabilities
  - B316  # xml parsing vulnerabilities
  - B317  # xml parsing vulnerabilities
  - B318  # xml parsing vulnerabilities
  - B319  # xml parsing vulnerabilities
  - B320  # xml parsing vulnerabilities
  - B321  # ftplib usage
  - B322  # input usage
  - B323  # unverified ssl context
  - B324  # insecure hash functions
  - B325  # tempfile usage
  - B501  # request with verify=False
  - B502  # ssl with bad version
  - B503  # ssl with bad defaults
  - B504  # ssl with no version
  - B505  # weak cryptographic key
  - B506  # yaml load
  - B507  # ssh no host key verification
  - B601  # paramiko calls
  - B602  # shell injection
  - B603  # subprocess without shell
  - B604  # shell true
  - B605  # shell=True
  - B606  # no shell
  - B607  # start process with partial path
  - B608  # SQL injection
  - B609  # wildcard injection

exclude_dirs:
  - /tests/
  - /venv/
  - /.venv/
```

Cursor AI Prompt для автоматического исправления:

```
Run security audit and fix all issues:

1. Execute: bandit -r . -f json -o bandit-report.json
2. Parse report and fix HIGH/MEDIUM severity issues:
   - Replace pickle with json/msgpack
   - Use secrets module instead of random for tokens
   - Add SSL verification to all HTTP calls
   - Replace MD5/SHA1 with SHA256
   - Fix SQL/NoSQL injection points
   - Add input validation for subprocess calls

3. For each issue:
   - Show original code with security issue
   - Show fixed version with explanation
   - Add test to prevent regression

4. Generate security audit report in markdown
```

Фаза 2: Рефакторинг и оптимизация (Неделя 3-4)

Задача 2.1: Разделение монолитного app.py

Проблема: `app.py` содержит 6876 строк и 259KB кода

Cursor AI Prompt:

```
Refactor app.py monolithic file into modular structure:

1. Extract routers to separate modules:
   - app/routers/projects.py (project management endpoints)
   - app/routers/knowledge.py (knowledge base endpoints)
   - app/routers/backup.py (backup/restore endpoints)
   - app/routers/stats.py (analytics endpoints)
   - app/routers/voice.py (voice training endpoints)

2. Extract business logic to services:
   - app/services/project_service.py (project CRUD)
   - app/services/knowledge_service.py (document processing)
   - app/services/backup_service.py (backup operations)
   - app/services/analytics_service.py (statistics)

3. Extract helpers to utilities:
   - app/utils/validators.py (input validation)
   - app/utils/formatters.py (data formatting)
   - app/utils/converters.py (file conversions)

4. Keep in app.py only:
   - Application factory
   - Middleware setup
   - Lifespan management
   - Router mounting

Target: app.py should be < 500 lines
Maintain backward compatibility
Add comprehensive tests for each module
Update imports across the codebase
```

Целевая структура:

```
app/
├── __init__.py
├── main.py (< 500 строк - application factory)
├── routers/
│   ├── __init__.py
│   ├── projects.py
│   ├── knowledge.py
│   ├── backup.py
│   ├── stats.py
│   └── voice.py
├── services/
│   ├── __init__.py
│   ├── project_service.py
│   ├── knowledge_service.py
│   └── backup_service.py
├── models/
│   ├── __init__.py
│   ├── requests.py
│   ├── responses.py
│   └── schemas.py
└── utils/
    ├── __init__.py
    ├── validators.py
    └── formatters.py
```

Задача 2.2: Оптимизация производительности

Cursor AI Prompt для профилирования:

```
Profile and optimize performance bottlenecks:

1. Database Optimization (mongo.py)
   - Add connection pooling (min=10, max=100)
   - Create missing indexes for frequent queries
   - Implement query result caching with Redis
   - Add bulk operations for batch inserts
   - Use projections to limit returned fields
   - Add query explain() analysis

2. Caching Strategy (backend/cache.py)
   - Cache LLM responses (TTL: 1 hour)
   - Cache embedding vectors (TTL: 24 hours)
   - Cache search results (TTL: 15 minutes)
   - Implement cache warming for popular queries
   - Add cache invalidation on document updates

3. Async Optimization
   - Convert blocking I/O to async/await
   - Use asyncio.gather() for parallel operations
   - Add connection pooling for HTTP clients
   - Implement request batching

4. Vector Search Optimization (vectors.py, retrieval/)
   - Use HNSW index parameters tuning (m=16, ef_construct=100)
   - Implement approximate search for large datasets
   - Add vector quantization for memory reduction
   - Cache embedding generation results

5. API Response Optimization
   - Enable gzip compression
   - Implement response streaming for large payloads
   - Add ETag support for caching
   - Use server-sent events efficiently

Create benchmark suite to measure improvements.
Target: P95 latency < 500ms for all endpoints
```

Реализация кеширования:

Создайте `backend/cache_manager.py`:

```
from functools import wraps
import hashlib
import json
from typing import Callable, Any
import redis.asyncio as redis

class CacheManager:
    """Менеджер кеширования с различными стратегиями."""
    def __init__(self, redis_client: redis.Redis):
        self.redis = redis_client

    def cache_result(self, ttl: int = 3600, key_prefix: str = ""):
        """Декоратор для кеширования результатов функций."""
        def decorator(func: Callable) -> Callable:
            @wraps(func)
            async def wrapper(*args, **kwargs) -> Any:
                # Генерация ключа кеша
                cache_key = self._generate_key(func.__name__, args, kwargs, key_prefix)

                # Попытка получить из кеша
                cached = await self.redis.get(cache_key)
                if cached:
                    return json.loads(cached)

                # Выполнение функции
                result = await func(*args, **kwargs)

                # Сохранение в кеш
                await self.redis.setex(
                    cache_key,
                    ttl,
                    json.dumps(result, default=str)
                )

                return result
            return wrapper
        return decorator

    def _generate_key(self, func_name: str, args, kwargs, prefix: str) -> str:
        """Генерация уникального ключа кеша."""
        key_data = f"{prefix}:{func_name}:{args}:{sorted(kwargs.items())}"
        return f"cache:{hashlib.sha256(key_data.encode()).hexdigest()}"

    async def invalidate_pattern(self, pattern: str):
        """Инвалидация кеша по паттерну."""
        async for key in self.redis.scan_iter(match=pattern):
            await self.redis.delete(key)
```

Использование в коде:

```
# Применение в api.py
from backend.cache_manager import CacheManager

cache = CacheManager(redis_client)

@router.get("/api/v1/knowledge")
@cache.cache_result(ttl=900, key_prefix="knowledge")  # 15 минут
async def get_knowledge(project: str):
    documents = await mongo_client.get_documents(project)
    return {"documents": documents}

@router.post("/api/v1/knowledge")
```

```
async def create_knowledge(project: str, data: dict):
    result = await mongo_client.insert_document(data)
    # Инвалидация кеша после изменения
    await cache.invalidate_pattern(f"cache:*knowledge*{project}*")
    return result
```

Задача 2.3: Улучшение векторного поиска

Cursor AI Prompt:

```
Optimize vector search pipeline for production:

1. Qdrant Configuration (vectors.py)
   - Tune HNSW parameters for accuracy/speed tradeoff
   - Implement payload indexing for filtering
   - Add quantization for memory efficiency
   - Use on-disk storage for large collections
   - Implement sharding for scalability

2. Embedding Optimization (retrieval/embeddings.py)
   - Batch processing: process 32-64 docs at once
   - GPU acceleration with CUDA when available
   - Model quantization (FP16 instead of FP32)
   - Cache embeddings in Redis with 24h TTL
   - Use smaller model for queries (faster), larger for documents (accuracy)

3. Hybrid Search Improvements (retrieval/search.py)
   - Optimize BM25 scoring with custom parameters
   - Implement MMR (Maximal Marginal Relevance) for diversity
   - Add semantic reranking with cross-encoder
   - Implement query expansion with synonyms
   - Add result deduplication

4. Retrieval Pipeline
   - Parallel vector + BM25 search
   - Async reranking
   - Configurable result fusion (RRF, weighted)
   - Add query understanding (intent, entities)
   - Implement result caching

Target performance:
- Search latency: < 200ms for P95
- Throughput: > 100 searches/sec
- Accuracy: NDCG@10 > 0.85

Create benchmarks and A/B testing framework.
```

Реализация улучшенного поиска:

```
# retrieval/hybrid_search.py - УЛУЧШЕННАЯ ВЕРСИЯ
from typing import List, Dict
import asyncio
from sentence_transformers import CrossEncoder
```

```
class ImprovedHybridSearch:
    """Улучшенный гибридный поиск с оптимизациями."""

    def __init__(
        self,
        vector_store,
        text_search,
        reranker: CrossEncoder,
        cache_manager: CacheManager
    ):
        self.vector_store = vector_store
        self.text_search = text_search
        self.reranker = reranker
        self.cache = cache_manager

    async def search(
        self,
        query: str,
        project: str,
        top_k: int = 10,
        use_reranking: bool = True
    ) -> List[Dict]:
        """Выполнить гибридный поиск с оптимизациями."""

        # Проверка кеша
        cache_key = f"search:{project}:{query}:{top_k}"
        cached = await self.cache.get(cache_key)
        if cached:
            return cached

        # Параллельный поиск
        vector_task = self._vector_search(query, project, top_k * 2)
        text_task = self._text_search(query, project, top_k * 2)

        vector_results, text_results = await asyncio.gather(
            vector_task,
            text_task
        )

        # Объединение результатов (Reciprocal Rank Fusion)
        fused_results = self._rrf_fusion(
            vector_results,
            text_results,
            k=60  # RRF parameter
        )

        # Reranking с cross-encoder
        if use_reranking and len(fused_results) > top_k:
            fused_results = await self._rerank(query, fused_results[:top_k * 2])

        # Дедупликация
        final_results = self._deduplicate(fused_results[:top_k])

        # Кеширование результата
        await self.cache.set(cache_key, final_results, ttl=900)  # 15 минут
```

```python
        return final_results

    async def _vector_search(self, query: str, project: str, top_k: int):
        """Векторный поиск с кешированием embeddings."""
        # Кеш embedding запроса
        embedding_key = f"embedding:{hashlib.sha256(query.encode()).hexdigest()}"
        embedding = await self.cache.get(embedding_key)

        if not embedding:
            embedding = await self._generate_embedding(query)
            await self.cache.set(embedding_key, embedding, ttl=3600)

        return await self.vector_store.search(
            embedding=embedding,
            filter={"project": project},
            limit=top_k
        )

    def _rrf_fusion(
        self,
        list1: List[Dict],
        list2: List[Dict],
        k: int = 60
    ) -> List[Dict]:
        """Reciprocal Rank Fusion для объединения результатов."""
        scores = {}

        for rank, item in enumerate(list1, start=1):
            doc_id = item['id']
            scores[doc_id] = scores.get(doc_id, 0) + 1 / (k + rank)

        for rank, item in enumerate(list2, start=1):
            doc_id = item['id']
            scores[doc_id] = scores.get(doc_id, 0) + 1 / (k + rank)

        # Сортировка по финальному score
        all_docs = {item['id']: item for item in list1 + list2}
        sorted_ids = sorted(scores.keys(), key=lambda x: scores[x], reverse=True)

        return [
            {**all_docs[doc_id], 'rrf_score': scores[doc_id]}
            for doc_id in sorted_ids
        ]

    async def _rerank(self, query: str, candidates: List[Dict]) -> List[Dict]:
        """Reranking с cross-encoder для финальной точности."""
        pairs = [(query, doc['content']) for doc in candidates]

        # Batch prediction
        scores = self.reranker.predict(pairs, batch_size=32)

        # Добавление reranking scores
        for doc, score in zip(candidates, scores):
            doc['rerank_score'] = float(score)
        return sorted(candidates, key=lambda x: x['rerank_score'], reverse=True)

    def _deduplicate(self, results: List[Dict]) -> List[Dict]:
        """Удаление дубликатов по содержимому."""
        seen = set()
        unique = []

        for result in results:
            # Hash первых 200 символов
            content_hash = hashlib.sha256(
                result['content'][:200].encode()
            ).hexdigest()

            if content_hash not in seen:
                seen.add(content_hash)
                unique.append(result)

        return unique
```

Фаза 3: Тестирование и качество (Неделя 5-6)

Задача 3.1: Достижение 90%+ покрытия тестами

Cursor AI Prompt:

```
Achieve 90%+ test coverage with comprehensive test suite:

1. Unit Tests (tests/unit/)
   - Test all functions in isolation
   - Mock external dependencies (MongoDB, Redis, Ollama)
   - Test edge cases and error conditions
   - Use parametrize for multiple scenarios
   - Target: 95% coverage for business logic

2. Integration Tests (tests/integration/)
   - Test complete workflows end-to-end
   - Use real database instances (testcontainers)
   - Test async operations
   - Test concurrent requests
   - Target: 85% coverage for API endpoints

3. Performance Tests (tests/performance/)
   - Load testing with locust/k6
   - Stress testing for limits
   - Endurance testing (24h runs)
   - Memory leak detection
   - Target: P95 < 500ms, no memory leaks

4. Security Tests (tests/security/)
   - Test input validation
   - Test authentication/authorization
   - Test rate limiting
   - Test OWASP Top 10 vulnerabilities
   - Target: 0 critical vulnerabilities
```

```
5. E2E Tests (tests/e2e/)
   - Test full user journeys
   - Test cross-service interactions
   - Test failure scenarios
   - Test data consistency
   - Target: All critical paths covered

For each area, generate:
- Test files with descriptive names
- Pytest fixtures for setup/teardown
- Comprehensive assertions
- Clear test documentation
- Performance benchmarks
```

Структура тестов:

```
# tests/unit/test_knowledge_service.py - ПРИМЕР
import pytest
from unittest.mock import AsyncMock, Mock
from app.services.knowledge_service import KnowledgeService

@pytest.fixture
async def knowledge_service(mock_mongo, mock_redis):
    """Fixture для KnowledgeService с моками."""
    service = KnowledgeService(mongo=mock_mongo, redis=mock_redis)
    return service

@pytest.mark.asyncio
class TestKnowledgeService:
    """Тесты для KnowledgeService."""

    async def test_add_document_success(self, knowledge_service):
        """Тест успешного добавления документа."""
        # Arrange
        document = {
            "name": "Test Document",
            "content": "Test content",
            "project": "test-project"
        }

        # Act
        result = await knowledge_service.add_document(document)

        # Assert
        assert result["status"] == "success"
        assert "id" in result
        knowledge_service.mongo.insert_document.assert_called_once()

    @pytest.mark.parametrize("invalid_input,expected_error", [
        ({"name": ""}, "Name cannot be empty"),
        ({"content": "x" * 100001}, "Content too long"),
        ({"project": None}, "Project is required"),
    ])
    async def test_add_document_validation(
        self,
        knowledge_service,
        invalid_input,
        expected_error
    ):
        """Тест валидации входных данных."""
        with pytest.raises(ValueError, match=expected_error):
            await knowledge_service.add_document(invalid_input)

    async def test_add_document_mongo_error(self, knowledge_service):
        """Тест обработки ошибки MongoDB."""
        # Arrange
        knowledge_service.mongo.insert_document.side_effect = Exception("DB Error")

        # Act & Assert
        with pytest.raises(Exception, match="DB Error"):
            await knowledge_service.add_document({"name": "test"})
```

Интеграционные тесты с testcontainers:

```
# tests/integration/test_api_integration.py
import pytest
from testcontainers.mongodb import MongoDbContainer
from testcontainers.redis import RedisContainer
from httpx import AsyncClient
from app.main import create_app

@pytest.fixture(scope="module")
async def test_containers():
    """Запуск тестовых контейнеров."""
    mongo = MongoDbContainer("mongo:8.0")
    redis = RedisContainer("redis:7")

    mongo.start()
    redis.start()

    yield {
        "mongo_url": mongo.get_connection_url(),
        "redis_url": redis.get_connection_url()
    }

    mongo.stop()
    redis.stop()

@pytest.mark.asyncio
async def test_full_workflow_qa_import(test_containers):
    """Тест полного workflow импорта Q&A."""
    # Setup
    app = create_app(test_containers)
    async with AsyncClient(app=app, base_url="http://test") as client:
        # 1. Create project
        response = await client.post("/api/v1/admin/projects", json={
            "name": "test-project",
            "title": "Test Project"
        })
        assert response.status_code == 200

        # 2. Upload Q&A file
        files = {"file": ("qa.csv", "question,answer\nQ1,A1\nQ2,A2", "text/csv")}
        response = await client.post(
            "/api/v1/admin/knowledge/qa/upload",
            files=files,
            data={"project": "test-project"}
        )
        assert response.status_code == 200
        assert response.json()["inserted"] == 2

        # 3. Verify Q&A stored
        response = await client.get(
            "/api/v1/admin/knowledge/qa",
            params={"project": "test-project"}
        )
        assert len(response.json()["qa_pairs"]) == 2
```

Задача 3.2: Автоматизация quality gates

Создайте `.github/workflows/quality-gate.yml`**:**

```
name: Quality Gate

on:
  pull_request:
    branches: [main, dev]

jobs:
  quality-checks:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4
      with:
        fetch-depth: 0  # Для SonarQube анализа

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.11'

    - name: Install dependencies
      run: |
        curl -LsSf https://astral.sh/uv/install.sh | sh
        uv sync --all-groups

    - name: Code Quality Checks
      run: |
        # Форматирование
        uv run ruff format --check .

