Novel Summarizer — Backend

Ниже — готовые файлы для репозитория: **server.js** и **README.md**. Копируй их в GitHub как есть. Подходит для локального запуска и Replit/Render/Railway.

---

server.js

```
import 'dotenv/config';
import express from 'express';
import cors from 'cors';
import axios from 'axios';
import { JSDOM } from 'jsdom';
import { Readability } from '@mozilla/readability';
import OpenAI from 'openai';

const app = express();
app.use(cors());
app.use(express.json({ limit: '1mb' }));

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

// Очень простое хранилище (в памяти). Для продакшена замените на БД.
const DB = { novels: {} }; // { id: { id, url, title, lang, depth, style,
chapters:[{n,url,text,summary}], createdAt } }
const newId = () => Math.random().toString(36).slice(2, 10);

// ---------- Утилиты ----------
const fetchHTML = async (url) => {
  const { data } = await axios.get(url, {
    timeout: 20000,
    headers: {
      'User-Agent': 'NovelSummarizer/1.0 (+https://example.com)'
    }
  });
  return data;
};

const extractText = (html, baseURL) => {
  const dom = new JSDOM(html, { url: baseURL });
  const reader = new Readability(dom.window.document);
  const art = reader.parse();
  // Фолбэк: если Readability ничего не нашёл, берём видимый текст
  const text = (art?.textContent || dom.window.document.body.textContent ||
'').replace(/\n{3,}/g, '\n\n');
  return text.trim();
};
```

```
const summarize = async ({ text, lang = 'RU', depth = 'short', style =
'neutral' }) => {
  const targetLen = depth === 'full' ? 'подробный' : depth === 'mid' ?
'средний' : 'краткий';
  const tone = style === 'dramatic' ? 'драматичный' : style === 'humor' ? 'с
иронией' : 'нейтральный';
  const prompt = `Сделай ${targetLen} пересказ главы (${lang}). Тон: ${tone}.
Выдай строго в формате:
1) Ключевые пункты: маркированный список из 3–7 пунктов.
2) Синопсис: 5–8 предложений связного текста.
3) Герои: список «имя — роль/статус».
Текст главы между <chapter>…</chapter>.
<chapter>\n${text.slice(0, 18000)}\n</chapter>`;

  const resp = await openai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages: [{ role: 'user', content: prompt }],
    temperature: 0.3,
  });
  return resp.choices?.[0]?.message?.content?.trim() || '';
};

// Нахождение ссылок на главы (очень простой хелпер; можно заменить под
конкретные сайты)
const findChapterLinks = (html, baseURL) => {
  const dom = new JSDOM(html, { url: baseURL });
  const links = [...dom.window.document.querySelectorAll('a')];
  const cand = links
    .map(a => ({ href: a.href, text: (a.textContent || '').trim() }))
    .filter(x => /chapter|chapitre|capitulo|глава|том\s*\d+.*глава/
i.test((x.text + ' ' + x.href)));

  const withN = cand.map(x => {
    const m = (x.text || x.href).match(/(?:глава|chapter|capitulo|chapitre)
\s*(\d+)/i);
    return { ...x, n: m ? parseInt(m[1], 10) : null };
  });

  withN.sort((a, b) => (a.n || 999999) - (b.n || 999999));

  const seen = new Set();
  const out = [];
  for (const x of withN) {
    if (!seen.has(x.href)) { seen.add(x.href); out.push(x); }
  }
  return out.map((x, i) => ({ n: x.n ?? (i + 1), url: x.href }));
};

// ---------- API ----------
// 1) Начать обработку: получить оглавление и зарегистрировать новеллу
app.post('/api/process', async (req, res) => {
```

```
  try {
    const { url, lang = 'RU', depth = 'short', style = 'neutral', title =
'' } = req.body || {};
    if (!url) return res.status(400).json({ error: 'url required' });

    // ВАЖНО: уважайте robots.txt и ToS источника; обрабатывайте только
публичные страницы
    const html = await fetchHTML(url);
    const chapters = findChapterLinks(html, url);
    if (!chapters.length) return res.status(400).json({ error: 'Не нашёл
главы на странице' });

    const id = newId();
    const name = title || (() => {
      try { const u = new URL(url); return
decodeURIComponent(u.pathname.split('/').filter(Boolean).pop() ||
u.hostname); }
      catch { return url; }
    })();

    DB.novels[id] = {
      id, url, title: name, lang, depth, style,
      chapters: chapters.slice(0, 200).map(c => ({ ...c })), // ограничим до
200 глав для безопасности
      createdAt: Date.now(),
    };

    res.json({ id, chapters: DB.novels[id].chapters });
  } catch (e) {
    res.status(500).json({ error: e.message || 'internal error' });
  }
});

// 2) Сгенерировать/получить пересказ конкретной главы
app.post('/api/novels/:id/chapters/:n/summary', async (req, res) => {
  try {
    const nov = DB.novels[req.params.id];
    if (!nov) return res.status(404).json({ error: 'novel not found' });
    const ch = nov.chapters.find(x => String(x.n) === String(req.params.n));
    if (!ch) return res.status(404).json({ error: 'chapter not found' });

    if (!ch.text) {
      const html = await fetchHTML(ch.url);
      ch.text = extractText(html, ch.url);
      if (!ch.text) return res.status(400).json({ error: 'chapter text not
found' });
    }
    ch.summary = await summarize({ text: ch.text, lang: nov.lang, depth:
nov.depth, style: nov.style });
    res.json({ n: ch.n, summary: ch.summary });
  } catch (e) {
```

```
    res.status(500).json({ error: e.message || 'internal error' });
  }
});

// 3) Метаданные новеллы (для фронта)
app.get('/api/novels/:id', (req, res) => {
  const n = DB.novels[req.params.id];
  if (!n) return res.status(404).json({ error: 'not found' });
  res.json({ id: n.id, title: n.title, url: n.url, lang: n.lang, depth:
n.depth, style: n.style,
    chapters: n.chapters.map(c => ({ n: c.n, hasSummary: !!c.summary })) });
});

// 4) Список новелл (простая лента)
app.get('/api/novels', (req, res) => {
  const items = Object.values(DB.novels)
    .sort((a, b) => b.createdAt - a.createdAt)
    .map(n => ({ id: n.id, title: n.title, url: n.url, chapters:
n.chapters.length }));
  res.json(items);
});

// 5) Скачать собранные пересказы (txt)
app.get('/api/novels/:id/download.txt', (req, res) => {
  const n = DB.novels[req.params.id];
  if (!n) return res.status(404).send('not found');
  const body = [
    `# ${n.title}`,
    `Источник: ${n.url}`,
    '',
    ...n.chapters.filter(c => c.summary).map(c => `Глава ${c.n}\n${c.summary}
\n`)
  ].join('\n');
  res.setHeader('Content-Type', 'text/plain; charset=utf-8');
  res.setHeader('Content-Disposition', `attachment; filename="${(n.title ||
'novel').replace(/[^a-z0-9_\-\.]+/gi,'_')}_summaries.txt"`);
  res.send(body);
});

// Healthcheck
app.get('/health', (_, res) => res.json({ ok: true }));

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log('API running on http://localhost:' +
PORT));
```