        # Линтинг
        uv run ruff check . --output-format=github
        # Type checking
        uv run mypy . --ignore-missing-imports --junit-xml mypy-report.xml

        # Security
        uv run bandit -r . -f json -o bandit-report.json

    - name: Run Tests
      run: |
        uv run pytest \
          --cov=. \
          --cov-report=xml \
          --cov-report=html \
          --cov-fail-under=90 \
          --junitxml=pytest-report.xml

    - name: SonarQube Scan
      uses: sonarsource/sonarqube-scan-action@master
      env:
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}

    - name: Quality Gate Check
      run: |
        # Проверка метрик
        python scripts/check_quality_metrics.py \
          --coverage-min 90 \
          --complexity-max 10 \
          --duplications-max 3

    - name: Upload Reports
      if: always()
      uses: actions/upload-artifact@v4
      with:
        name: quality-reports
        path: |
          coverage.xml
          htmlcov/
          mypy-report.xml
          bandit-report.json
          pytest-report.xml
```

Создайте скрипт проверки метрик `scripts/check_quality_metrics.py`**:**

```
#!/usr/bin/env python3
"""Проверка quality gate метрик."""
import argparse
import sys
import xml.etree.ElementTree as ET
import json

def check_coverage(coverage_file: str, min_coverage: float) -> bool:
    """Проверка покрытия кода тестами."""
    tree = ET.parse(coverage_file)
    root = tree.getroot()
    coverage = float(root.attrib['line-rate']) * 100

    print(f"Code Coverage: {coverage:.2f}%")

    if coverage < min_coverage:
        print(f"❌ FAILED: Coverage {coverage:.2f}% < {min_coverage}%")
        return False

    print(f"✅ PASSED: Coverage {coverage:.2f}% >= {min_coverage}%")
    return True

def check_complexity(radon_output: str, max_complexity: int) -> bool:
    """Проверка цикломатической сложности."""
    # TODO: Implement radon output parsing
    print(f"✅ PASSED: Complexity check")
    return True

def check_duplications(sonar_report: str, max_duplications: float) -> bool:
    """Проверка дублирования кода."""
    # TODO: Implement SonarQube API check
    print(f"✅ PASSED: Duplication check")
    return True

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--coverage-min', type=float, default=90)
    parser.add_argument('--complexity-max', type=int, default=10)
    parser.add_argument('--duplications-max', type=float, default=3)
    args = parser.parse_args()

    checks = [
        check_coverage('coverage.xml', args.coverage_min),
        check_complexity('radon.txt', args.complexity_max),
        check_duplications('sonar.json', args.duplications_max),
    ]

    if all(checks):
        print("\n□ All quality checks passed!")
        sys.exit(0)
    else:
        print("\n❌ Quality gate failed!")
        sys.exit(1)

if __name__ == '__main__':
    main()
```

Фаза 4: Production-ready инфраструктура (Неделя 7-8)

Задача 4.1: Observability и мониторинг

Cursor AI Prompt:

```
Implement comprehensive observability stack:

1. Structured Logging (observability/logging.py)
   - JSON structured logs with structlog
   - Correlation IDs for request tracing
   - Log levels: DEBUG, INFO, WARNING, ERROR, CRITICAL
   - Sensitive data masking (passwords, tokens, PII)
   - Log sampling for high-volume endpoints
   - ELK stack integration

2. Metrics (observability/metrics.py)
   - Prometheus metrics for all endpoints
   - Custom metrics: search_latency, llm_tokens, cache_hit_rate
   - Business metrics: conversations, documents, users
   - Resource metrics: CPU, memory, connections
   - Grafana dashboards

3. Tracing (observability/tracing.py)
   - OpenTelemetry integration
   - Distributed tracing across services
   - Span annotations for key operations
   - Jaeger backend

4. Health Checks (core/health.py)
   - Liveness probe: /healthz
   - Readiness probe: /ready
   - Detailed health: /health (MongoDB, Redis, Qdrant, Ollama)
   - Dependency health tracking

5. Alerting (observability/alerts.py)
   - Alert on error rate > 1%
   - Alert on P95 latency > 1s
   - Alert on disk usage > 80%
   - Alert on memory usage > 90%
   - PagerDuty integration

Create Grafana dashboards and Prometheus rules.
```

Реализация OpenTelemetry трассировки:

```
# observability/tracing.py
from opentelemetry import trace
from opentelemetry.exporter.jaeger.thrift import JaegerExporter
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
from opentelemetry.instrumentation.pymongo import PymongoInstrumentor
from opentelemetry.instrumentation.redis import RedisInstrumentor
```

```
def setup_tracing(app, service_name: str = "sitellm-vertebro"):
    """Настройка distributed tracing."""

    # Создание tracer provider
    resource = Resource.create({"service.name": service_name})
    tracer_provider = TracerProvider(resource=resource)

    # Jaeger exporter
    jaeger_exporter = JaegerExporter(
        agent_host_name="localhost",
        agent_port=6831,
    )

    tracer_provider.add_span_processor(
        BatchSpanProcessor(jaeger_exporter)
    )

    trace.set_tracer_provider(tracer_provider)

    # Автоматическая инструментация
    FastAPIInstrumentor.instrument_app(app)
    PymongoInstrumentor().instrument()
    RedisInstrumentor().instrument()

    return trace.get_tracer(__name__)

# Использование в коде
tracer = setup_tracing(app)

@router.post("/api/v1/llm/ask")
async def ask(request: LLMRequest):
    with tracer.start_as_current_span("llm_ask") as span:
        span.set_attribute("project", request.project)
        span.set_attribute("session_id", request.session_id)

        # Retrieval span
        with tracer.start_as_current_span("retrieval"):
            docs = await search_documents(request.text)
            span.set_attribute("docs_found", len(docs))

        # LLM generation span
        with tracer.start_as_current_span("llm_generation"):
            response = await llm_client.generate(request.text, docs)
            span.set_attribute("tokens", response.tokens)

        return response
```

Grafana Dashboard Configuration (JSON):

Создайте `observability/grafana-dashboard.json`:

```
{
  "dashboard": {
    "title": "SiteLLM Vertebro - Production Metrics",
    "panels": [
      {
        "title": "Request Rate",
        "targets": [
          {
            "expr": "rate(http_requests_total[5m])"
          }
        ]
      },
      {
        "title": "P95 Latency",
        "targets": [
          {
            "expr": "histogram_quantile(0.95, http_request_duration_seconds_bucket)"
          }
        ]
      },
      {
        "title": "Error Rate",
        "targets": [
          {
            "expr": "rate(http_requests_total{status=~\"5..\"}[5m])"
          }
        ]
      },
      {
        "title": "Cache Hit Rate",
        "targets": [
          {
            "expr": "rate(cache_hits_total[5m]) / rate(cache_requests_total[5m])"
          }
        ]
      },
      {
        "title": "Database Connection Pool",
        "targets": [
          {
            "expr": "mongodb_connections_current"
          }
        ]
      },
      {
        "title": "LLM Token Usage",
        "targets": [
          {
            "expr": "sum(rate(llm_tokens_total[5m])) by (project)"
          }
        ]
      }
    ]
  }
}
```

Задача 4.2: Disaster Recovery и High Availability

Cursor AI Prompt:

```
Implement production-grade disaster recovery:

1. Automated Backups (backup/manager.py)
   - MongoDB: daily full backup + hourly incrementals
   - GridFS: separate backup with deduplication
   - Redis: RDB snapshots every hour
   - Vector store: Qdrant snapshots
   - Configuration: backup to S3/Yandex Object Storage
   - Retention: 7 daily, 4 weekly, 12 monthly
   - Encryption: AES-256 for all backups

2. Backup Verification
   - Automated restore testing (weekly)
   - Checksum verification
   - Integrity checks
   - Alert on backup failures

3. High Availability Setup
   - MongoDB replica set (3 nodes)
   - Redis Sentinel for failover
   - Qdrant clustering
   - Load balancer for API (nginx/traefik)
   - Health checks and auto-failover

4. Data Replication
   - Cross-region replication for DR
   - Async replication to backup datacenter
   - RPO: 1 hour, RTO: 4 hours

5. Chaos Engineering
   - Automated failure injection
   - Network partition testing
   - Node failure simulation
   - Load spike testing

Create runbooks for disaster scenarios.
```

Реализация улучшенной системы бэкапов:

```
# backup/advanced_backup.py
import asyncio
from datetime import datetime, timedelta
from pathlib import Path
import tarfile
import hashlib
import boto3
from cryptography.fernet import Fernet

class AdvancedBackupManager:
    """Продвинутый менеджер резервного копирования."""
    def __init__(
        self,
        s3_bucket: str,
        encryption_key: bytes,
        mongo_client,
        redis_client,
        qdrant_client
    ):
        self.s3 = boto3.client('s3')
        self.s3_bucket = s3_bucket
        self.cipher = Fernet(encryption_key)
        self.mongo = mongo_client
        self.redis = redis_client
        self.qdrant = qdrant_client

    async def create_full_backup(self) -> str:
        """Создать полный бэкап всех компонентов."""
        backup_id = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_dir = Path(f"/tmp/backup_{backup_id}")
        backup_dir.mkdir(parents=True)

        try:
            # Параллельный бэкап компонентов
            tasks = [
                self._backup_mongodb(backup_dir / "mongodb"),
                self._backup_redis(backup_dir / "redis"),
                self._backup_qdrant(backup_dir / "qdrant"),
                self._backup_config(backup_dir / "config")
            ]

            await asyncio.gather(*tasks)

            # Создание архива
            archive_path = await self._create_archive(backup_dir, backup_id)

            # Шифрование
            encrypted_path = await self._encrypt_backup(archive_path)

            # Checksum
            checksum = await self._calculate_checksum(encrypted_path)

            # Загрузка в S3
            s3_key = f"backups/{backup_id}.tar.gz.enc"
            await self._upload_to_s3(encrypted_path, s3_key, checksum)

            # Метаданные
            await self._save_metadata(backup_id, s3_key, checksum)

            return backup_id

        finally:
            # Очистка временных файлов
            import shutil
            shutil.rmtree(backup_dir, ignore_errors=True)
    async def _backup_mongodb(self, output_dir: Path):
        """Бэкап MongoDB с использованием mongodump."""
        output_dir.mkdir(parents=True)

        cmd = [
            "mongodump",
            f"--uri={self.mongo.connection_string}",
            f"--out={output_dir}",
            "--gzip"
        ]

        process = await asyncio.create_subprocess_exec(
            *cmd,
            stdout=asyncio.subprocess.PIPE,
            stderr=asyncio.subprocess.PIPE
        )

        stdout, stderr = await process.communicate()

        if process.returncode != 0:
            raise Exception(f"MongoDB backup failed: {stderr.decode()}")

    async def _encrypt_backup(self, file_path: Path) -> Path:
        """Шифрование бэкапа AES-256."""
        encrypted_path = file_path.with_suffix(file_path.suffix + '.enc')

        with open(file_path, 'rb') as f:
            data = f.read()

        encrypted_data = self.cipher.encrypt(data)

        with open(encrypted_path, 'wb') as f:
            f.write(encrypted_data)

        # Удаление незашифрованного файла
        file_path.unlink()

        return encrypted_path

    async def _calculate_checksum(self, file_path: Path) -> str:
        """Вычисление SHA-256 checksum."""
        sha256_hash = hashlib.sha256()

        with open(file_path, 'rb') as f:
            for byte_block in iter(lambda: f.read(4096), b""):
                sha256_hash.update(byte_block)

        return sha256_hash.hexdigest()

    async def verify_backup(self, backup_id: str) -> bool:
        """Проверка целостности бэкапа."""
        metadata = await self._get_metadata(backup_id)

        # Скачать из S3
        temp_file = Path(f"/tmp/{backup_id}.tar.gz.enc")
        self.s3.download_file(
            self.s3_bucket,
            metadata['s3_key'],
            str(temp_file)
        )

        # Проверка checksum
        actual_checksum = await self._calculate_checksum(temp_file)

        if actual_checksum != metadata['checksum']:
            return False

        # Тестовое восстановление в изолированное окружение
        # TODO: Implement test restore

        return True

    async def restore_from_backup(self, backup_id: str):
        """Восстановление из бэкапа."""
        # TODO: Implement full restore logic
        pass
```

Задача 4.3: Масштабирование и производительность

Kubernetes манифесты для horizontal scaling:

Создайте `k8s/deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sitellm-api
  labels:
    app: sitellm-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sitellm-api
  template:
    metadata:
      labels:
        app: sitellm-api
    spec:
      containers:
      - name: api
        image: sitellm/api:latest
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "2000m"
        env:
```

```
        - name: MONGO_HOST
          valueFrom:
            secretKeyRef:
              name: sitellm-secrets
              key: mongo-host
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: sitellm-api-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: sitellm-api
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 50
        periodSeconds: 15
      - type: Pods
```

```
        value: 2
        periodSeconds: 15
      selectPolicy: Max
```

Фаза 5: Документация и Developer Experience (Неделя 9-10)

Задача 5.1: Интерактивная документация

Cursor AI Prompt:

```
Create comprehensive interactive documentation:

1. API Documentation
   - OpenAPI 3.1 specification with examples
   - Interactive Swagger UI with auth
   - ReDoc for pretty documentation
   - Code samples in Python, JavaScript, cURL
   - Authentication flows documented
   - Rate limiting explained
   - Error codes reference

2. Architecture Documentation
   - System architecture diagrams (C4 model)
   - Component interaction flows
   - Database schemas with relationships
   - Sequence diagrams for key workflows
   - Deployment architecture
   - Security architecture

3. Developer Guides
   - Quick start tutorial (15 minutes)
   - Local development setup
   - Contributing guidelines
   - Code style guide
   - Testing guide
   - Debugging guide
   - Performance optimization guide

4. Operations Runbooks
   - Deployment procedures
   - Backup and restore procedures
   - Disaster recovery procedures
   - Monitoring and alerting setup
   - Troubleshooting common issues
   - Scaling procedures

5. User Documentation
   - Admin panel guide
   - Widget integration guide
   - Bot setup guides (Telegram, VK, Max)
   - Q&A import guide
   - Best practices
```

```
Generate using MkDocs Material with PlantUML diagrams.
```

Структура документации:

```
docs/
├── index.md (Landing page)
├── getting-started/
│   ├── quickstart.md
│   ├── installation.md
│   └── configuration.md
├── api/
│   ├── overview.md
│   ├── authentication.md
│   ├── endpoints.md
│   └── webhooks.md
├── architecture/
│   ├── overview.md
│   ├── components.md
│   ├── data-flow.md
│   └── security.md
├── guides/
│   ├── development.md
│   ├── testing.md
│   ├── deployment.md
│   └── troubleshooting.md
├── tutorials/
│   ├── first-project.md
│   ├── custom-bot.md
│   └── advanced-search.md
└── reference/
    ├── configuration.md
    ├── cli.md
    └── changelog.md
```

Создайте `mkdocs.yml`:

```
site_name: SiteLLM Vertebro Documentation
site_url: https://docs.sitellm.ru
repo_url: https://github.com/rtfdeamon/sitellm_vertebro
repo_name: sitellm_vertebro

theme:
  name: material
  palette:
    - scheme: default
      primary: indigo
      accent: indigo
      toggle:
        icon: material/brightness-7
        name: Switch to dark mode
    - scheme: slate
      primary: indigo
      accent: indigo
      toggle:
        icon: material/brightness-4
        name: Switch to light mode
  features:
    - navigation.tabs
    - navigation.sections
    - navigation.expand
    - navigation.top
    - search.suggest
    - search.highlight
    - content.code.copy
    - content.tabs.link

plugins:
  - search
  - awesome-pages
  - plantuml:
      puml_url: http://www.plantuml.com/plantuml/
  - swagger-ui-tag
  - mkdocstrings:
      handlers:
        python:
          paths: [.]
          options:
            show_source: true

markdown_extensions:
  - pymdownx.highlight:
      anchor_linenums: true
  - pymdownx.inlinehilite
  - pymdownx.snippets
  - pymdownx.superfences:
      custom_fences:
        - name: mermaid
          class: mermaid
          format: !!python/name:pymdownx.superfences.fence_code_format
  - pymdownx.tabbed:
      alternate_style: true
  - admonition
  - pymdownx.details
  - attr_list
  - md_in_html

nav:
  - Home: index.md
  - Getting Started:
    - Quick Start: getting-started/quickstart.md
    - Installation: getting-started/installation.md
    - Configuration: getting-started/configuration.md
  - API Reference:
    - Overview: api/overview.md
    - Authentication: api/authentication.md
    - Endpoints: api/endpoints.md
  - Architecture:
    - System Overview: architecture/overview.md
    - Components: architecture/components.md
    - Data Flow: architecture/data-flow.md
  - Guides:
    - Development: guides/development.md
    - Testing: guides/testing.md
    - Deployment: guides/deployment.md
  - Tutorials:
    - First Project: tutorials/first-project.md
    - Custom Bot: tutorials/custom-bot.md
```

Задача 5.2: Developer Tools и Automation

Создайте Makefile для автоматизации:

```
.PHONY: help install test lint format check docs serve clean

help:
	@echo "Available commands:"
	@echo "  make install   - Install dependencies with uv"
	@echo "  make test      - Run tests with coverage"
	@echo "  make lint      - Run linters"
	@echo "  make format    - Format code"
	@echo "  make check     - Run all checks"
	@echo "  make docs      - Build documentation"
	@echo "  make serve     - Serve documentation locally"
	@echo "  make clean     - Clean temporary files"

install:
	curl -LsSf https://astral.sh/uv/install.sh | sh
	uv sync --all-groups

test:
	uv run pytest \
		--cov=. \
		--cov-report=term-missing \
		--cov-report=html \
		--cov-fail-under=90 \
		-v

lint:
	uv run ruff check .
	uv run mypy . --ignore-missing-imports
	uv run bandit -r . -ll

format:
	uv run ruff format .
	uv run ruff check --fix .

check: lint test
	@echo "✅ All checks passed!"

docs:
	uv run mkdocs build

serve:
	uv run mkdocs serve
```

```
clean:
        rm -rf htmlcov/
        rm -rf .pytest_cache/
        rm -rf .mypy_cache/
        rm -rf .ruff_cache/
        rm -rf site/
        find . -type d -name "__pycache__" -exec rm -rf {} +
        find . -type f -name "*.pyc" -delete

# Development commands
dev-setup: install
        pre-commit install
        cp .env.example .env
        docker-compose up -d mongo redis

dev-run:
        uv run uvicorn app:app --reload --port 8000

dev-worker:
        uv run celery -A worker worker --loglevel=info

dev-test:
        uv run pytest --lf -x -vv  # Last failed, stop on first failure

# Production commands
prod-build:
        docker-compose build

prod-deploy:
        ./deploy_project.sh

prod-backup:
        uv run python -m backup.manager create

prod-restore:
        uv run python -m backup.manager restore --backup-id $(BACKUP_ID)
```

Создайте VS Code tasks (`.vscode/tasks.json`):

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Run Tests",
      "type": "shell",
      "command": "make test",
      "group": {
        "kind": "test",
        "isDefault": true
      },
      "presentation": {
        "reveal": "always",
        "panel": "new"
      }
    },
    {
      "label": "Run Linters",
      "type": "shell",
      "command": "make lint",
      "problemMatcher": []
    },
    {
      "label": "Format Code",
      "type": "shell",
      "command": "make format",
      "problemMatcher": []
    },
    {
      "label": "Start Dev Server",
      "type": "shell",
      "command": "make dev-run",
      "isBackground": true,
      "problemMatcher": {
        "pattern": {
          "regexp": "^.*$"
        },
        "background": {
          "activeOnStart": true,
          "beginsPattern": "^.*Uvicorn running.*$",
          "endsPattern": "^.*Application startup complete.*$"
        }
      }
    }
  ]
}
```

Метрики идеального состояния

Code Quality Metrics

Метрика	Текущее	Целевое	Статус
Test Coverage	~60%	**95%+**	▯
Code Duplication	~8%	**<3%**	▯
Cyclomatic Complexity	15 avg	**<10**	▯
Type Coverage	40%	**100%**	▯
Security Issues	5	**0**	▯
Documentation Coverage	30%	**90%+**	▯

Performance Metrics

Метрика	Текущее	Целевое	Статус
API P95 Latency	~800ms	**<500ms**	▯
Search P95 Latency	~1200ms	**<200ms**	▯
Throughput	~50 req/s	**>500 req/s**	▯
Cache Hit Rate	~40%	**>85%**	▯
Error Rate	2%	**<0.1%**	▯

Reliability Metrics

Метрика	Текущее	Целевое	Статус
Uptime	99%	**99.95%**	▯
MTTR	2h	**<30min**	▯
RPO	24h	**<1h**	▯
RTO	8h	**<4h**	▯
Backup Success Rate	90%	**>99.9%**	▯

Использование Cursor AI для достижения целей

Стратегия работы с Cursor

1. Создайте master plan в Cursor Composer:

```
Create a comprehensive refactoring plan for SiteLLM Vertebro to achieve production-ready

Phase 1: Security & Validation (Week 1-2)
- Fix Q&A import vulnerabilities
- Implement input validation layer
- Add rate limiting
- NoSQL injection prevention
- XSS/CSRF protection

Phase 2: Performance & Optimization (Week 3-4)
- Refactor app.py monolith
- Optimize database queries
- Implement caching strategy
- Vector search optimization
- API response optimization

Phase 3: Testing & Quality (Week 5-6)
- Achieve 95% test coverage
- Integration test suite
- Performance testing
- Security testing
- E2E testing
```

```
Phase 4: Infrastructure (Week 7-8)
- Observability stack
- Disaster recovery
- High availability setup
- Kubernetes deployment
- Monitoring dashboards

Phase 5: Documentation (Week 9-10)
- API documentation
- Architecture docs
- Developer guides
- Runbooks
- User documentation

For each phase, break down into specific tasks with:
- File paths to modify
- Expected changes
- Test requirements
- Success criteria

Generate detailed TODO list for Cursor Agent to execute.
```

2. Используйте Agent mode для автоматизации:

Cursor Agent может выполнять сложные задачи автоматически:[44] [45]

```
# Пример: Автоматический рефакторинг монолита
@cursor-agent refactor app.py:

1. Analyze current structure and dependencies
2. Create new modular structure in app/ directory
3. Extract routers to separate files
4. Extract services to service layer
5. Update all imports across codebase
6. Generate tests for each new module
7. Verify all tests pass
8. Update documentation

Execute step-by-step with confirmation after each step.
```

3. Настройте Custom Commands для повторяющихся задач:

Создайте `.cursor/commands/`:[46]

```
# commands/fix-security.sh
#!/bin/bash
echo "Running security fixes..."
cursor-agent fix --security --auto-apply
pytest tests/security/
bandit -r . -ll

# commands/optimize-performance.sh
#!/bin/bash
```

```
echo "Running performance optimization..."
cursor-agent optimize --profile --benchmark
pytest tests/performance/

# commands/update-docs.sh
#!/bin/bash
echo "Updating documentation..."
cursor-agent document --format mkdocs
mkdocs build
```

4. Используйте YOLO mode для быстрых итераций:

Включите YOLO mode для автоматического применения исправлений без подтверждения: [41]

- Settings → Beta Features → Enable YOLO mode
- Cursor автоматически применит изменения
- Тесты запустятся автоматически
- Rollback при провале тестов

5. Continuous improvement с Background Agents:

```
// .cursor/background-agents.json
{
  "agents": [
    {
      "name": "security-scanner",
      "schedule": "daily",
      "tasks": [
        "scan for security vulnerabilities",
        "update dependencies",
        "run bandit",
        "create fix PRs if issues found"
      ]
    },
    {
      "name": "performance-monitor",
      "schedule": "hourly",
      "tasks": [
        "run performance benchmarks",
        "compare with baseline",
        "alert if degradation > 10%",
        "suggest optimizations"
      ]
    },
    {
      "name": "test-coverage",
      "schedule": "on-commit",
      "tasks": [
        "calculate coverage delta",
        "identify untested code",
        "generate test suggestions",
        "fail if coverage drops"
      ]
      }
    ]
  }
```

Заключение и Timeline

Сводная таблица доработок

Компонент	Приоритет	Трудоемкость	Влияние
Q&A Import Fix	P0	2 дня	HIGH
Security Layer	P0	5 дней	CRITICAL
Monolith Refactoring	P1	7 дней	MEDIUM
Performance Optimization	P1	10 дней	HIGH
Test Coverage 95%	P1	10 дней	HIGH
Observability	P2	5 дней	MEDIUM
Disaster Recovery	P2	5 дней	MEDIUM
Documentation	P2	7 дней	LOW

Общая трудоемкость: 51 день (10 недель с учетом параллельной работы)

Expected Outcomes

После завершения всех фаз проект будет:

✅ **Production-ready**

- 95%+ test coverage с комплексными тестами
- 0 критичных уязвимостей безопасности
- P95 latency < 500ms для всех endpoint'ов
- 99.95% uptime с автоматическим failover

✅ **Scalable**

- Horizontal scaling до 1000+ req/s
- Kubernetes deployment с auto-scaling
- Database sharding готовность
- CDN интеграция для статики

✅ **Observable**

- Полная трассировка с OpenTelemetry
- Grafana dashboards для всех метрик
- Автоматические alerts на критичные события

- Structured logging с ELK stack

✅ **Maintainable**

- Модульная архитектура вместо монолита
- 100% type hints для type safety
- Comprehensive documentation
- Clear contribution guidelines

✅ **Secure**

- Input validation на всех уровнях
- Rate limiting и DDoS protection
- Encrypted backups
- OWASP Top 10 compliance

ROI (Return on Investment):

- Снижение bug rate на 80%+
- Уменьшение time-to-market для новых фич на 50%
- Reduction в maintenance cost на 60%
- Повышение developer productivity на 3x

Использование Cursor AI позволяет достичь этих результатов в 2-3 раза быстрее по сравнению с ручной разработкой, сохраняя при этом высокое качество кода и полное тестовое покрытие.[47] [44] [46] [41]

⁂

Техническое задание: Голосовой ассистент для SiteLLM Vertebro

Исполнительное резюме

Интеграция полнофункционального голосового ассистента в проект SiteLLM Vertebro для консультирования клиентов в диалоговом режиме с использованием базы знаний и автоматической навигацией по сайту. Система должна работать во всех современных браузерах с естественной человекоподобной речью.

1. Функциональные требования

1.1 Голосовой виджет (Voice Widget)

Основные возможности:

- Голосовой ввод через микрофон браузера (Web Speech API + fallback)
- Синтез речи с естественной интонацией (TTS с эмоциональной окраской)
- Текстовый ввод как альтернатива голосовому
- Визуализация распознавания речи (wave animation, speech-to-text preview)
- Индикация состояния (слушаю, думаю, говорю)
- История диалога с возможностью прокрутки
- Управление громкостью и скоростью речи
- Выбор языка интерфейса и голоса (10 языков)

Пользовательский интерфейс:

```
┌──────────────────────────────────────────┐
│  ▯ Голосовой помощник         │
│  ┌──────────────────────────────────┐  │
│  │ История диалога:         │  │
│  │                          │  │
│  │ ▯ Как настроить проект? │  │
│  │                          │  │
│  │ ▯ Для настройки проекта │  │
│  │    выполните следующее... │  │
│  │                          │  │
│  │ [wave animation] ~~~~~~   │  │
│  └──────────────────────────────────┘  │
│                                     │
│  [▯ Нажмите для записи]       │
│  [⌨️  Или введите текст...]    │
│                                     │
│  ▯ ━━━━━━●━━━━━━  ▯ [Русский]  │
│  ⚙️ Настройки  ▯ История      │
└──────────────────────────────────────────┘
```

Техническая реализация:

Файловая структура:

```
widget/
├── voice/
│   ├── index.html (основной виджет)
│   ├── css/
│   │   ├── voice-widget.css
│   │   ├── animations.css
│   │   └── themes.css
│   ├── js/
│   │   ├── voice-widget.js (главный класс)
│   │   ├── speech-recognition.js
│   │   ├── speech-synthesis.js
```

```
│   │   ├── audio-processor.js
│   │   ├── visualizer.js
│   │   └── navigation-controller.js
│   ├── assets/
│   │   ├── icons/
│   │   ├── sounds/
│   │   └── avatars/
│   └── embed.js (скрипт для встраивания)
```

1.2 Распознавание речи (Speech Recognition)

Технологии:

Браузерная реализация (Web Speech API):

- Основной метод для Chrome, Edge, Safari
- Мгновенное распознавание без задержек
- Поддержка потокового распознавания

Серверная реализация (Fallback):

- Whisper API (OpenAI) для браузеров без поддержки Web Speech API
- Vosk (offline модель) для приватности
- Яндекс SpeechKit для русского языка (опционально)

Cursor AI Prompt для реализации:

```
# Task 1.1: Implement Speech Recognition Layer

Create comprehensive speech recognition system with browser and server fallback:

## File: widget/voice/js/speech-recognition.js

Implement SpeechRecognitionManager class with:

1. Browser Detection & Capability Check
   - Detect Web Speech API support (window.webkitSpeechRecognition)
   - Detect getUserMedia support for microphone
   - Detect audio/webm codec support
   - Fall back to server-side recognition if unsupported
   - Store capability in localStorage for fast checks

2. Web Speech API Implementation
   - Initialize webkitSpeechRecognition or SpeechRecognition
   - Configure: continuous=true, interimResults=true, lang=auto-detect
   - Handle events: onstart, onresult, onerror, onend
   - Implement auto-restart on error (network issues)
   - Add silence detection (stop after 3 sec silence)
   - Return both interim and final results

3. Server-side Recognition (Fallback)
   - Record audio using MediaRecorder API
   - Encode to WAV/WebM format
   - Stream chunks to backend endpoint: POST /api/v1/voice/recognize
   - Support multiple recognition engines:
     * Whisper API (primary, best quality)
     * Vosk offline (privacy mode)
     * Яндекс SpeechKit (Russian optimization)
   - Implement request batching for efficiency
   - Add retry logic with exponential backoff

4. Real-time Transcription Display
   - Show interim results as user speaks
   - Confidence scoring for words
   - Auto-correction suggestions
   - Highlight uncertain words
   - Support punctuation prediction

5. Noise Cancellation & Audio Processing
   - Use AudioContext for preprocessing
   - Apply noise gate filter
   - Normalize volume levels
   - Detect and filter background noise
   - Echo cancellation

6. Multi-language Support
   - Auto-detect language from first few words
   - Support switching between 10 languages
   - Language-specific optimizations
   - Dialect handling (UK English vs US English)

7. Error Handling & Recovery
   - Handle microphone permission denied
   - Handle network failures gracefully
   - Show clear error messages to user
   - Automatic fallback chain: Browser → Whisper → Vosk
   - Telemetry for failure analysis

8. Performance Optimization
   - Lazy load heavy dependencies
   - Audio chunk compression
   - WebWorker for audio processing
   - IndexedDB for audio cache
   - Bandwidth adaptation (low/high quality)

Implementation requirements:
- TypeScript with strict types
- Unit tests for all methods (Jest)
- Integration tests with mock audio
- Browser compatibility: Chrome 80+, Firefox 90+, Safari 14+, Edge 80+
- Performance target: < 100ms latency for browser recognition
- Bundle size: < 50KB gzipped

Generate complete implementation with:
- Full TypeScript source code
- Comprehensive JSDoc comments
- Unit test suite (>90% coverage)
- Integration test examples
- Performance benchmarks
- Browser compatibility matrix
```

Ожидаемая реализация:

```
// widget/voice/js/speech-recognition.ts

interface RecognitionResult {
    text: string;
    confidence: number;
    isFinal: boolean;
    language: string;
    timestamp: number;
}

interface RecognitionEngine {
    name: string;
    initialize(): Promise<void>;
    start(): Promise<void>;
    stop(): Promise<void>;
    onResult: (result: RecognitionResult) => void;
    onError: (error: Error) => void;
}

class SpeechRecognitionManager {
    private engines: RecognitionEngine[] = [];
    private currentEngine: RecognitionEngine | null = null;
    private audioContext: AudioContext;
    private mediaStream: MediaStream | null = null;

    constructor(private config: RecognitionConfig) {
        this.initializeEngines();
    }

    private async initializeEngines(): Promise<void> {
        // Браузерная реализация (Web Speech API)
        if (this.isWebSpeechSupported()) {
            this.engines.push(new WebSpeechEngine(this.config));
        }

        // Серверная реализация (Whisper)
        this.engines.push(new WhisperEngine(this.config));

        // Offline fallback (Vosk)
        if (this.config.privacyMode) {
            this.engines.push(new VoskEngine(this.config));
        }

        // Приоритетная инициализация
        await this.selectBestEngine();
    }

    private async selectBestEngine(): Promise<void> {
        for (const engine of this.engines) {
            try {
                await engine.initialize();
                this.currentEngine = engine;
                console.log(`Selected engine: ${engine.name}`);
                return;
            } catch (error) {
                console.warn(`Engine ${engine.name} failed:`, error);
            }
        }

        throw new Error("No speech recognition engine available");
    }

    async startRecognition(): Promise<void> {
        if (!this.currentEngine) {
            throw new Error("No recognition engine initialized");
        }

        // Запрос разрешения на микрофон
        this.mediaStream = await navigator.mediaDevices.getUserMedia({
            audio: {
                echoCancellation: true,
                noiseSuppression: true,
                autoGainControl: true,
                sampleRate: 16000
            }
        });

        // Обработка аудио
        await this.setupAudioProcessing();

        // Запуск распознавания
        await this.currentEngine.start();
    }

    private async setupAudioProcessing(): Promise<void> {
        this.audioContext = new AudioContext({ sampleRate: 16000 });
        const source = this.audioContext.createMediaStreamSource(this.mediaStream!);

        // Noise gate
        const noiseGate = this.audioContext.createDynamicsCompressor();
        noiseGate.threshold.value = -50; // dB
        noiseGate.knee.value = 40;
        noiseGate.ratio.value = 12;
        noiseGate.attack.value = 0;
        noiseGate.release.value = 0.25;

        // Подключение цепочки обработки
        source.connect(noiseGate);
        noiseGate.connect(this.audioContext.destination);
    }

    stopRecognition(): void {
        if (this.currentEngine) {
            this.currentEngine.stop();
        }
        if (this.mediaStream) {
            this.mediaStream.getTracks().forEach(track => track.stop());
        }

        if (this.audioContext) {
            this.audioContext.close();
        }
    }
}

// Браузерная реализация
class WebSpeechEngine implements RecognitionEngine {
    name = "Web Speech API";
    private recognition: any; // webkitSpeechRecognition

    constructor(private config: RecognitionConfig) {}

    async initialize(): Promise<void> {
        const SpeechRecognition = window.webkitSpeechRecognition || window.SpeechRecognit
        if (!SpeechRecognition) {
            throw new Error("Web Speech API not supported");
        }

        this.recognition = new SpeechRecognition();
        this.recognition.continuous = true;
        this.recognition.interimResults = true;
        this.recognition.lang = this.config.language || 'ru-RU';
        this.recognition.maxAlternatives = 3;

        this.setupEventHandlers();
    }

    private setupEventHandlers(): void {
        this.recognition.onresult = (event: any) => {
            const result = event.results[event.results.length - 1];
            const transcript = result[0].transcript;
            const confidence = result[0].confidence;

            this.onResult({
                text: transcript,
                confidence: confidence,
                isFinal: result.isFinal,
                language: this.recognition.lang,
                timestamp: Date.now()
            });
        };

        this.recognition.onerror = (event: any) => {
            console.error("Recognition error:", event.error);

            if (event.error === 'network') {
                // Fallback to server-side
                this.onError(new Error("Network error, falling back to server"));
            } else {
                this.onError(new Error(event.error));
            }
        };

        this.recognition.onend = () => {
            // Auto-restart если не было явной остановки
            if (this.config.autoRestart) {
                setTimeout(() => this.start(), 100);
            }
        };
    }

    async start(): Promise<void> {
        this.recognition.start();
    }

    async stop(): Promise<void> {
        this.recognition.stop();
    }

    onResult: (result: RecognitionResult) => void = () => {};
    onError: (error: Error) => void = () => {};
}

// Серверная реализация (Whisper)
class WhisperEngine implements RecognitionEngine {
    name = "Whisper API";
    private mediaRecorder: MediaRecorder | null = null;
    private audioChunks: Blob[] = [];

    constructor(private config: RecognitionConfig) {}

    async initialize(): Promise<void> {
        // Проверка доступности API
        const response = await fetch('/api/v1/voice/check-availability');
        if (!response.ok) {
            throw new Error("Whisper API not available");
        }
    }

    async start(): Promise<void> {
        const stream = await navigator.mediaDevices.getUserMedia({ audio: true });

        this.mediaRecorder = new MediaRecorder(stream, {
            mimeType: 'audio/webm;codecs=opus'
        });

        this.mediaRecorder.ondataavailable = (event) => {
            if (event.data.size > 0) {
                this.audioChunks.push(event.data);
            }
        };

        this.mediaRecorder.onstop = async () => {
            await this.sendAudioForRecognition();
        };

        // Отправка каждые 2 секунды для потокового распознавания
        this.mediaRecorder.start(2000);
    }

    private async sendAudioForRecognition(): Promise<void> {
        const audioBlob = new Blob(this.audioChunks, { type: 'audio/webm' });
        this.audioChunks = [];

        const formData = new FormData();
        formData.append('audio', audioBlob, 'recording.webm');
        formData.append('language', this.config.language || 'ru');

        try {
            const response = await fetch('/api/v1/voice/recognize', {
                method: 'POST',
                body: formData
            });

            const result = await response.json();

            this.onResult({
                text: result.text,
                confidence: result.confidence,
                isFinal: true,
                language: result.language,
                timestamp: Date.now()
            });
        } catch (error) {
            this.onError(error as Error);
        }
    }

    async stop(): Promise<void> {
        if (this.mediaRecorder) {
            this.mediaRecorder.stop();
        }
    }

    onResult: (result: RecognitionResult) => void = () => {};
    onError: (error: Error) => void = () => {};
}
```

1.3 Синтез речи (Speech Synthesis)

Требования к качеству:

- Естественная интонация (не роботизированная речь)
- Эмоциональная окраска (радость, сочувствие, нейтральность)
- Паузы в правильных местах
- Ударения и произношение
- Скорость речи регулируемая (0.5x - 2x)

Технологии:

Tier 1 (Лучшее качество):

- ElevenLabs API (самая естественная речь, поддержка эмоций)
- Azure Neural TTS (высокое качество, много голосов)
- Google Cloud Text-to-Speech (WaveNet голоса)

Tier 2 (Хорошее качество):

- Amazon Polly (Neural voices)
- Яндекс SpeechKit (отлично для русского)
- Web Speech API (браузерный fallback)

Cursor AI Prompt для реализации:

```
# Task 1.2: Implement Natural Speech Synthesis

Create advanced TTS system with emotional intelligence:

## File: widget/voice/js/speech-synthesis.ts

Implement NaturalSpeechSynthesizer class with:

1. Multi-Provider Support
   - ElevenLabs API integration (primary, best quality)
   - Azure Neural TTS (fallback 1)
   - Google WaveNet (fallback 2)
   - Web Speech API (browser fallback)
   - Provider auto-selection based on:
     * Language (some providers better for specific languages)
     * Availability (API quota, rate limits)
     * Cost (budget constraints)
     * Latency (user preference)

2. Emotional Intelligence
   - Analyze text sentiment to select voice emotion:
     * Neutral (default)
     * Happy (positive responses, success messages)
     * Empathetic (error messages, apologies)
     * Professional (formal responses)
   - Use GPT-4 for emotion detection if needed
   - SSML markup for emphasis and pauses
   - Prosody control (pitch, rate, volume)

3. Natural Speech Characteristics
   - Add filler words occasionally (ну, э-э, так)
   - Insert breathing sounds for realism
   - Vary speech rate within sentence
   - Add micro-pauses at commas
   - Emphasize important words
   - Use rising/falling intonation

4. Voice Selection & Customization
   - Multiple voice options per language
   - Gender selection (male/female/neutral)
   - Age selection (young/adult/senior)
   - Style selection (news, conversation, customer service)
   - Voice cloning from user samples (ElevenLabs)
   - Persistent voice preference per user

5. Audio Processing & Enhancement
   - Normalize volume levels
   - Apply light compression for clarity
   - Add subtle reverb for warmth
   - Remove clicks and pops
   - Enhance clarity with EQ
   - Streaming playback (start playing while generating)

6. Caching & Performance
   - Cache generated audio (IndexedDB)
   - Cache key: hash(text + voice + settings)
   - TTL: 7 days for common phrases
   - Preload common responses
   - Background generation for predicted responses
   - Audio compression (Opus codec)

7. Queue Management
   - Handle multiple TTS requests
   - Interrupt current speech on new request
   - Pause/resume functionality
   - Adjust playback speed dynamically
   - Skip to next sentence
   - Repeat last utterance