`README.md`

````
# Novel Summarizer — Backend

Лёгкий Node.js API для пересказа глав новелл по URL. Парсит оглавление,
достаёт текст каждой главы и вызывает OpenAI для генерации пересказа.

## Возможности
- `POST /api/process` — принимает URL страницы новеллы, ищет ссылки на главы
и регистрирует новеллу
- `POST /api/novels/:id/chapters/:n/summary` — генерирует (или возвращает
кеш) пересказ главы
- `GET /api/novels/:id` — метаданные новеллы + статусы глав
- `GET /api/novels` — список новелл
- `GET /api/novels/:id/download.txt` — скачать собранные пересказы
- `GET /health` — healthcheck

## Требования
- Node.js 18+
- Ключ OpenAI в переменной окружения `OPENAI_API_KEY`

## Установка
```bash
npm init -y
npm i express cors axios jsdom @mozilla/readability openai dotenv
````

Создайте `.env`:

```
OPENAI_API_KEY=sk-...ваш_ключ...
PORT=3000
```

Добавьте в `package.json` (ESM):

```
{
  "type": "module",
  "scripts": {
    "start": "node server.js"
  }
}
```

Запуск

```
npm start
# API: http://localhost:3000
```
```

Подключение фронтенда

Во фронтенде задайте URL API: `API_BASE = 'https://your-host.example.com'` или через локальное хранилище (в моём index.html это делается в Меню → API URL). Дальше используйте эндпоинты выше.

Деплой

- **Replit**: импортируйте репозиторий, добавьте секрет `OPENAI_API_KEY`, нажмите Run.
- **Render/Railway**: создайте Web Service, задайте переменные окружения. Build: `npm install`, Start: `npm start`.
- **Vercel**: этот сервер — долгоиграющие запросы (fetch внешних страниц) → лучше Render/Railway. При желании — как serverless, но учитывайте таймауты.

Юридически важно

- Обрабатывайте только публичные страницы, уважайте **robots.txt** и **Terms of Service** источника. Не обходите paywall/логин.
- В выдаче всегда держите активную **ссылку на источник**.
- Не храните полный текст глав, если это запрещено; храните только краткое содержание.

Ограничения и тюнинг

- По умолчанию ограничение до 200 глав на новеллу (см. `slice(0, 200)`).
- Хук `findChapterLinks` очень простой; при необходимости напишите адаптер под конкретный сайт.
- Модель: `gpt-4o-mini` (быстро/дёшево). Можно заменить на более мощную.
- Для продакшена добавьте БД (Postgres/SQLite) и очереди задач (BullMQ/Cloud Tasks).

Лицензия

MIT ```

Быстрый чек-лист для репозитория

1. Создай репо и добавь файлы: `server.js`, `.env.example`, `README.md`, `package.json` (с `"type":"module"`).
2. Укажи в README ссылку на фронтенд.
3. Проверь `GET /health` после деплоя.