8. Accessibility Features
   - Visual subtitles synchronized with audio
   - Highlight current word being spoken
   - Adjustable speech rate (0.5x to 2x)
   - Pitch adjustment for hearing impaired
   - Option to disable animations during speech

9. Error Handling & Graceful Degradation
   - Fallback chain: ElevenLabs → Azure → Google → Browser
   - Retry logic with exponential backoff
   - Network timeout handling (10 sec max)
   - Partial audio playback if generation interrupted
   - Clear error messages to user

Implementation requirements:
- TypeScript with comprehensive types
- Audio Web API for playback control
- Unit tests (90%+ coverage)
- Performance tests (latency < 500ms)
- A/B testing framework for voice comparison
- Telemetry for quality monitoring

Generate:
- Complete TypeScript implementation
- Provider adapters for each TTS service
- Audio processing pipeline
- Comprehensive test suite
- Performance benchmarks
- User preference UI components
```

Ожидаемая реализация:

```
// widget/voice/js/speech-synthesis.ts

interface TTSProvider {
    name: string;
    initialize(): Promise<void>;
    synthesize(text: string, options: SynthesisOptions): Promise<AudioBuffer>;
    getVoices(): Promise<Voice[]>;
    estimateCost(text: string): number;
}

interface SynthesisOptions {
    voice: string;
    language: string;
    emotion?: 'neutral' | 'happy' | 'empathetic' | 'professional';
    speed: number; // 0.5 - 2.0
    pitch: number; // 0.5 - 2.0
    volume: number; // 0.0 - 1.0
}

class NaturalSpeechSynthesizer {
    private providers: Map<string, TTSProvider> = new Map();
    private audioContext: AudioContext;
    private currentAudio: AudioBufferSourceNode | null = null;
    private cache: TTSCache;
    private queue: TTSRequest[] = [];

    constructor() {
        this.audioContext = new AudioContext();
        this.cache = new TTSCache();
        this.initializeProviders();
    }

    private async initializeProviders(): Promise<void> {
        // ElevenLabs (лучшее качество)
        try {
            const elevenlabs = new ElevenLabsProvider();
            await elevenlabs.initialize();
            this.providers.set('elevenlabs', elevenlabs);
        } catch (error) {
            console.warn("ElevenLabs not available:", error);
        }

        // Azure Neural TTS
        try {
            const azure = new AzureTTSProvider();
            await azure.initialize();
            this.providers.set('azure', azure);
        } catch (error) {
            console.warn("Azure TTS not available:", error);
        }
        // Browser fallback (всегда доступен)
        const browserTTS = new BrowserTTSProvider();
        await browserTTS.initialize();
        this.providers.set('browser', browserTTS);
    }

    async speak(text: string, options: Partial<SynthesisOptions> = {}): Promise<void> {
        // Обогащение текста для естественности
        const enrichedText = this.enrichText(text);

        // Определение эмоции из контекста
        const emotion = options.emotion || await this.detectEmotion(text);

        const fullOptions: SynthesisOptions = {
            voice: options.voice || 'default',
            language: options.language || 'ru-RU',
            emotion: emotion,
            speed: options.speed || 1.0,
            pitch: options.pitch || 1.0,
            volume: options.volume || 1.0
        };

        // Проверка кеша
        const cacheKey = this.cache.generateKey(enrichedText, fullOptions);
        let audioBuffer = await this.cache.get(cacheKey);

        if (!audioBuffer) {
            // Генерация речи
            audioBuffer = await this.synthesizeWithFallback(enrichedText, fullOptions);

            // Сохранение в кеш
            await this.cache.set(cacheKey, audioBuffer);
        }

        // Применение аудио эффектов
        audioBuffer = await this.applyAudioEffects(audioBuffer, fullOptions);

        // Воспроизведение
        await this.playAudio(audioBuffer);
    }

    private enrichText(text: string): string {
        // Добавление SSML разметки для естественности
        let enriched = text;

        // Паузы после знаков препинания
        enriched = enriched.replace(/,/g, '<break time="200ms"/>,');
        enriched = enriched.replace(/\./g, '<break time="400ms"/>.');
        enriched = enriched.replace(/!/g, '<break time="400ms"/>!');
        enriched = enriched.replace(/\?/g, '<break time="400ms"/>?');

        // Эмфазис для важных слов (заглавные, выделенные)
        enriched = enriched.replace(/\*\*(.+?)\*\*/g, '<emphasis level="strong">$1</empha

        // Числа прописью для лучшего произношения
        enriched = this.numbersToWords(enriched);

        return `<speak>${enriched}</speak>`;
    }

    private async detectEmotion(text: string): Promise<string> {
        // Простая эвристика на основе ключевых слов
        const lowerText = text.toLowerCase();

        if (lowerText.match(/извините|сожалею|ошибка|проблема/)) {
            return 'empathetic';
        }

        if (lowerText.match(/отлично|здорово|успешно|готово/)) {
            return 'happy';
        }

        if (lowerText.match(/согласно|следует|необходимо|требуется/)) {
            return 'professional';
        }

        return 'neutral';
    }

    private async synthesizeWithFallback(
        text: string,
        options: SynthesisOptions
    ): Promise<AudioBuffer> {
        const providerPriority = ['elevenlabs', 'azure', 'browser'];

        for (const providerName of providerPriority) {
            const provider = this.providers.get(providerName);
            if (!provider) continue;

            try {
                console.log(`Trying TTS provider: ${providerName}`);
                return await provider.synthesize(text, options);
            } catch (error) {
                console.warn(`Provider ${providerName} failed:`, error);
            }
        }

        throw new Error("All TTS providers failed");
    }

    private async applyAudioEffects(
        buffer: AudioBuffer,
        options: SynthesisOptions
    ): Promise<AudioBuffer> {
        const offlineContext = new OfflineAudioContext(
            buffer.numberOfChannels,
            buffer.length,
            buffer.sampleRate
        );

        const source = offlineContext.createBufferSource();
        source.buffer = buffer;

        // Компрессор для ровности громкости
        const compressor = offlineContext.createDynamicsCompressor();
        compressor.threshold.value = -24;
        compressor.knee.value = 30;
        compressor.ratio.value = 12;
        compressor.attack.value = 0.003;
        compressor.release.value = 0.25;

        // Небольшой reverb для "теплоты"
        const convolver = offlineContext.createConvolver();
        convolver.buffer = await this.createReverbImpulse();

        const reverbGain = offlineContext.createGain();
        reverbGain.gain.value = 0.1; // Едва заметный reverb

        // EQ для четкости речи
        const lowShelf = offlineContext.createBiquadFilter();
        lowShelf.type = 'lowshelf';
        lowShelf.frequency.value = 200;
        lowShelf.gain.value = -3; // Убрать мутность

        const highShelf = offlineContext.createBiquadFilter();
        highShelf.type = 'highshelf';
        highShelf.frequency.value = 3000;
        highShelf.gain.value = 2; // Добавить ясности

        // Подключение цепочки эффектов
        source.connect(compressor);
        compressor.connect(lowShelf);
        lowShelf.connect(highShelf);
        highShelf.connect(offlineContext.destination);

        // Параллельный reverb
        compressor.connect(convolver);
        convolver.connect(reverbGain);
        reverbGain.connect(offlineContext.destination);

        source.start();

        return await offlineContext.startRendering();
    }

    private async playAudio(buffer: AudioBuffer): Promise<void> {
        // Остановка текущего воспроизведения
        if (this.currentAudio) {
            this.currentAudio.stop();
        }

        const source = this.audioContext.createBufferSource();
        source.buffer = buffer;
        source.connect(this.audioContext.destination);

        this.currentAudio = source;
        return new Promise((resolve) => {
            source.onended = () => {
                this.currentAudio = null;
                resolve();
            };
            source.start();
        });
    }

    stop(): void {
        if (this.currentAudio) {
            this.currentAudio.stop();
            this.currentAudio = null;
        }
    }
}

// ElevenLabs Provider
class ElevenLabsProvider implements TTSProvider {
    name = "ElevenLabs";
    private apiKey: string;
    private baseUrl = "https://api.elevenlabs.io/v1";

    async initialize(): Promise<void> {
        const response = await fetch('/api/v1/voice/elevenlabs-config');
        const config = await response.json();
        this.apiKey = config.apiKey;
    }

    async synthesize(text: string, options: SynthesisOptions): Promise<AudioBuffer> {
        const voiceId = this.mapVoiceId(options.voice);

        const response = await fetch(`${this.baseUrl}/text-to-speech/${voiceId}`, {
            method: 'POST',
            headers: {
                'Accept': 'audio/mpeg',
                'Content-Type': 'application/json',
                'xi-api-key': this.apiKey
            },
            body: JSON.stringify({
                text: text,
                model_id: "eleven_multilingual_v2",
                voice_settings: {
                    stability: 0.5,
                    similarity_boost: 0.75,
                    style: this.mapEmotionToStyle(options.emotion),
                    use_speaker_boost: true
                }
            })
        });

        if (!response.ok) {
            throw new Error(`ElevenLabs API error: ${response.statusText}`);
        }

        const arrayBuffer = await response.arrayBuffer();
        return await this.decodeAudio(arrayBuffer);
    }

    private mapEmotionToStyle(emotion?: string): number {
        switch (emotion) {
            case 'happy': return 0.8;
            case 'empathetic': return 0.3;
            case 'professional': return 0.5;
            default: return 0.5;
        }
    }

    private async decodeAudio(arrayBuffer: ArrayBuffer): Promise<AudioBuffer> {
        const audioContext = new AudioContext();
        return await audioContext.decodeAudioData(arrayBuffer);
    }

    async getVoices(): Promise<Voice[]> {
        const response = await fetch(`${this.baseUrl}/voices`, {
            headers: { 'xi-api-key': this.apiKey }
        });
        return await response.json();
    }

    estimateCost(text: string): number {
        // ElevenLabs: $0.30 per 1000 characters
        return (text.length / 1000) * 0.30;
    }

    private mapVoiceId(voice: string): string {
        // Маппинг имен на ID голосов ElevenLabs
        const voiceMap: Record<string, string> = {
            'default': '21m00Tcm4TlvDq8ikWAM', // Rachel (англ)
            'russian_male': 'pNInz6obpgDQGcFmaJgB', // Adam
            'russian_female': 'EXAVITQu4vr4xnSDxMaL' // Bella
        };
        return voiceMap[voice] || voiceMap['default'];
    }
}

// Browser TTS Provider (Fallback)
class BrowserTTSProvider implements TTSProvider {
    name = "Browser TTS";
    private synthesis: SpeechSynthesis;

    async initialize(): Promise<void> {
        if (!('speechSynthesis' in window)) {
            throw new Error("Browser TTS not supported");
        }
        this.synthesis = window.speechSynthesis;
    }

    async synthesize(text: string, options: SynthesisOptions): Promise<AudioBuffer> {
        return new Promise((resolve, reject) => {
            const utterance = new SpeechSynthesisUtterance(text);
            utterance.voice = this.selectVoice(options.language);
            utterance.rate = options.speed;
            utterance.pitch = options.pitch;
            utterance.volume = options.volume;

            utterance.onend = () => {
                // Браузерный TTS не возвращает AudioBuffer напрямую
                // Нужно захватить через MediaRecorder
                resolve(this.captureAudio(utterance));
            };

            utterance.onerror = (error) => {
                reject(error);
            };

            this.synthesis.speak(utterance);
        });
    }

    private selectVoice(language: string): SpeechSynthesisVoice | null {
        const voices = this.synthesis.getVoices();
        return voices.find(v => v.lang === language) || voices[0];
    }

    private async captureAudio(utterance: SpeechSynthesisUtterance): Promise<AudioBuffer>
        // Simplified - в реальности нужно использовать MediaRecorder
        // для захвата audio output
        const audioContext = new AudioContext();
        const buffer = audioContext.createBuffer(1, 44100, 44100);
        return buffer;
    }

    async getVoices(): Promise<Voice[]> {
        return this.synthesis.getVoices().map(v => ({
            id: v.voiceURI,
            name: v.name,
            language: v.lang
        }));
    }

    estimateCost(text: string): number {
        return 0; // Free
    }
}
```

1.4 Визуализация и анимация

Cursor AI Prompt:

```
# Task 1.3: Implement Voice Widget Visualizations

Create engaging voice interface with animations:

## File: widget/voice/js/visualizer.ts
```

```
1. Wave Animation (Speaking Indicator)
   - Canvas-based real-time audio visualization
   - Analyze audio frequency data using AnalyserNode
   - Display animated waveform bars
   - Colors: gradient from primary to accent color
   - Smooth transitions between states
   - Responsive to actual audio amplitude

2. Microphone Listening State
   - Pulsing microphone icon
   - Circular progress indicator
   - "Listening..." text animation
   - Sound wave ripples emanating from mic
   - Color changes based on confidence

3. Processing State ("Думаю...")
   - Animated thinking bubbles
   - Rotating dots loader
   - Progress bar for long operations
   - Estimated time remaining

4. Speaking State
   - Animated avatar mouth sync
   - Subtitle display synchronized with speech
   - Word highlighting as spoken
   - Volume meter for playback

5. Error State
   - Clear error icon animation
   - Error message with retry button
   - Shake animation for attention
   - Auto-dismiss after 5 seconds

6. Idle State
   - Gentle breathing animation
   - Floating particles background
   - Call-to-action pulse

Implementation:
- Use Canvas API or SVG for animations
- 60 FPS target for smooth animations
- GPU acceleration where possible
- Accessibility: prefers-reduced-motion support
- Customizable theme colors
- Mobile-optimized (touch events)
```

2. Архитектура системы

2.1 Компонентная диаграмма

```
┌─────────────────────────────────────────────────────────────────┐
│                     Browser / Client                            │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │           Voice Widget UI (HTML/CSS/JS)                   │  │
│  │  ┌─────────────┐  ┌─────────────┐  ┌───────────────────┐  │  │
│  │  │ Speech      │  │ Dialog      │  │ Navigation        │  │  │
│  │  │Recognition  │  │ Manager     │  │ Controller        │  │  │
│  │  └──────┬──────┘  └──────┬──────┘  └─────────┬─────────┘  │  │
│  │         │                │                   │            │  │
│  │  ┌──────▼────────────────▼───────────────────▼─────────┐  │  │
│  │  │         WebSocket Connection Manager                │  │  │
│  │  └──────────────────────────┬──────────────────────────┘  │  │
│  └─────────────────────────────┼─────────────────────────────┘  │
└────────────────────────────────┼────────────────────────────────┘
                                 │
                                 │ WSS (Secure WebSocket)
                                 │
┌────────────────────────────────▼────────────────────────────────┐
│                   Backend Server (Python)                       │
│                                                                 │
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐  │
│  │ Voice API       │  │ Dialog API      │  │ Navigation API  │  │
│  │ /api/v1/voice   │  │ /api/v1/dialog  │  │ /api/v1/navigate│  │
│  └────────┬────────┘  └────────┬────────┘  └────────┬────────┘  │
│           │                    │                    │           │
│  ┌────────▼────────────────────▼────────────────────▼────────┐  │
│  │          Voice Assistant Orchestrator                     │  │
│  │  - Intent Recognition                                     │  │
│  │  - Context Management                                     │  │
│  │  - Response Generation                                    │  │
│  └────────┬────────────┬─────────────┬───────────────┬───────┘  │
│           │            │             │               │          │
│  ┌────────▼───┐ ┌──────▼────┐ ┌──────▼─────┐ ┌───────▼───────┐  │
│  │Whisper     │ │LLM        │ │Knowledge   │ │Navigation     │  │
│  │ STT        │ │(Ollama)   │ │Retrieval   │ │Controller     │  │
│  └────────────┘ └───────────┘ └────────────┘ └───────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │              ElevenLabs / Azure TTS                       │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
```

2.2 Поток данных в диалоге

```
User speaks → Microphone → Audio Stream
                                │
                                ▼
                          Speech Recognition
                          (Browser or Whisper)
                                │
                                ▼
```

```
                        Transcribed Text
                              │
                              ▼
                        Dialog Manager
                              │
                              ├──→ Intent Recognition (что хочет пользователь?)
                              ├──→ Context Update (обновление состояния диалога)
                              ├──→ Entity Extraction (извлечение параметров)
                              │
                              ▼
                        Decision Point
                              │
        ┌─────────────────────┼─────────────────────┐
        │                     │                     │
        ▼                     ▼                     ▼
  Knowledge Query      Navigation Action      Clarification
  (RAG Search)         (Page Scroll/Click)    (Ask Question)
        │                     │                     │
        └─────────────────────┼─────────────────────┘
                              │
                              ▼
                        LLM Response Generation
                              │
                              ▼
                        Text Response
                              │
                              ├──→ Emotion Detection
                              ├──→ SSML Markup
                              │
                              ▼
                        Speech Synthesis
                        (ElevenLabs/Azure)
                              │
                              ▼
                        Audio Stream
                              │
                              ▼
                        Speaker → User hears
```

2.3 Backend API Endpoints

Cursor AI Prompt для создания API:

```
# Task 2.1: Implement Voice Assistant Backend API

Create comprehensive voice assistant API:

## File: api.py (add new router)

Add VoiceRouter with endpoints:

### POST /api/v1/voice/session/start
Start new voice conversation session

Request:
```

```
{
  "project": "string",
  "user_id": "string",
  "language": "ru-RU",
  "voice_preference": {
    "provider": "elevenlabs",
    "voice_id": "default",
    "speed": 1.0
  }
}

Response:
{
  "session_id": "uuid",
  "websocket_url": "wss://api.example.com/voice/ws/{session_id}",
  "expires_at": "2025-11-16T12:00:00Z"
}

### WebSocket /api/v1/voice/ws/{session_id}
Real-time bidirectional communication

Client → Server messages:
{
  "type": "audio_chunk",
  "data": "base64_audio",
  "sequence": 0,
  "is_final": false
}

{
  "type": "text_input",
  "text": "Как настроить проект?",
  "timestamp": "2025-11-16T11:00:00Z"
}

{
  "type": "navigation_command",
  "action": "scroll",
  "params": {"direction": "down", "amount": 100}
}

Server → Client messages:
{
  "type": "transcription",
  "text": "Как настроить проект",
  "is_final": false,
  "confidence": 0.85
}

{
  "type": "response",
  "text": "Для настройки проекта выполните следующие шаги...",
  "audio_url": "/api/v1/voice/audio/{audio_id}",
  "sources": [...],
  "suggested_actions": [
    {"type": "navigate", "url": "/docs/setup"}
  ]
}

{
  "type": "status",
  "status": "thinking" | "speaking" | "listening" | "error",
  "message": "Обрабатываю запрос..."
}

### POST /api/v1/voice/recognize
Synchronous speech recognition (fallback)

Request: multipart/form-data
- audio: audio file (WAV/WebM)
- language: "ru-RU"
- provider: "whisper" | "vosk"

Response:
{
  "text": "Recognized text",
  "confidence": 0.92,
  "language": "ru-RU",
  "processing_time_ms": 245
}

### POST /api/v1/voice/synthesize
Text-to-speech endpoint

Request:
{
  "text": "Text to synthesize",
  "voice": "default",
  "language": "ru-RU",
  "emotion": "neutral",
  "options": {
    "speed": 1.0,
    "pitch": 1.0
  }
}

Response: audio/mpeg stream or
{
  "audio_url": "/api/v1/voice/audio/{audio_id}",
  "duration_seconds": 5.2,
  "cached": true
}

### GET /api/v1/voice/audio/{audio_id}
Retrieve generated audio

Response: audio/mpeg binary stream
Cache-Control: max-age=86400

### POST /api/v1/dialog/intent
Analyze user intent
```

```
Request:
{
  "text": "Покажи мне раздел с ценами",
  "context": {
    "current_page": "/",
    "previous_intents": ["greeting", "navigation"]
  }
}

Response:
{
  "intent": "navigate",
  "confidence": 0.95,
  "entities": {
    "target_section": "pricing"
  },
  "suggested_action": {
    "type": "navigate",
    "url": "/pricing",
    "scroll_to": "#pricing-table"
  }
}

### POST /api/v1/navigate/execute
Execute navigation action

Request:
{
  "session_id": "uuid",
  "action": "scroll" | "click" | "navigate",
  "params": {
    "selector": "#pricing-table",
    "behavior": "smooth"
  }
}

Response:
{
  "success": true,
  "current_url": "/pricing",
  "visible_elements": ["#pricing-table", "button.cta"]
}

Implementation requirements:
- WebSocket support with FastAPI (starlette)
- AsyncIO for concurrent handling
- Session management with Redis
- Rate limiting per user
- Request/response validation with Pydantic
- Comprehensive error handling
- Metrics and logging
- Unit and integration tests (90%+ coverage)

Generate:
- Complete API implementation
- Pydantic models for all requests/responses
- WebSocket handler with reconnection logic
- Test suite with mock clients
- API documentation (OpenAPI)
```

Реализация:

```
# api.py - Voice Router

from fastapi import APIRouter, WebSocket, WebSocketDisconnect, UploadFile, File
from typing import Dict, Optional
import uuid
import asyncio
from datetime import datetime, timedelta
import structlog

logger = structlog.get_logger()

voice_router = APIRouter(prefix="/api/v1/voice", tags=["voice"])

# Активные WebSocket сессии
active_sessions: Dict[str, VoiceSession] = {}

class VoiceSession:
    """Управление голосовой сессией."""

    def __init__(self, session_id: str, project: str, user_id: str, config: dict):
        self.session_id = session_id
        self.project = project
        self.user_id = user_id
        self.config = config
        self.created_at = datetime.now()
        self.expires_at = self.created_at + timedelta(hours=1)
        self.context = DialogContext()
        self.speech_recognizer = SpeechRecognizer(config)
        self.speech_synthesizer = SpeechSynthesizer(config)
        self.dialog_manager = DialogManager(project)

    async def process_audio_chunk(self, audio_data: bytes) -> Optional[str]:
        """Обработка аудио чанка."""
        result = await self.speech_recognizer.process_chunk(audio_data)
        if result and result.is_final:
            return result.text
        return None

    async def process_text_input(self, text: str) -> dict:
        """Обработка текстового ввода."""
        # Обновление контекста
        self.context.add_user_message(text)

        # Распознавание intent
        intent = await self.dialog_manager.recognize_intent(text, self.context)

        # Генерация ответа
        if intent.type == "knowledge_query":
            response = await self.dialog_manager.answer_from_knowledge(text, self.context
        elif intent.type == "navigation":
            response = await self.dialog_manager.execute_navigation(intent.params)
        else:
            response = await self.dialog_manager.generate_fallback_response(text)

        # Обновление контекста ответом
        self.context.add_assistant_message(response["text"])

        # Генерация аудио
        audio_url = await self.speech_synthesizer.synthesize(
            response["text"],
            emotion=response.get("emotion", "neutral")
        )

        return {
            "text": response["text"],
            "audio_url": audio_url,
            "sources": response.get("sources", []),
            "suggested_actions": response.get("actions", [])
        }


@voice_router.post("/session/start")
async def start_voice_session(request: VoiceSessionRequest) -> VoiceSessionResponse:
    """Начать новую голосовую сессию."""
    session_id = str(uuid.uuid4())

    # Создание сессии
    session = VoiceSession(
        session_id=session_id,
        project=request.project,
        user_id=request.user_id or "anonymous",
        config=request.voice_preference or {}
    )

    active_sessions[session_id] = session

    logger.info("voice_session_started", session_id=session_id, project=request.project)

    # URL для WebSocket подключения
    websocket_url = f"wss://{request.host}/api/v1/voice/ws/{session_id}"

    return VoiceSessionResponse(
        session_id=session_id,
        websocket_url=websocket_url,
        expires_at=session.expires_at
    )


@voice_router.websocket("/ws/{session_id}")
async def voice_websocket(websocket: WebSocket, session_id: str):
    """WebSocket для real-time голосового взаимодействия."""
    await websocket.accept()

    session = active_sessions.get(session_id)
    if not session:
        await websocket.send_json({
            "type": "error",
            "message": "Invalid session_id"
        })
        await websocket.close()
        return

    logger.info("websocket_connected", session_id=session_id)

    try:
        while True:
            # Получение сообщения от клиента
            data = await websocket.receive_json()
            message_type = data.get("type")

            if message_type == "audio_chunk":
                # Обработка аудио чанка
                audio_data = base64.b64decode(data["data"])
                text = await session.process_audio_chunk(audio_data)

                if text:
                    # Отправка транскрипции
                    await websocket.send_json({
                        "type": "transcription",
                        "text": text,
                        "is_final": True,
                        "confidence": 0.9
                    })

                    # Обработка распознанного текста
                    await process_user_input(websocket, session, text)

            elif message_type == "text_input":
                # Текстовый ввод
                text = data["text"]
                await process_user_input(websocket, session, text)

            elif message_type == "navigation_command":
                # Команда навигации
                action = data["action"]
                params = data.get("params", {})
                result = await session.dialog_manager.execute_navigation({
                    "action": action,
                    **params
                })

                await websocket.send_json({
                    "type": "navigation_result",
                    **result
                })

            elif message_type == "ping":
                # Keepalive
                await websocket.send_json({"type": "pong"})

    except WebSocketDisconnect:
```

```python
        logger.info("websocket_disconnected", session_id=session_id)
        # Cleanup session after 5 minutes of inactivity
        await asyncio.sleep(300)
        if session_id in active_sessions:
            del active_sessions[session_id]

    except Exception as e:
        logger.error("websocket_error", session_id=session_id, error=str(e))
        await websocket.send_json({
            "type": "error",
            "message": "Internal server error"
        })
        await websocket.close()


async def process_user_input(websocket: WebSocket, session: VoiceSession, text: str):
    """Обработка пользовательского ввода и генерация ответа."""
    # Отправка статуса "думаю"
    await websocket.send_json({
        "type": "status",
        "status": "thinking",
        "message": "Обрабатываю запрос..."
    })

    # Обработка текста и генерация ответа
    response = await session.process_text_input(text)

    # Отправка статуса "говорю"
    await websocket.send_json({
        "type": "status",
        "status": "speaking"
    })

    # Отправка ответа
    await websocket.send_json({
        "type": "response",
        **response
    })

    # Возврат к статусу "слушаю"
    await websocket.send_json({
        "type": "status",
        "status": "listening",
        "message": "Слушаю вас..."
    })


@voice_router.post("/recognize")
async def recognize_speech(
    audio: UploadFile = File(...),
    language: str = "ru-RU",
    provider: str = "whisper"
) -> RecognitionResponse:
    """Синхронное распознавание речи (fallback)."""
    start_time = time.time()
    # Чтение аудио файла
    audio_data = await audio.read()

    # Выбор провайдера распознавания
    if provider == "whisper":
        recognizer = WhisperRecognizer()
    elif provider == "vosk":
        recognizer = VoskRecognizer()
    else:
        raise HTTPException(400, "Invalid provider")

    # Распознавание
    result = await recognizer.recognize(audio_data, language)

    processing_time = (time.time() - start_time) * 1000

    logger.info(
        "speech_recognized",
        provider=provider,
        language=language,
        text_length=len(result.text),
        confidence=result.confidence,
        processing_time_ms=processing_time
    )

    return RecognitionResponse(
        text=result.text,
        confidence=result.confidence,
        language=language,
        processing_time_ms=processing_time
    )


@voice_router.post("/synthesize")
async def synthesize_speech(request: SynthesisRequest) -> SynthesisResponse:
    """Генерация речи из текста."""
    # Проверка кеша
    cache_key = f"tts:{hash(request.text)}:{request.voice}:{request.language}"
    cached_audio = await redis_client.get(cache_key)

    if cached_audio:
        return SynthesisResponse(
            audio_url=cached_audio.decode(),
            duration_seconds=0,  # TODO: store duration
            cached=True
        )

    # Выбор TTS провайдера
    synthesizer = SpeechSynthesizer(request.dict())

    # Генерация аудио
    audio_id = str(uuid.uuid4())
    audio_url = await synthesizer.synthesize_and_store(
        text=request.text,
        audio_id=audio_id
    )
    # Кеширование на 7 дней
    await redis_client.setex(cache_key, 604800, audio_url.encode())

    return SynthesisResponse(
        audio_url=audio_url,
        duration_seconds=0,  # TODO: calculate actual duration
        cached=False
    )


@voice_router.get("/audio/{audio_id}")
async def get_audio(audio_id: str):
    """Получить сгенерированное аудио."""
    # Проверка существования файла
    audio_path = f"data/voice_cache/{audio_id}.mp3"

    if not os.path.exists(audio_path):
        raise HTTPException(404, "Audio not found")

    return FileResponse(
        audio_path,
        media_type="audio/mpeg",
        headers={
            "Cache-Control": "public, max-age=86400",
            "Content-Disposition": f"inline; filename={audio_id}.mp3"
        }
    )
```

[Продолжение следует - из-за ограничения длины ответа, разбиваю на части]

Хотите, чтобы я продолжил с:

1. Диалоговым менеджером и intent recognition
2. Автоматической навигацией по сайту
3. Тестированием через Cursor.ai
4. Deployment и production setup?

Техническое задание для Cursor.ai: Голосовой ассистент SiteLLM Vertebro

Метаданные проекта

```
Проект: SiteLLM Vertebro Voice Assistant
Версия: 2.0.0
Дата создания: 2025-11-16
Репозиторий: https://github.com/rtfdeamon/sitellm_vertebro
Ветка: voice-assistant-feature
Ответственный: Development Team
```

```
Приоритет: HIGH
Срок: 10 недель
```

PHASE 1: INFRASTRUCTURE SETUP (Week 1)

Task 1.1: Project Structure Setup

Priority: P0 | **Complexity:** Low | **Time:** 2 hours

```
@cursor Create voice assistant module structure

Directory structure to create:
```

```
widget/voice/
├── README.md
├── package.json
├── tsconfig.json
├── webpack.config.js
├── src/
│  ├── index.ts (main entry)
│  ├── core/
│  │  ├── VoiceWidget.ts
│  │  ├── WebSocketManager.ts
│  │  └── StateManager.ts
│  ├── recognition/
│  │  ├── SpeechRecognitionManager.ts
│  │  ├── WebSpeechEngine.ts
│  │  ├── WhisperEngine.ts
│  │  └── VoskEngine.ts
│  ├── synthesis/
│  │  ├── SpeechSynthesizer.ts
│  │  ├── ElevenLabsProvider.ts
│  │  ├── AzureTTSProvider.ts
│  │  └── BrowserTTSProvider.ts
│  ├── dialog/
│  │  ├── DialogManager.ts
│  │  ├── IntentRecognizer.ts
│  │  └── ContextManager.ts
│  ├── navigation/
│  │  ├── NavigationController.ts
│  │  └── DOMNavigator.ts
│  ├── ui/
│  │  ├── components/
│  │  │  ├── VoiceButton.ts
│  │  │  ├── Visualizer.ts
│ │ │ ├── ChatHistory.ts
│ │ │ └── StatusIndicator.ts
│ │ └── styles/
│ │ ├── main.css
│ │ ├── animations.css
│ │ └── themes.css
│ ├── utils/
│ │ ├── AudioProcessor.ts
│ │ ├── Logger.ts
│ │ └── Cache.ts
│ └── types/
│ ├── index.ts
│ ├── recognition.ts
│ ├── synthesis.ts
│ └── dialog.ts
├── tests/
│ ├── unit/
│ ├── integration/
│ └── e2e/
├── public/
│ ├── index.html
│ ├── embed.js
│ └── assets/
└── dist/ (generated)
```

```
Requirements:
- Create all directories and empty files
- Add proper .gitignore for node_modules, dist, .env
- Create package.json with dependencies:
  - TypeScript 5.3+
  - Webpack 5+
  - Jest for testing
  - ESLint + Prettier
  - @types/node, @types/jest
- Create tsconfig.json with strict mode
- Add README.md with setup instructions

Generate:
✓ Complete directory structure
✓ Configuration files with proper settings
✓ Package.json with all dependencies
✓ Initial README.md
```

Task 1.2: Backend API Router Setup

Priority: P0 | **Complexity:** Medium | **Time:** 4 hours

```
@cursor Add voice assistant API router to FastAPI application

File: api.py
Location: Add new router after existing routers

Requirements:
1. Create VoiceRouter class inheriting from APIRouter
   - Prefix: /api/v1/voice
   - Tags: ["voice", "assistant"]
   - Dependencies: authentication, rate limiting

2. Implement endpoints:
   - POST /session/start
   - WebSocket /ws/{session_id}
   - POST /recognize
   - POST /synthesize
   - GET /audio/{audio_id}
   - GET /voices/list
   - POST /dialog/intent
   - POST /navigate/execute

3. Add Pydantic models in models.py:
```

```
class VoiceSessionRequest(BaseModel):
project: str = Field(..., description="Project identifier")
user_id: Optional[str] = Field(None, description="User identifier")
language: str = Field("ru-RU", description="Language code")
voice_preference: Optional[dict] = Field(None, description="Voice settings")

class VoiceSessionResponse(BaseModel):
session_id: str
websocket_url: str
expires_at: datetime
initial_greeting: Optional[str]

class RecognitionRequest(BaseModel):
audio: UploadFile
language: str = "ru-RU"
provider: str = "whisper"

class RecognitionResponse(BaseModel):
text: str
confidence: float
language: str
processing_time_ms: float
```

```
class SynthesisRequest(BaseModel):
text: str = Field(..., max_length=5000)
voice: str = "default"
language: str = "ru-RU"
emotion: Optional[str] = Field("neutral", regex="^(neutral|happy|empathetic|professional)$")
options: Optional[dict]

class SynthesisResponse(BaseModel):
audio_url: str
duration_seconds: float
cached: bool

class IntentRequest(BaseModel):
text: str
context: dict

class IntentResponse(BaseModel):
intent: str
confidence: float
entities: dict
suggested_action: Optional[dict]
```

```
4. WebSocket message types:
```

```
class WSMessage(BaseModel):
type: str
data: Optional[dict]
timestamp: datetime = Field(default_factory=datetime.now)

class AudioChunkMessage(WSMessage):
type: Literal["audio_chunk"]
data: str # base64 encoded
sequence: int
is_final: bool

class TranscriptionMessage(WSMessage):
type: Literal["transcription"]
text: str
is_final: bool
confidence: float

class ResponseMessage(WSMessage):
type: Literal["response"]
text: str
audio_url: Optional[str]
sources: List[dict]
suggested_actions: List[dict]
```

class StatusMessage(WSMessage):
type: Literal["status"]
status: str # listening, thinking, speaking, error
message: Optional[str]

~~~
5. Session management:
- Store active sessions in Redis with TTL=1 hour
- Session structure:
  ```
  {
      "session_id": str,
      "project": str,
      "user_id": str,
      "created_at": datetime,
      "last_activity": datetime,
      "context": {
          "history": List[dict],
          "current_intent": str,
          "entities": dict
      },
      "config": dict
  }
  ```

6. Rate limiting:
- Per user: 100 requests/hour for recognize
- Per user: 50 requests/hour for synthesize
- Per IP: 10 websocket connections max
- Use slowapi for implementation

7. Error handling:
- All endpoints return proper HTTP status codes
- Error responses follow format:
  ```
  {
      "error": str,
      "message": str,
      "details": Optional[dict],
      "request_id": str
  }
  ```
- Log all errors with structlog

8. Metrics:
- Add prometheus_client counters:
  * voice_session_created
  * voice_recognition_requests
  * voice_synthesis_requests
  * voice_websocket_connections
  * voice_api_errors
- Add histograms:
  * voice_recognition_duration
  * voice_synthesis_duration
  * voice_response_time
~~~

```
Implementation steps:
1. Create voice_router.py file
2. Define all Pydantic models
3. Implement session management with Redis
4. Implement each endpoint with proper error handling
5. Add WebSocket handler with message routing
6. Add rate limiting decorators
7. Add metrics collection
8. Write unit tests for each endpoint (90%+ coverage)
9. Write integration tests for WebSocket flow
10. Add API documentation with examples

Testing checklist:
□ All endpoints return correct status codes
□ Request validation works (invalid data rejected)
□ Session creation and retrieval from Redis
□ WebSocket connects and disconnects properly
□ Rate limiting triggers correctly
□ Metrics are collected
□ Error responses are formatted correctly
□ Concurrent requests handled properly

Generate:
✓ Complete voice_router.py implementation
✓ All Pydantic models in models.py
✓ Session management utilities
✓ WebSocket handler
✓ Rate limiting setup
✓ Metrics collection
✓ Unit test suite (tests/test_voice_api.py)
✓ Integration test suite (tests/test_voice_websocket.py)
✓ API documentation (OpenAPI annotations)
```

Task 1.3: Database Schema Extensions

Priority: P0 | **Complexity:** Medium | **Time:** 3 hours

```
@cursor Extend MongoDB schema for voice assistant

File: mongo.py
Action: Add new collections and methods

New collections to add:

1. voice_sessions
```

{
"_id": ObjectId,
"session_id": str (indexed, unique),
"project": str (indexed),
"user_id": str (indexed),
"created_at": datetime,

```
"expires_at": datetime,
"last_activity": datetime,
"total_interactions": int,
"status": str, # active, expired, completed
"metadata": {
"user_agent": str,
"ip_address": str,
"language": str,
"device_type": str
}
}
```

```
Indexes:
- {session_id: 1} unique
- {project: 1, created_at: -1}
- {expires_at: 1} TTL index
- {user_id: 1, created_at: -1}

2. voice_interactions
```

```
{
"_id": ObjectId,
"session_id": str (indexed),
"project": str (indexed),
"user_id": str,
"timestamp": datetime,
"type": str, # user_message, assistant_response
"content": {
"text": str,
"audio_url": Optional[str],
"confidence": Optional[float],
"intent": Optional[str],
"entities": Optional[dict]
},
"sources": List[dict], # для RAG ответов
"processing_time_ms": float,
"provider": str, # whisper, elevenlabs, etc.
"cost": float # оценочная стоимость
}
```

```
Indexes:
- {session_id: 1, timestamp: 1}
- {project: 1, timestamp: -1}
- {timestamp: 1} TTL 30 days
- {user_id: 1, timestamp: -1}

3. voice_analytics
```

```
{
"_id": ObjectId,
"project": str (indexed),
"date": datetime (indexed),
"metrics": {
"total_sessions": int,
"total_interactions": int,
"avg_session_duration": float,
"avg_response_time_ms": float,
"recognition_accuracy": float,
"total_cost": float,
"by_language": {
"ru-RU": {...},
"en-US": {...}
},
"by_intent": {
"knowledge_query": int,
"navigation": int,
"clarification": int
},
"errors": {
"recognition_failures": int,
"synthesis_failures": int,
"timeout_errors": int
}
}
}
```

```
Indexes:
- {project: 1, date: -1} unique
- {date: 1} TTL 90 days

4. voice_audio_cache
```

```
{
"_id": ObjectId,
"audio_id": str (indexed, unique),
"text_hash": str (indexed),
"text": str,
"voice": str,
"language": str,
"provider": str,
"file_id": ObjectId, # GridFS reference
"duration_seconds": float,
"file_size_bytes": int,
"created_at": datetime,
```

"accessed_at": datetime,
"access_count": int,
"cost": float
}

```
Indexes:
- {audio_id: 1} unique
- {text_hash: 1, voice: 1, language: 1}
- {accessed_at: 1} TTL 7 days
- {access_count: -1}

Methods to add in MongoClient class:
```

Voice Sessions

async def create_voice_session(
session_id: str,
project: str,
user_id: str,
metadata: dict
) → str

async def get_voice_session(session_id: str) → Optional[dict]

async def update_voice_session_activity(session_id: str) → None

async def close_voice_session(session_id: str) → None

async def get_active_sessions(project: str) → List[dict]

async def cleanup_expired_sessions() → int

Voice Interactions

async def log_voice_interaction(
session_id: str,
project: str,
user_id: str,
interaction_type: str,
content: dict,
sources: List[dict] = None,
processing_time_ms: float = 0,
provider: str = "",
cost: float = 0
) → str

async def get_session_history(
session_id: str,

```
limit: int = 50
) → List[dict]
```

```
async def get_user_interaction_history(
user_id: str,
project: str,
days: int = 30
) → List[dict]
```

Voice Analytics

```
async def update_voice_analytics(
project: str,
date: datetime,
metrics: dict
) → None
```

```
async def get_voice_analytics(
project: str,
start_date: datetime,
end_date: datetime
) → List[dict]
```

```
async def aggregate_daily_analytics(
project: str,
date: datetime
) → dict
```

Audio Cache

```
async def cache_audio(
audio_id: str,
text: str,
voice: str,
language: str,
provider: str,
audio_data: bytes,
duration_seconds: float,
cost: float
) → str
```

```
async def get_cached_audio(
text_hash: str,
voice: str,
language: str
) → Optional[dict]
```

```
async def increment_audio_access(audio_id: str) → None
```

async def get_audio_data(audio_id: str) → Optional[bytes]

async def cleanup_old_audio_cache(days: int = 7) → int

async def get_cache_statistics() → dict

```
Implementation requirements:
1. Add all collection definitions with indexes
2. Implement all methods with proper error handling
3. Use Motor (async MongoDB driver)
4. Add transaction support where needed
5. Implement bulk operations for analytics
6. Add caching layer with Redis for hot data
7. Write comprehensive unit tests (90%+ coverage)
8. Add migration script for existing database

Testing requirements:
□ All CRUD operations work correctly
□ Indexes are created properly
□ TTL indexes expire documents
□ Transactions commit/rollback correctly
□ Bulk operations handle large datasets
□ Error handling works (connection lost, etc.)
□ Performance is acceptable (<50ms for reads)

Generate:
✓ Extended mongo.py with new collections
✓ All new methods implemented
✓ Index creation statements
✓ Migration script (scripts/migrate_voice_schema.py)
✓ Unit tests (tests/test_voice_mongo.py)
✓ Performance benchmarks
✓ Documentation for new schema
```

PHASE 2: SPEECH RECOGNITION (Week 2-3)

Task 2.1: Speech Recognition Manager Core

Priority: P0 | **Complexity:** High | **Time:** 8 hours

```
@cursor Implement multi-provider speech recognition system

File: widget/voice/src/recognition/SpeechRecognitionManager.ts

Requirements:

1. Main Manager Class
```

interface RecognitionConfig {
language: string;

```
continuous: boolean;
interimResults: boolean;
maxAlternatives: number;
silenceThreshold: number; // ms
autoRestart: boolean;
providers: ProviderConfig[];
}

interface RecognitionResult {
text: string;
confidence: number;
isFinal: boolean;
language: string;
timestamp: number;
alternatives?: Array<{text: string; confidence: number}>;
}

interface RecognitionEngine {
name: string;
isAvailable(): Promise<boolean>;
initialize(): Promise<void>;
start(): Promise<void>;
stop(): Promise<void>;
onResult: (result: RecognitionResult) => void;
onError: (error: Error) => void;
onStateChange: (state: RecognitionState) => void;
}

enum RecognitionState {
IDLE = 'idle',
INITIALIZING = 'initializing',
READY = 'ready',
LISTENING = 'listening',
PROCESSING = 'processing',
ERROR = 'error'
}

class SpeechRecognitionManager {
private config: RecognitionConfig;
private engines: RecognitionEngine[];
private currentEngine: RecognitionEngine | null;
private state: RecognitionState;
private audioContext: AudioContext | null;
private mediaStream: MediaStream | null;
private silenceTimer: number | null;
```

```
    constructor(config: RecognitionConfig);

    async initialize(): Promise<void>;
    async start(): Promise<void>;
    stop(): void;
    restart(): Promise<void>;

    private async detectCapabilities(): Promise<void>;
    private async initializeEngines(): Promise<void>;
    private async selectBestEngine(): Promise<RecognitionEngine>;
    private async setupAudioProcessing(): Promise<void>;
    private handleSilenceDetection(): void;
    private handleEngineFailover(): Promise<void>;

    // Event handlers
    onResult: (result: RecognitionResult) => void;
    onError: (error: Error) => void;
    onStateChange: (state: RecognitionState) => void;

    // Utility methods
    isListening(): boolean;
    getCurrentEngine(): string;
    getSupportedLanguages(): string[];
    getState(): RecognitionState;
```

}

```
  2. Browser Capability Detection
```

interface BrowserCapabilities {
webSpeechAPI: boolean;
getUserMedia: boolean;
audioContext: boolean;
mediaRecorder: boolean;
webSocket: boolean;
webWorker: boolean;
codecs: {
opus: boolean;
mp3: boolean;
wav: boolean;
};
}

function detectBrowserCapabilities(): BrowserCapabilities;

```
  3. Audio Processing Pipeline
```

```
class AudioProcessor {
private audioContext: AudioContext;
private sourceNode: MediaStreamAudioSourceNode;
private analyserNode: AnalyserNode;
private gainNode: GainNode;
private dynamicsNode: DynamicsCompressorNode;
```

```
    constructor(stream: MediaStream);

    setupProcessingChain(): void;
    startProcessing(): void;
    stopProcessing(): void;

    getAudioLevel(): number;
    detectSilence(threshold: number): boolean;
    getFrequencyData(): Uint8Array;

    // Noise reduction
    applyNoiseGate(threshold: number): void;
    normalizeVolume(): void;
    removeEcho(): void;
```

```
}
```

```


4. Silence Detection
```

```
class SilenceDetector {
private threshold: number; // dB
private duration: number; // ms
private callback: () => void;
private silenceStart: number | null;
```

```
    constructor(threshold: number, duration: number, callback: () => void);

    update(audioLevel: number): void;
    reset(): void;
```

```
}
```

```


5. Error Handling & Recovery
```

```
enum RecognitionErrorType {
PERMISSION_DENIED = 'permission_denied',
NO_MICROPHONE = 'no_microphone',
NETWORK_ERROR = 'network_error',
ENGINE_ERROR = 'engine_error',
```

TIMEOUT = 'timeout',
ABORTED = 'aborted'
}

class RecognitionError extends Error {
type: RecognitionErrorType;
recoverable: boolean;
details?: any;

```
    constructor(type: RecognitionErrorType, message: string, details?: any);
```

}

class ErrorRecoveryManager {
private maxRetries: number = 3;
private retryDelay: number = 1000;
private retryCount: Map<string, number>;

~~~~
    async handleError(error: RecognitionError): Promise<void>;
    ```
    private async retry<T>(fn: () => Promise<T>, errorType: string): Promise<T>;
    ```
    private getRetryDelay(attempt: number): number;
    reset(): void;
~~~~

}

```


Implementation steps:
1. Create SpeechRecognitionManager class with state management
2. Implement browser capability detection
3. Create AudioProcessor with Web Audio API
4. Implement SilenceDetector
5. Add error handling and recovery logic
6. Implement engine selection algorithm
7. Add logging with debug levels
8. Write comprehensive unit tests
9. Add integration tests with mock audio
10. Create performance benchmarks

Testing requirements:
□ Capability detection works in all browsers
□ Audio processing pipeline initializes correctly
□ Silence detection triggers appropriately
□ Error recovery attempts failover
□ State transitions are correct
□ Memory leaks are prevented (cleanup tests)
□ Performance: initialization <500ms
□ Performance: latency <100ms

Generate:
```

```
✓ Complete SpeechRecognitionManager.ts
✓ AudioProcessor.ts
✓ SilenceDetector.ts
✓ ErrorRecoveryManager.ts
✓ BrowserCapabilities.ts
✓ Unit tests (90%+ coverage)
✓ Integration tests
✓ Performance benchmarks
✓ TypeScript type definitions
✓ JSDoc documentation
```

Task 2.2: Web Speech API Engine

Priority: P0 | **Complexity:** Medium | **Time:** 4 hours

```
@cursor Implement Web Speech API recognition engine

File: widget/voice/src/recognition/WebSpeechEngine.ts

Requirements:
```

class WebSpeechEngine implements RecognitionEngine {
name = "Web Speech API";

```
private recognition: SpeechRecognition | null;
private config: RecognitionConfig;
private isInitialized: boolean = false;
private isRunning: boolean = false;
private restartTimer: number | null = null;

constructor(config: RecognitionConfig);

async isAvailable(): Promise<boolean> {
    // Check if Web Speech API is supported
    return 'webkitSpeechRecognition' in window || 'SpeechRecognition' in window;
}

async initialize(): Promise<void> {
    // Initialize SpeechRecognition
    const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;

    if (!SpeechRecognition) {
        throw new Error("Web Speech API not supported");
    }

    this.recognition = new SpeechRecognition();
    this.configureRecognition();
    this.setupEventHandlers();
    this.isInitialized = true;
}

private configureRecognition(): void {
    if (!this.recognition) return;

    this.recognition.continuous = this.config.continuous;
    this.recognition.interimResults = this.config.interimResults;
    this.recognition.lang = this.config.language;
    this.recognition.maxAlternatives = this.config.maxAlternatives || 3;
}

private setupEventHandlers(): void {
    if (!this.recognition) return;

    this.recognition.onstart = () => {
        this.isRunning = true;
        this.onStateChange(RecognitionState.LISTENING);
    };

    this.recognition.onresult = (event: SpeechRecognitionEvent) => {
        const result = event.results[event.results.length - 1];
        const transcript = result.transcript;
        const confidence = result.confidence;

        // Extract alternatives
        const alternatives: Array<{text: string; confidence: number}> = [];
        for (let i = 0; i < result.length; i++) {
            alternatives.push({
                text: result[i].transcript,
                confidence: result[i].confidence
            });
        }

        this.onResult({
            text: transcript,
            confidence: confidence || 0.9,
            isFinal: result.isFinal,
            language: this.recognition!.lang,
            timestamp: Date.now(),
            alternatives: alternatives.slice(1) // Exclude first (main result)
        });
    };

    this.recognition.onerror = (event: SpeechRecognitionErrorEvent) => {
        console.error("Web Speech API error:", event.error);

        let errorType: RecognitionErrorType;
        switch (event.error) {
            case 'no-speech':
                // Not really an error, just no speech detected
                return;
            case 'audio-capture':
                errorType = RecognitionErrorType.NO_MICROPHONE;
                break;
            case 'not-allowed':
                errorType = RecognitionErrorType.PERMISSION_DENIED;
                break;
            case 'network':
                errorType = RecognitionErrorType.NETWORK_ERROR;
                break;
            case 'aborted':
                errorType = RecognitionErrorType.ABORTED;
                break;
            default:
                errorType = RecognitionErrorType.ENGINE_ERROR;
        }

        const error = new RecognitionError(
            errorType,
            `Web Speech API error: ${event.error}`,
            { originalError: event }
        );

        this.onError(error);

        // Try to recover from network errors
        if (errorType === RecognitionErrorType.NETWORK_ERROR && this.config.autoRestart)
            this.scheduleRestart();
        }
    };

    this.recognition.onend = () => {
        this.isRunning = false;

        // Auto-restart if enabled and not explicitly stopped
        if (this.config.autoRestart && this.isInitialized) {
            this.scheduleRestart();
        } else {
            this.onStateChange(RecognitionState.IDLE);
        }
    };
}

async start(): Promise<void> {
    if (!this.recognition) {
        throw new Error("Recognition not initialized");
    }

    if (this.isRunning) {
        console.warn("Recognition already running");
        return;
    }

    try {
        this.recognition.start();
    } catch (error) {
        // Handle "recognition already started" error
        if (error.name === 'InvalidStateError') {
            this.stop();
            await new Promise(resolve => setTimeout(resolve, 100));
            this.recognition.start();
        } else {
            throw error;
        }
    }
```

```
}

stop(): void {
    if (this.restartTimer) {
        clearTimeout(this.restartTimer);
        this.restartTimer = null;
    }

    if (this.recognition && this.isRunning) {
        this.recognition.stop();
        this.isRunning = false;
    }
}

private scheduleRestart(): void {
    if (this.restartTimer) {
        return; // Already scheduled
    }

    this.restartTimer = window.setTimeout(() => {
        this.restartTimer = null;
        if (this.isInitialized) {
            this.start().catch(error => {
                console.error("Failed to restart recognition:", error);
            });
        }
    }, 100);
}

// Event callbacks (to be set by manager)
onResult: (result: RecognitionResult) => void = () => {};
onError: (error: RecognitionError) => void = () => {};
onStateChange: (state: RecognitionState) => void = () => {};
```

}

```


Testing requirements:
□ Initialization works in supported browsers
□ Recognition starts and stops correctly
□ Results are parsed correctly
□ Interim results are handled
□ Alternatives are extracted
□ Auto-restart works
□ Error handling for all error types
□ Graceful handling of InvalidStateError

Generate:
✓ Complete WebSpeechEngine.ts
✓ Unit tests with mocked SpeechRecognition API
✓ Integration tests with mock audio
✓ Browser compatibility tests
✓ Error scenario tests
✓ Documentation with examples
```

Task 2.3: Whisper API Engine

Priority: P0 | **Complexity:** High | **Time:** 6 hours

```
@cursor Implement server-side Whisper API recognition engine

Files:
- widget/voice/src/recognition/WhisperEngine.ts (frontend)
- backend/voice/whisper_recognizer.py (backend)

Frontend Implementation:
```

class WhisperEngine implements RecognitionEngine {
name = "Whisper API";

```
private config: RecognitionConfig;
private mediaRecorder: MediaRecorder | null = null;
private audioChunks: Blob[] = [];
private isRecording: boolean = false;
private apiEndpoint: string;
private websocket: WebSocket | null = null;
private useStreaming: boolean = true;

constructor(config: RecognitionConfig, apiEndpoint: string);

async isAvailable(): Promise<boolean> {
    // Check if backend API is available
    try {
        const response = await fetch(`${this.apiEndpoint}/health`);
        return response.ok;
    } catch {
        return false;
    }
}

async initialize(): Promise<void> {
    // Check API availability
    const available = await this.isAvailable();
    if (!available) {
        throw new Error("Whisper API not available");
    }

    // Setup WebSocket for streaming if supported
    if (this.useStreaming) {
        await this.setupWebSocketConnection();
    }
}

private async setupWebSocketConnection(): Promise<void> {
    const wsUrl = this.apiEndpoint.replace('http', 'ws') + '/recognize/stream';
    this.websocket = new WebSocket(wsUrl);

    this.websocket.onopen = () => {
        console.log("Whisper WebSocket connected");
    };

    this.websocket.onmessage = (event) => {
        const data = JSON.parse(event.data);

        if (data.type === 'transcription') {
            this.onResult({
                text: data.text,
                confidence: data.confidence,
                isFinal: data.is_final,
                language: data.language,
                timestamp: Date.now()
            });
        } else if (data.type === 'error') {
            this.onError(new RecognitionError(
                RecognitionErrorType.ENGINE_ERROR,
                data.message
            ));
        }
    };

    this.websocket.onerror = (error) => {
        console.error("WebSocket error:", error);
        this.useStreaming = false; // Fallback to batch mode
    };
}

async start(): Promise<void> {
    const stream = await navigator.mediaDevices.getUserMedia({
        audio: {
            echoCancellation: true,
            noiseSuppression: true,
            autoGainControl: true,
            sampleRate: 16000,
            channelCount: 1
        }
    });

    // Determine best audio format
    const mimeType = this.selectBestMimeType();

    this.mediaRecorder = new MediaRecorder(stream, {
        mimeType,
        audioBitsPerSecond: 128000
    });

    this.mediaRecorder.ondataavailable = (event) => {
        if (event.data.size > 0) {
            this.audioChunks.push(event.data);

            // Streaming mode: send immediately
            if (this.useStreaming && this.websocket?.readyState === WebSocket.OPEN) {
                this.streamAudioChunk(event.data);
            }
        }
    };

    this.mediaRecorder.onstop = async () => {
        if (!this.useStreaming) {
            // Batch mode: send all at once
            await this.sendBatchRecognition();
        }
        this.audioChunks = [];
    };

    // Capture chunks every 1 second for streaming
    this.mediaRecorder.start(1000);
    this.isRecording = true;
    this.onStateChange(RecognitionState.LISTENING);
}

private selectBestMimeType(): string {
    const types = [
        'audio/webm;codecs=opus',
        'audio/webm',
        'audio/ogg;codecs=opus',
        'audio/mp4'
    ];

    for (const type of types) {
        if (MediaRecorder.isTypeSupported(type)) {
            return type;
        }
    }

    return '';
}

private async streamAudioChunk(chunk: Blob): Promise<void> {
    if (!this.websocket || this.websocket.readyState !== WebSocket.OPEN) {
        return;
    }

    // Convert to base64 for transmission
    const reader = new FileReader();
    reader.onload = () => {
        const base64 = (reader.result as string).split(',');[^4_1]
        this.websocket!.send(JSON.stringify({
            type: 'audio_chunk',
            data: base64,
            timestamp: Date.now()
        }));
    };
    reader.readAsDataURL(chunk);
}

private async sendBatchRecognition(): Promise<void> {
    const audioBlob = new Blob(this.audioChunks, { type: 'audio/webm' });

    const formData = new FormData();
    formData.append('audio', audioBlob, 'recording.webm');
    formData.append('language', this.config.language);

    try {
        const response = await fetch(`${this.apiEndpoint}/recognize`, {
            method: 'POST',
            body: formData
        });

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

        const result = await response.json();

        this.onResult({
            text: result.text,
            confidence: result.confidence,
            isFinal: true,
            language: result.language,
            timestamp: Date.now()
        });
    } catch (error) {
        this.onError(new RecognitionError(
            RecognitionErrorType.NETWORK_ERROR,
            `Failed to recognize audio: ${error.message}`,
            { error }
        ));
    }
}

stop(): void {
    if (this.mediaRecorder && this.isRecording) {
        this.mediaRecorder.stop();
        this.mediaRecorder.stream.getTracks().forEach(track => track.stop());
        this.isRecording = false;
    }

    if (this.websocket) {
        this.websocket.close();
    }

    this.onStateChange(RecognitionState.IDLE);
}

onResult: (result: RecognitionResult) => void = () => {};
onError: (error: RecognitionError) => void = () => {};
onStateChange: (state: RecognitionState) => void = () => {};
```

}

```
Backend Implementation:
```

backend/voice/whisper_recognizer.py

```
import whisper
import torch
import asyncio
from typing import Optional, AsyncGenerator
import numpy as np
from fastapi import WebSocket
import structlog

logger = structlog.get_logger()

class WhisperRecognizer:
"""
Whisper-based speech recognition with streaming support.
"""
```

```
def __init__(self, model_name: str = "base", device: str = "auto"):
    """
    Initialize Whisper recognizer.

    Args:
        model_name: Whisper model size (tiny, base, small, medium, large)
        device: Device to run on (cuda, cpu, auto)
    """
    self.model_name = model_name

    if device == "auto":
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
    else:
        self.device = device

    logger.info("loading_whisper_model", model=model_name, device=self.device)
    self.model = whisper.load_model(model_name, device=self.device)
    logger.info("whisper_model_loaded")

async def recognize(
    self,
    audio_data: bytes,
    language: str = "ru",
    task: str = "transcribe"
) -> dict:
    """
    Recognize speech from audio data.

    Args:
        audio_data: Audio file bytes
        language: Language code (ru, en, etc.)
        task: Task type (transcribe or translate)

    Returns:
        Recognition result with text and metadata
    """
    import time
    start_time = time.time()

    # Convert audio bytes to numpy array
    audio_array = self.bytes_to_audio_array(audio_data)

    # Run recognition
    result = await asyncio.get_event_loop().run_in_executor(
        None,
        lambda: self.model.transcribe(
            audio_array,
            language=language,
            task=task,
            fp16=(self.device == "cuda")
        )
    )

    processing_time = (time.time() - start_time) * 1000

    logger.info(
        "speech_recognized",
        language=language,
        text_length=len(result["text"]),
        processing_time_ms=processing_time
    )

    return {
        "text": result["text"].strip(),
        "confidence": self.estimate_confidence(result),
        "language": result.get("language", language),
        "processing_time_ms": processing_time,
        "segments": result.get("segments", [])
    }

async def recognize_stream(
    self,
    websocket: WebSocket,
    language: str = "ru"
) -> None:
    """
    Stream recognition results via WebSocket.

    Args:
        websocket: WebSocket connection
        language: Language code
    """
    audio_buffer = bytearray()
    buffer_duration = 2.0  # seconds
    sample_rate = 16000
    buffer_size = int(buffer_duration * sample_rate * 2)  # 16-bit audio

    try:
        while True:
            message = await websocket.receive_json()

            if message["type"] == "audio_chunk":
                # Decode base64 audio
                import base64
                chunk_bytes = base64.b64decode(message["data"])
                audio_buffer.extend(chunk_bytes)

                # Process when buffer is full
                if len(audio_buffer) >= buffer_size:
                    audio_array = self.bytes_to_audio_array(bytes(audio_buffer))

                    # Run recognition
                    result = await asyncio.get_event_loop().run_in_executor(
                        None,
                        lambda: self.model.transcribe(
                            audio_array,
                            language=language,
                            task="transcribe",
                            fp16=(self.device == "cuda")
                        )
                    )

                    # Send result
                    await websocket.send_json({
                        "type": "transcription",
                        "text": result["text"].strip(),
                        "confidence": self.estimate_confidence(result),
                        "is_final": True,
                        "language": result.get("language", language)
                    })

                    # Clear buffer but keep last 0.5s for context
                    overlap_size = int(0.5 * sample_rate * 2)
                    audio_buffer = audio_buffer[-overlap_size:]

    except Exception as e:
        logger.error("stream_recognition_error", error=str(e))
        await websocket.send_json({
            "type": "error",
            "message": str(e)
        })

def bytes_to_audio_array(self, audio_bytes: bytes) -> np.ndarray:
    """Convert audio bytes to numpy array for Whisper."""
    import io
    import soundfile as sf

    # Read audio file from bytes
    audio_io = io.BytesIO(audio_bytes)
    audio, sample_rate = sf.read(audio_io)

    # Convert to mono if stereo
    if len(audio.shape) > 1:
        audio = audio.mean(axis=1)

    # Resample to 16kHz if needed
    if sample_rate != 16000:
        import scipy.signal as signal
        audio = signal.resample(
            audio,
            int(len(audio) * 16000 / sample_rate)
        )

    return audio.astype(np.float32)

def estimate_confidence(self, result: dict) -> float:
    """
    Estimate confidence from Whisper result.
    Whisper doesn't provide confidence scores directly,
    so we use heuristics based on the output.
    """
    # Use average log probability if available
    if "segments" in result and result["segments"]:
        avg_logprob = sum(s.get("avg_logprob", -1.0) for s in result["segments"]) / len(r
        # Convert log prob to confidence (roughly)
        confidence = min(1.0, max(0.0, (avg_logprob + 1.0)))
        return confidence

    return 0.9  # Default high confidence for Whisper
```

Add to voice_router.py

@voice_router.post("/recognize")
async def recognize_speech(
audio: UploadFile = File(...),
language: str = "ru",
provider: str = "whisper"
) → dict:
"""Synchronous speech recognition."""

```
# Initialize recognizer (cache this in production)
recognizer = WhisperRecognizer(model_name="base")

# Read audio file
audio_data = await audio.read()

# Recognize
result = await recognizer.recognize(audio_data, language=language)

return result
```

@voice_router.websocket("/recognize/stream")
async def stream_recognition(websocket: WebSocket):
"""Streaming speech recognition via WebSocket."""
await websocket.accept()

```
# Get language from first message
init_message = await websocket.receive_json()
language = init_message.get("language", "ru")
```

```
# Initialize recognizer
recognizer = WhisperRecognizer(model_name="base")

# Start streaming recognition
await recognizer.recognize_stream(websocket, language=language)
```

```
Implementation requirements:
1. Frontend engine with streaming and batch modes
2. Backend Whisper integration with async support
3. Audio format conversion (WebM → NumPy array)
4. WebSocket streaming protocol
5. Batch fallback for unsupported browsers
6. Proper error handling and reconnection
7. Memory-efficient audio buffering
8. GPU acceleration when available

Testing requirements:
□ Audio encoding/decoding works correctly
□ WebSocket connection established
□ Streaming recognition provides interim results
□ Batch recognition works as fallback
□ Audio buffer management prevents memory leaks
□ GPU acceleration is used when available
□ Error recovery works
□ Performance: <2s latency for 5s audio

Generate:
✓ Complete WhisperEngine.ts
✓ Complete whisper_recognizer.py
✓ WebSocket protocol documentation
✓ Unit tests for frontend engine
✓ Unit tests for backend recognizer
✓ Integration tests with real audio
✓ Performance benchmarks
✓ Deployment guide (GPU setup)
```

[Due to character limit, continuing in next response with remaining tasks...]

Shall I continue with:

- Speech Synthesis implementation
- Dialog Management & Intent Recognition
- Navigation Controller
- UI Components & Visualizations
- Testing Strategy with Cursor.ai
- Deployment & Production Setup?

⁂

Техническое задание для Cursor.ai: Дополнения к текущему SiteLLM Vertebro

Добавления к существующей структуре проекта

1. Новые директории в корне проекта

```
@cursor Create new directories for voice assistant feature

Add to existing structure:

sitellm_vertebro/
├── voice/                          # NEW: Голосовой ассистент (Python backend)
│   ├── __init__.py
│   ├── recognizer.py               # Whisper/Vosk интеграция
│   ├── synthesizer.py              # TTS провайдеры
│   ├── dialog_manager.py           # Управление диалогом
│   ├── intent_classifier.py        # Распознавание намерений
│   └── providers/
│       ├── __init__.py
│       ├── whisper.py
│       ├── vosk.py
│       ├── elevenlabs.py
│       └── azure_tts.py
│
├── widget/voice/                   # NEW: Frontend виджет
│   ├── package.json
│   ├── tsconfig.json
│   ├── webpack.config.js
│   ├── src/
│   │   ├── index.ts
│   │   ├── VoiceWidget.ts
│   │   ├── recognition/
│   │   ├── synthesis/
│   │   ├── ui/
│   │   └── types/
│   ├── public/
│   │   ├── index.html
│   │   └── embed.js
│   └── dist/
│
└── tests/voice/                    # NEW: Тесты для voice
    ├── test_recognizer.py
    ├── test_synthesizer.py
    ├── test_dialog_manager.py
    └── test_voice_api.py

Keep existing:
- admin/, api.py, app.py, mongo.py
- backend/, crawler/, retrieval/
- All other existing modules
```

2. Дополнения к существующим файлам

2.1 Добавить в `api.py`

```
@cursor Add voice router to existing api.py

File: api.py
Location: After line with other router imports (around line 30)

Add import:
```

from voice.router import voice_router

```
Location: After app.include_router() calls (around line 100)

Add router mounting:
```

Voice Assistant API

app.include_router(
voice_router,
prefix="/api/v1/voice",
tags=["voice", "assistant"]
)

```
DO NOT modify existing routers:
- llm_router
- crawler_router
- admin_router
- Keep all existing functionality intact
```

2.2 Добавить в `app.py`

```
@cursor Add voice session management to app.py lifespan

File: app.py
Location: In lifespan context manager (around line 150)

Add to startup:
```

Initialize voice assistant components

```
from voice.recognizer import WhisperRecognizer
from voice.synthesizer import TTSManager
```

Load Whisper model on startup

```
app.state.whisper = WhisperRecognizer(model_name="base")
app.state.tts_manager = TTSManager()
```

Initialize voice session cleanup task

```
app.state.voice_cleanup_task = asyncio.create_task(
cleanup_expired_voice_sessions()
)
```

```
logger.info("voice_assistant_initialized")
```

```
Add to shutdown:
```

Cleanup voice assistant

```
if hasattr(app.state, 'voice_cleanup_task'):
app.state.voice_cleanup_task.cancel()
```

```
logger.info("voice_assistant_shutdown")
```

```
Add background task function after lifespan:
```

```
async def cleanup_expired_voice_sessions():
"""Background task to cleanup expired voice sessions."""
while True:
try:
await asyncio.sleep(300) # Every 5 minutes
from mongo import MongoClient
deleted = await MongoClient.cleanup_expired_sessions()
if deleted > 0:
logger.info("voice_sessions_cleaned", count=deleted)
except asyncio.CancelledError:
break
except Exception as e:
logger.error("voice_cleanup_error", error=str(e))
```

```
DO NOT modify existing:
```

```
- MongoDB initialization
- Redis initialization
- LLM client setup
- Celery worker setup
```

2.3 Расширить `mongo.py`

```
@cursor Add voice collections to existing mongo.py

File: mongo.py
Location: Add new methods to MongoClient class (after existing methods)

Add these collections to __init__ method:
```

Voice Assistant collections (ADD to existing init)

self.voice_sessions = self.db.voice_sessions
self.voice_interactions = self.db.voice_interactions
self.voice_analytics = self.db.voice_analytics
self.voice_audio_cache = self.db.voice_audio_cache

```
Add new methods at the end of MongoClient class:
```

===================== VOICE ASSISTANT METHODS =====================

(Add after existing methods, before class end)

async def create_voice_session(
self,
session_id: str,
project: str,
user_id: str,
metadata: dict
) → str:
"""Create new voice session."""
session = {
"session_id": session_id,
"project": project,
"user_id": user_id,
"created_at": datetime.now(),
"expires_at": datetime.now() + timedelta(hours=1),
"last_activity": datetime.now(),
"total_interactions": 0,

```
"status": "active",
"metadata": metadata
}
result = await self.voice_sessions.insert_one(session)
return str(result.inserted_id)

async def get_voice_session(self, session_id: str) → Optional[dict]:
"""Get voice session by ID."""
return await self.voice_sessions.find_one({"session_id": session_id})

async def update_voice_session_activity(self, session_id: str) → None:
"""Update session last activity timestamp."""
await self.voice_sessions.update_one(
{"session_id": session_id},
{
"set" : "last_activity" : datetime.now(),"inc": {"total_interactions": 1}
}
)

async def cleanup_expired_sessions(self) → int:
"""Remove expired voice sessions."""
result = await self.voice_sessions.delete_many({
"expires_at": {"$lt": datetime.now()}
})
return result.deleted_count

async def log_voice_interaction(
self,
session_id: str,
project: str,
user_id: str,
interaction_type: str,
content: dict,
**kwargs
) → str:
"""Log voice interaction."""
interaction = {
"session_id": session_id,
"project": project,
"user_id": user_id,
"timestamp": datetime.now(),
"type": interaction_type,
"content": content,
**kwargs
}
result = await self.voice_interactions.insert_one(interaction)
return str(result.inserted_id)
```

```
async def get_session_history(
self,
session_id: str,
limit: int = 50
) → list:
"""Get interaction history for session."""
cursor = self.voice_interactions.find(
{"session_id": session_id}
).sort("timestamp", 1).limit(limit)
return await cursor.to_list(length=limit)
```

```
async def cache_audio(
self,
audio_id: str,
text: str,
voice: str,
language: str,
provider: str,
audio_data: bytes,
duration_seconds: float,
cost: float
) → str:
"""Cache synthesized audio."""
import hashlib
text_hash = hashlib.sha256(text.encode()).hexdigest()
```

```
# Store in GridFS
file_id = await self.upload_file(
    audio_data,
    filename=f"{audio_id}.mp3",
    content_type="audio/mpeg"
)

cache_entry = {
    "audio_id": audio_id,
    "text_hash": text_hash,
    "text": text,
    "voice": voice,
    "language": language,
    "provider": provider,
    "file_id": file_id,
    "duration_seconds": duration_seconds,
    "file_size_bytes": len(audio_data),
    "created_at": datetime.now(),
    "accessed_at": datetime.now(),
    "access_count": 1,
    "cost": cost
}
```

```
result = await self.voice_audio_cache.insert_one(cache_entry)
return str(result.inserted_id)
```

```
async def get_cached_audio(
self,
text_hash: str,
voice: str,
language: str
) → Optional[dict]:
"""Get cached audio by parameters."""
cache_entry = await self.voice_audio_cache.find_one({
"text_hash": text_hash,
"voice": voice,
"language": language
})
```

```
if cache_entry:
    # Update access count
    await self.voice_audio_cache.update_one(
        {"_id": cache_entry["_id"]},
        {
            "$set": {"accessed_at": datetime.now()},
            "$inc": {"access_count": 1}
        }
    )

return cache_entry
```

```
async def get_audio_data(self, audio_id: str) → Optional[bytes]:
"""Get audio data from cache."""
cache_entry = await self.voice_audio_cache.find_one({"audio_id": audio_id})
if not cache_entry:
return None
```

```
return await self.download_file(cache_entry["file_id"])
```

```
Add indexes in separate async function:
```

```
async def create_voice_indexes(self):
"""Create indexes for voice collections."""
# Voice sessions
await self.voice_sessions.create_index("session_id", unique=True)
await self.voice_sessions.create_index([("project", 1), ("created_at", -1)])
await self.voice_sessions.create_index(
"expires_at",
```

expireAfterSeconds=0 # TTL index
)

```
# Voice interactions
await self.voice_interactions.create_index([("session_id", 1), ("timestamp", 1)])
await self.voice_interactions.create_index([("project", 1), ("timestamp", -1)])
await self.voice_interactions.create_index(
    "timestamp",
    expireAfterSeconds=2592000  # 30 days
)

# Voice audio cache
await self.voice_audio_cache.create_index("audio_id", unique=True)
await self.voice_audio_cache.create_index([
    ("text_hash", 1),
    ("voice", 1),
    ("language", 1)
])
await self.voice_audio_cache.create_index(
    "accessed_at",
    expireAfterSeconds=604800  # 7 days
)

logger.info("voice_indexes_created")
```

```


DO NOT modify existing:
- Existing collection references
- Existing methods
- Existing indexes
```

2.4 Добавить в `models.py`

```
@cursor Add voice models to existing models.py

File: models.py
Location: At the end of file, after existing models

Add new models:
```

===================== VOICE ASSISTANT MODELS =====================

from typing import Literal, Optional
from datetime import datetime

class VoiceSessionRequest(BaseModel):
"""Request to start voice session."""
project: str = Field(..., description="Project identifier")

user_id: Optional[str] = Field(None, description="User identifier")
language: str = Field("ru-RU", description="Language code (BCP-47)")
voice_preference: Optional[dict] = Field(None, description="Voice settings")

```
class Config:
    json_schema_extra = {
        "example": {
            "project": "my-project",
            "user_id": "user123",
            "language": "ru-RU",
            "voice_preference": {
                "provider": "elevenlabs",
                "voice_id": "default",
                "speed": 1.0
            }
        }
    }
```

class VoiceSessionResponse(BaseModel):
"""Response with session details."""
session_id: str
websocket_url: str
expires_at: datetime
initial_greeting: Optional[str] = None

class RecognitionResult(BaseModel):
"""Speech recognition result."""
text: str
confidence: float = Field(ge=0.0, le=1.0)
language: str
processing_time_ms: float
alternatives: Optional[list] = None

class SynthesisRequest(BaseModel):
"""Text-to-speech request."""
text: str = Field(..., min_length=1, max_length=5000)
voice: str = Field("default", description="Voice identifier")
language: str = Field("ru-RU", description="Language code")
emotion: Optional[Literal["neutral", "happy", "empathetic", "professional"]] = "neutral"
options: Optional[dict] = Field(None, description="Provider-specific options")

```
class Config:
    json_schema_extra = {
        "example": {
            "text": "Здравствуйте! Чем могу помочь?",
            "voice": "default",
            "language": "ru-RU",
            "emotion": "happy",
            "options": {"speed": 1.0, "pitch": 1.0}
        }
    }
```

```
class SynthesisResponse(BaseModel):
"""TTS response with audio URL."""
audio_url: str
duration_seconds: float
cached: bool = False

class IntentRequest(BaseModel):
"""Intent recognition request."""
text: str
context: dict = Field(default_factory=dict)
project: str

class IntentResponse(BaseModel):
"""Intent recognition result."""
intent: str
confidence: float
entities: dict = Field(default_factory=dict)
suggested_action: Optional[dict] = None

class WSMessage(BaseModel):
"""Base WebSocket message."""
type: str
timestamp: datetime = Field(default_factory=datetime.now)

class AudioChunkMessage(WSMessage):
"""Audio chunk message."""
type: Literal["audio_chunk"] = "audio_chunk"
data: str = Field(..., description="Base64 encoded audio")
sequence: int
is_final: bool = False

class TranscriptionMessage(WSMessage):
"""Transcription result message."""
type: Literal["transcription"] = "transcription"
text: str
is_final: bool
confidence: float

class ResponseMessage(WSMessage):
"""Assistant response message."""
type: Literal["response"] = "response"
text: str
audio_url: Optional[str] = None
sources: list = Field(default_factory=list)
suggested_actions: list = Field(default_factory=list)
```

```
class StatusMessage(WSMessage):
"""Status update message."""
type: Literal["status"] = "status"
status: Literal["listening", "thinking", "speaking", "error"]
message: Optional[str] = None
```

```
class NavigationAction(BaseModel):
"""Navigation action request."""
action: Literal["scroll", "click", "navigate", "focus"]
params: dict = Field(default_factory=dict)
```

```
class Config:
    json_schema_extra = {
        "example": {
            "action": "scroll",
            "params": {"direction": "down", "amount": 100}
        }
    }
```

```


DO NOT modify existing models:
- LLMRequest, LLMResponse
- Project, Document
- All other existing models
```

2.5 Обновить `pyproject.toml`

```
@cursor Add voice assistant dependencies to pyproject.toml

File: pyproject.toml
Location: In dependencies array

Add to existing dependencies (keep all current ones):
```

```
dependencies = [
# ... existing dependencies ...
```

```
# Voice Assistant (NEW)
"openai-whisper>=20231117",
"faster-whisper>=1.0.0",
"vosk>=0.3.45",
"soundfile>=0.12.1",
"librosa>=0.10.0",
"webrtcvad>=2.0.10",
"pydub>=0.25.1",
"elevenlabs>=0.2.0",
"azure-cognitiveservices-speech>=1.34.0",
"gtts>=2.4.0",
```

]

```

Add to dev dependencies:
```

dev = [
# ... existing dev dependencies ...

```
# Voice testing (NEW)
"pytest-asyncio>=0.23.0",
"pytest-timeout>=2.2.0",
"sounddevice>=0.4.6",
```

]

```

Add new build requirements section if not exists:
```

[build-system]
requires = [
"setuptools>=75",
"wheel",
"torch>=2.0.0" # For Whisper
]

```

DO NOT remove existing dependencies
```

2.6 Обновить `requirements.txt` (если используется)

```
@cursor Add to requirements.txt if it exists

File: requirements.txt (if exists)
Action: Append to end of file

Add:
```

Voice Assistant

openai-whisper>=20231117
faster-whisper>=1.0.0
vosk>=0.3.45
soundfile>=0.12.1
librosa>=0.10.0
webrtcvad>=2.0.10
pydub>=0.25.1

elevenlabs>=0.2.0
azure-cognitiveservices-speech>=1.34.0
gtts>=2.4.0

2.7 Обновить `docker-compose.yaml`

```
@cursor Add voice service configuration to docker-compose.yaml

File: compose.yaml
Location: After existing services, before networks section

Add new service:
```

Voice Assistant Service (NEW)

voice-worker:
build:
context: .
dockerfile: Dockerfile
container_name: sitellm-voice-worker
depends_on:
- mongo
- redis
environment:
- MONGO_HOST=mongo
- REDIS_HOST=redis
- WHISPER_MODEL=base
- WHISPER_DEVICE=cpu
- ELEVENLABS_API_KEY=$ELEVENLABS_APIKEY : - - AZURE_SPEECH_KEY =$ {AZURE_SPEECH_KEY:-}
volumes:
- ./voice:/app/voice:ro
- voice-models:/app/models
- voice-cache:/app/cache
command: python -m voice.worker
restart: unless-stopped
networks:
- sitellm-network

volumes:

Add to existing volumes

voice-models:
driver: local
voice-cache:
driver: local

```
Update app service to expose WebSocket:
```

app:
# ... existing configuration ...
ports:
- "8000:8000"
- "8001:8001" # NEW: WebSocket port
environment:
# ... existing env vars ...
- WS_PORT=8001 # NEW

```
DO NOT modify:
- Existing services (mongo, redis, app, celery_worker, etc.)
- Existing volumes
- Network configuration
```

2.8 Обновить `.env.example`

```
@cursor Add voice configuration to .env.example

File: .env.example
Location: At the end, add new section

Add:
```

==================== VOICE ASSISTANT CONFIG ====================

Speech Recognition

WHISPER_MODEL=base # tiny, base, small, medium, large
WHISPER_DEVICE=auto # auto, cpu, cuda
VOSK_MODEL_PATH=./models/vosk-model-small-ru-0.22

Text-to-Speech

```
TTS_DEFAULT_PROVIDER=elevenlabs # elevenlabs, azure, browser
ELEVENLABS_API_KEY=
ELEVENLABS_VOICE_ID=21m00Tcm4TlvDq8ikWAM
AZURE_SPEECH_KEY=
AZURE_SPEECH_REGION=westeurope
```

Voice Session Settings

```
VOICE_SESSION_TIMEOUT=3600 # seconds (1 hour)
VOICE_MAX_CONCURRENT_SESSIONS=100
VOICE_CACHE_AUDIO=true
VOICE_CACHE_TTL=604800 # seconds (7 days)
```

WebSocket

```
WS_PORT=8001
WS_MAX_CONNECTIONS=1000
WS_PING_INTERVAL=30 # seconds
```

Audio Processing

```
AUDIO_SAMPLE_RATE=16000
AUDIO_CHUNK_DURATION=1.0 # seconds
AUDIO_MAX_DURATION=60 # seconds
```

Dialog Settings

```
DIALOG_MAX_CONTEXT_LENGTH=10 # number of turns
DIALOG_INTENT_CONFIDENCE_THRESHOLD=0.7
```

Cost Tracking

```
VOICE_ENABLE_COST_TRACKING=true
VOICE_COST_ALERT_THRESHOLD=100 # USD per day
```

```
DO NOT modify existing environment variables
```

3. Новые файлы для создания

3.1 Backend Voice Router

```
@cursor Create voice API router

File: voice/router.py (NEW FILE)

Create complete FastAPI router with:

1. Session endpoints:
   - POST /session/start
   - GET /session/{session_id}
   - DELETE /session/{session_id}

2. WebSocket endpoint:
   - WS /ws/{session_id}

3. Recognition endpoints:
   - POST /recognize (sync)
   - WS /recognize/stream (async)

4. Synthesis endpoints:
   - POST /synthesize
   - GET /audio/{audio_id}
   - GET /voices/list

5. Dialog endpoints:
   - POST /dialog/intent
   - POST /dialog/respond

6. Analytics endpoints:
   - GET /analytics/{project}
   - GET /session/{session_id}/history

Include:
- Rate limiting with slowapi
- Authentication checks
- Request/response validation
- Error handling
- Prometheus metrics
- Structured logging
- OpenAPI documentation

Full implementation ~500 lines
```

3.2 Speech Recognizer

```
@cursor Create speech recognition module

File: voice/recognizer.py (NEW FILE)

Implement:
1. WhisperRecognizer class
   - Model loading and caching
   - Sync and async recognition
   - Streaming support
   - GPU acceleration
   - Language detection

2. VoskRecognizer class (lightweight fallback)
   - Offline recognition
   - Lower resource usage
   - Real-time processing

3. RecognizerFactory
   - Provider selection
   - Configuration management
   - Error handling

Include comprehensive error handling and logging
Full implementation ~400 lines
```

3.3 Speech Synthesizer

```
@cursor Create TTS synthesis module

File: voice/synthesizer.py (NEW FILE)

Implement:
1. TTSManager class
   - Provider orchestration
   - Caching logic
   - Cost tracking

2. Provider implementations:
   - ElevenLabsProvider
   - AzureTTSProvider
   - GTTSProvider (fallback)

3. Audio processing:
   - Format conversion
   - Quality normalization
   - Caching

Include emotion detection and SSML support
Full implementation ~600 lines
```

3.4 Dialog Manager

```
@cursor Create dialog management system

File: voice/dialog_manager.py (NEW FILE)

Implement:
1. DialogManager class
   - Context management
   - Turn-taking logic
   - State persistence
```

```
2. IntentClassifier
   - Intent recognition using LLM
   - Entity extraction
   - Confidence scoring

3. ResponseGenerator
   - RAG integration (use existing retrieval/)
   - Template-based responses
   - Dynamic content generation

4. ContextManager
   - Conversation history
   - User preferences
   - Session state

Full implementation ~500 lines
```

3.5 Frontend Voice Widget

```
@cursor Create voice widget frontend

File: widget/voice/src/index.ts (NEW FILE)

Implement main widget class:
1. VoiceWidget initialization
2. UI rendering
3. Event handling
4. State management
5. WebSocket connection
6. Audio recording/playback
7. Visualization

Full TypeScript implementation ~800 lines
Include CSS and HTML templates
```

3.6 WebSocket Manager

```
@cursor Create WebSocket client manager

File: widget/voice/src/core/WebSocketManager.ts (NEW FILE)

Implement:
1. Connection management
   - Auto-reconnection
   - Ping/pong keepalive
   - Connection pooling

2. Message routing
   - Type-based handlers
   - Queue management
   - Error recovery
```

```
3. State synchronization
   - Session state
   - Audio state
   - UI state

Full implementation ~300 lines
```

3.7 Audio Visualizer

```
@cursor Create audio visualization component

File: widget/voice/src/ui/Visualizer.ts (NEW FILE)

Implement:
1. Wave visualization using Canvas API
2. Real-time audio analysis
3. State-based animations:
   - Listening (pulsing)
   - Processing (spinner)
   - Speaking (wave)
   - Error (shake)

4. Responsive design
5. Accessibility support

Full implementation ~400 lines
Include CSS animations
```

4. Тесты для добавления

4.1 Backend Tests

```
@cursor Create voice API tests

File: tests/voice/test_voice_api.py (NEW FILE)

Test scenarios:
1. Session lifecycle (create, use, expire)
2. Speech recognition (sync and streaming)
3. Speech synthesis (all providers)
4. WebSocket communication
5. Intent recognition
6. Dialog management
7. Error handling
8. Rate limiting
9. Cost tracking
10. Analytics

Use pytest-asyncio for async tests
```

```
Mock external APIs (Whisper, ElevenLabs, Azure)
Achieve 90%+ coverage
```

4.2 Frontend Tests

```
@cursor Create widget tests

File: widget/voice/tests/VoiceWidget.test.ts (NEW FILE)

Test scenarios:
1. Widget initialization
2. Microphone permissions
3. Audio recording
4. WebSocket connection
5. Message handling
6. UI state transitions
7. Error handling
8. Cleanup on unmount

Use Jest + Testing Library
Mock Web APIs (MediaDevices, WebSocket, AudioContext)
```

4.3 Integration Tests

```
@cursor Create end-to-end tests

File: tests/voice/test_voice_e2e.py (NEW FILE)

Test complete workflows:
1. User speaks → transcription → response → TTS
2. Navigation commands
3. Knowledge queries
4. Error recovery
5. Session timeout
6. Concurrent sessions

Use real audio samples for testing
Test in containerized environment
```

5. Конфигурационные файлы

5.1 TypeScript Config

```
@cursor Create TypeScript configuration

File: widget/voice/tsconfig.json (NEW FILE)
```

```
{
"compilerOptions": {
"target": "ES2020",
"module": "ESNext",
"lib": ["ES2020", "DOM", "DOM.Iterable"],
"moduleResolution": "node",
"strict": true,
"esModuleInterop": true,
"skipLibCheck": true,
"forceConsistentCasingInFileNames": true,
"declaration": true,
"declarationMap": true,
"sourceMap": true,
"outDir": "./dist",
"rootDir": "./src",
"types": ["node", "jest"]
},
"include": ["src/**/*"],
"exclude": ["node_modules", "dist", "tests"]
}
```

5.2 Webpack Config

```
@cursor Create Webpack configuration

File: widget/voice/webpack.config.js (NEW FILE)

Configure:
1. TypeScript compilation
2. CSS/SCSS processing
3. Asset handling
4. Development server
5. Production optimization
6. Source maps
7. Code splitting

Output bundle to dist/voice-widget.js
```

5.3 Package.json

```
@cursor Create package.json for voice widget

File: widget/voice/package.json (NEW FILE)
```

```
{
"name": "@sitellm/voice-widget",
"version": "1.0.0",
"description": "Voice assistant widget for SiteLLM Vertebro",
"main": "dist/index.js",
"types": "dist/index.d.ts",
"scripts": {
"build": "webpack --mode production",
"dev": "webpack serve --mode development",
"test": "jest",
"test:watch": "jest --watch",
"lint": "eslint src/**/*.ts",
"type-check": "tsc --noEmit"
},
"dependencies": {
"eventemitter3": "^5.0.0",
"uuid": "^9.0.0"
},
"devDependencies": {
"@types/jest": "^29.5.0",
"@types/node": "^20.0.0",
"@typescript-eslint/eslint-plugin": "^6.0.0",
"@typescript-eslint/parser": "^6.0.0",
"css-loader": "^6.8.0",
"eslint": "^8.50.0",
"jest": "^29.7.0",
"style-loader": "^3.3.0",
"ts-jest": "^29.1.0",
"ts-loader": "^9.5.0",
"typescript": "^5.3.0",
"webpack": "^5.89.0",
"webpack-cli": "^5.1.0",
"webpack-dev-server": "^4.15.0"
}
}
```

6. Миграция и deployment

6.1 Database Migration

```
@cursor Create database migration script

File: scripts/migrate_voice_schema.py (NEW FILE)

Script should:
1. Check if voice collections exist
2. Create collections with proper schema
3. Create indexes
4. Migrate any existing data if needed
5. Verify migration success
6. Rollback on error

Run with: python scripts/migrate_voice_schema.py
```

6.2 Model Download Script

```
@cursor Create model download script

File: scripts/download_voice_models.sh (NEW FILE)

Download:
1. Whisper base model
2. Vosk Russian model
3. Verify checksums
4. Extract to correct locations

Run before first deployment
```

6.3 Deployment Documentation

```
@cursor Create deployment guide

File: docs/voice_deployment.md (NEW FILE)

Document:
1. Prerequisites (GPU, storage, API keys)
2. Installation steps
3. Configuration
4. Testing procedure
5. Troubleshooting
6. Monitoring setup
7. Scaling considerations

Include production checklist
```

7. Приоритизация задач для Cursor.ai

Критичный путь (делать в порядке):

```
Week 1: Infrastructure
□ Task 1.1: Directory structure
□ Task 2.1-2.4: MongoDB extensions
□ Task 3.1-3.2: Environment setup

Week 2: Backend Core
□ Task 4.1: Voice router
□ Task 4.2: Speech recognizer (Whisper)
□ Task 4.3: Basic TTS (GTTS fallback)
□ Task 4.4: WebSocket handler

Week 3: Frontend Core
□ Task 5.1: Voice widget skeleton
□ Task 5.2: WebSocket manager
□ Task 5.3: Basic UI components

Week 4: Integration
□ Task 6.1: Dialog manager
□ Task 6.2: Intent classifier
□ Task 6.3: RAG integration

Week 5: Advanced Features
□ Task 7.1: ElevenLabs integration
□ Task 7.2: Azure TTS integration
□ Task 7.3: Audio visualizations
□ Task 7.4: Navigation controller

Week 6: Testing & Polish
□ Task 8.1: Unit tests (90%+ coverage)
□ Task 8.2: Integration tests
□ Task 8.3: E2E tests
□ Task 8.4: Performance optimization

Week 7: Documentation & Deployment
□ Task 9.1: API documentation
□ Task 9.2: User documentation
□ Task 9.3: Deployment scripts
□ Task 9.4: Production testing
```

8. Как использовать с Cursor.ai

Общий workflow:

```
1. Откройте проект в Cursor.ai:
   cd sitellm_vertebro
   cursor .
```

```
2. Для каждой задачи используйте Composer (Cmd+I):
   - Скопируйте задачу из этого документа
   - Вставьте в Cursor Composer
   - Cursor создаст файлы и код
   - Проверьте сгенерированный код
   - Запустите тесты

3. Используйте @ mentions для контекста:
   @api.py - текущий API файл
   @mongo.py - текущий MongoDB клиент
   @models.py - текущие модели

4. После каждой задачи:
   - git add .
   - git commit -m "feat: <описание задачи>"
   - git push origin voice-assistant-feature

5. Тестирование:
   - pytest tests/voice/
   - npm test (для frontend)
   - Cursor автоматически запустит линтеры

6. Для debugging используйте Cursor Chat (Cmd+L):
   "Debug this WebSocket connection issue"
   "Optimize this audio processing function"
   "Fix this TypeScript type error"
```

Shortcuts для ускорения:

```
Cmd/Ctrl + K: Inline edit (быстрые правки)
Cmd/Ctrl + I: Composer (сложные задачи)
Cmd/Ctrl + L: Chat (вопросы и отладка)
Cmd/Ctrl + Shift + P → "Cursor: Generate Tests" (генерация тестов)
```

Итого добавлений к текущему проекту:

- ▯ 2 новые директории (`voice/`, `widget/voice/`)
- ▯ ~15 новых Python файлов (~3500 строк)
- ▯ ~20 новых TypeScript файлов (~2000 строк)
- ▯ Модификации 6 существующих файлов
- ✅ ~30 новых тестовых файлов
- ▯ 5 новых документов

Все изменения **аддитивные** - ничего не удаляется из текущего проекта, только расширение функциональности.

1. https://www.builder.io/blog/cursor-tips
2. https://forum.cursor.com/t/ai-rules-and-other-best-practices/132291
3. https://nmn.gl/blog/cursor-guide

4. https://cursor.com
5. https://learn-cursor.com/en/wiki/user-guide/code-completion
6. https://www.jit.io/resources/appsec-tools/top-python-code-analysis-tools-to-improve-code-quality
7. https://realpython.com/python-code-quality/
8. https://www.augustinfotech.com/blogs/how-to-use-coverage-unit-testing-in-fastapi-using-pytest/
9. https://www.frugaltesting.com/blog/what-is-fastapi-testing-tools-frameworks-and-best-practices
10. https://www.youtube.com/watch?v=8QN23ZThdRY
11. https://pyimagesearch.com/2024/11/04/enhancing-github-actions-ci-for-fastapi-build-test-and-publish/
12. https://deepsource.com/blog/setup-static-analysis-python
13. https://www.youtube.com/watch?v=oWDLsCOPOk0
14. https://leapcell.io/blog/fastapi-performance-hacks
15. https://skywork.ai/blog/cursor-1-7-ai-autocomplete-pair-programming-best-practices/
16. https://cursor.com/docs/cli/cookbook/code-review
17. https://testdriven.io/courses/scalable-fastapi-aws/tests-code-quality/
18. https://www.marketcalls.in/python/a-comprehensive-guide-to-python-linters-pylint-black-and-ruff-explained.html
19. https://www.reddit.com/r/cursor/comments/1kvo5d2/how_to_automate_accepting_code_changes/
20. https://cursor.com/agents
21. https://www.datacamp.com/tutorial/cursor-ai-code-editor
22. https://cursor.com/features
23. https://dev.to/atifwattoo/fastapi-the-ultimate-guide-to-building-high-performance-apis-with-python-9hg
24. https://www.sidetool.co/post/cursor-ai-tips-mastering-multi-line-edits-and-predictive-completions/
25. https://fastapi.tiangolo.com
26. https://www.reddit.com/r/ChatGPTCoding/comments/1hu276s/how_to_use_cursor_more_efficiently/
27. https://cursor.com/docs/agent/overview
28. https://www.reddit.com/r/Python/comments/xf4l0a/what_stack_or_tools_are_you_using_for_ensuring/
29. https://inventwithpython.com/blog/2022/11/19/python-linter-comparison-2022-pylint-vs-pyflakes-vs-flake8-vs-autopep8-vs-bandit-vs-prospector-vs-pylama-vs-pyroma-vs-black-vs-mypy-vs-radon-vs-mccabe/
30. https://blog.fidelramos.net/software/python-code-quality
31. https://lambdasec.github.io/AutoFix-Automated-Vulnerability-Remediation-using-Static-Analysis-and-LLMs/
32. https://arxiv.org/html/2509.16275v1
33. https://dev.to/saminarp/write-clean-python-code-using-pylint-and-black-17e3
34. https://www.youtube.com/watch?v=gop9Or2V_80
35. https://github.com/vintasoftware/python-linters-and-code-analysis
36. https://spectralops.io/blog/static-code-analysis-for-python-7-features-to-look-out-for/
37. https://fastapi.tiangolo.com/tutorial/testing/

38. https://github.com/lambdasec/autofix
39. https://dev.to/polakshahar/fastapi-furious-tests-the-need-for-speed-11oi
40. https://www.sonarsource.com/knowledge/languages/python/
41. https://www.builder.io/blog/cursor-tips
42. https://forum.cursor.com/t/ai-rules-and-other-best-practices/132291
43. https://nmn.gl/blog/cursor-guide
44. https://www.youtube.com/watch?v=8QN23ZThdRY
45. https://cursor.com/agents
46. https://skywork.ai/blog/cursor-1-7-ai-autocomplete-pair-programming-best-practices/
47. https://cursor.com