### \*\*Проект: "AetherSound" — Генератор процедурной музыки на основе настроения\*\*

### \*\*1. Концепция и Ценностное предложение\*\*

\*\*Elevator Pitch:\*\* "AetherSound — это веб-приложение, которое создает уникальные, никогда не повторяющиеся звуковые ландшафты в реальном времени. Просто выберите настроение (например, 'Умиротворенный дождь', 'Космическое путешествие', 'Энергичный фокус'), настройте параметры, и погрузитесь в звучание, созданное искусственным интеллектом специально для вас."

\*\*Аналоги:\*\* Endel, GeneratedMusic, но более открытый и настраиваемый.

---

### \*\*2. Детальное описание функционала (User Stories)\*\*

#### \*\*Ядро приложения:\*\*

\* \*\*Генератор звука:\*\*

\* Генерация базовых波形ов (синус, пила, квадрат, шум) через `OscillatorNode`.

\* Наложение эффектов: реверб (`ConvolverNode`), дилей (`DelayNode`), фильтры (`BiquadFilterNode`), хорус, фленджер.

\* Управление огибающей (ADSR — Attack, Decay, Sustain, Release) для каждого звука с помощью `GainNode`.

\* \*\*Процедурная композиция:\*\*

\* \*\*Мелодия:\*\* Генерация последовательностей нот (паттернов) с помощью Марковских цепей. Цепь обучается на наборе MIDI-файлов в выбранном жанре (например, lo-fi hip-hop, ambient). Пользователь может влиять на "сложность" и "предсказуемость" мелодии.

\* \*\*Ритм:\*\* Генерация барабанных петель (kick, snare, hi-hat). Используются предзаписанные семплы (`.wav`), которые запускаются с заданным интервалом (`AudioBufferSourceNode`). Ритмические паттерны также генерируются процедурно.

\* \*\*Гармония:\*\* Автоматический подбор аккордовой последовательности, соответствующей выбранному настроению и тональности.

\* \*\*Пользовательский интерфейс:\*\*

\* \*\*Панель быстрого выбора:\*\* Большие кнопки с предустановками ("Настроения").

\* \*\*Расширенные настройки:\*\* Слайдеры и селекторы для тонкой настройки параметров:

\* \*Темп (BPM).\*

\* \*Ключ (Tonality).\*

\* \*Сложность мелодии.\*

\* \*Интенсивность ритма.\*

\* \*"Влажность" (уровень реверберации).\*

\* \*Громкость отдельных слоев (мелодия, бас, ритм, пады).\*

\* \*\*Визуализатор:\*\* Красивая визуализация звука на основе `AnalyserNode` (частотный спектр, waveform). Можно использовать Three.js или Canvas для создания абстрактных, тематических визуализаций под каждое "настроение".

\* \*\*Элементы управления:\*\* Кнопки Play/Pause, Stop. Кнопка "Randomize" для полного перегенерирования трека. Кнопка "Export to WAV" для экспорта сгенерированного отрывка (используя `MediaRecorder API` или `OfflineAudioContext`).

---

### \*\*3. Технологический стек (Tech Stack)\*\*

| Слой | Технологии | Обоснование |

| \*\*Frontend\*\* | \*\*React\*\* + \*\*TypeScript\*\* | Структура, типизация сложных объектов Audio API. |

| \*\*Стили\*\* | \*\*Tailwind CSS\*\* / \*\*Styled Components\*\* | Быстрое создание адаптивного и современного UI. |

| \*\*Браузерное Audio\*\* | \*\*Web Audio API\*\* | Основа всего. Для более сложной логики можно обернуть в библиотеку (Tone.js), но для полного контроля лучше свой слой абстракции. |

| \*\*Визуализация\*\* | \*\*Canvas API\*\* / \*\*D3.js\*\* / \*\*Three.js\*\* | Для рисования спектрограмм и абстрактных форм. |

| \*\*Генеративные алгоритмы\*\* | \*\*Собственная реализация на JavaScript\*\* | Марковские цепи для мелодий, клеточные автоматы или шум Перлина для параметров. |

| \*\*Семплы\*\* | Предзаписанные `.wav` файлы | Барабаны, перкуссия, атмосферные пады. |

| \*\*Бэкенд (минимальный)\*\* | \*\*Node.js\*\* + \*\*Express\*\* | Для обслуживания статики, семплов и, возможно, предобученных моделей Марковских цепей. |

| \*\*Хостинг\*\* | \*\*Vercel\*\* / \*\*Netlify\*\* (Frontend), \*\*Railway\*\* (Backend) | Простота деплоя. |

---

### \*\*4. Архитектура и ключевые компоненты\*\*

1. \*\*Ядро звукогенерации (Sound Engine):\*\*

\* Класс `Synth`, управляющий AudioContext.

\* Классы для отдельных инструментов (`LofiPiano`, `FMSynth`, `DrumMachine`), каждый из которых создает и соединяет нужные ноды.

\* Менеджер эффектов (`EffectRack`).

2. \*\*Движок процедурной генерации (Procedural Engine):\*\*

\* `MarkovChain`: класс для создания и использования цепей Маркова для генерации нот.

\* `ChordProgressionGenerator`: класс, отвечающий за гармонию.

\* `PatternGenerator`: создает ритмические паттерны.

3. \*\*Слой состояния (State Management):\*\*

\* \*\*Zustand\*\* или \*\*Redux Toolkit\*\*. Для хранения всего состояния приложения: текущие параметры звука, активные паттерны, состояние воспроизведения.

4. \*\*Слой визуализации (Visualization Layer):\*\*

\* Компонент, который подписывается на данные `AnalyserNode` и рисует кадры с помощью `requestAnimationFrame`.

---

### \*\*5. Распределение ролей в команде на 4 человека\*\*

1. \*\*Архитектор звука / DSP-инженер (Самая сложная роль):\*\*

\* Разработка ядра звукогенерации на Web Audio API.

\* Создание и настройка инструментов, эффектов.

\* Реализация ADSR-огибающих, фильтров.

\* Решение проблем с latency и производительностью.

2. \*\*Разработчик алгоритмов генерации:\*\*

\* Разработка и обучение Марковских цепей на наборах MIDI.

\* Реализация алгоритмов генерации аккордов и ритмов.

\* Создание системы "настроений" — как наборы параметров влияют на алгоритмы.

3. \*\*Frontend-разработчик (UI/UX):\*\*

\* Создание всего пользовательского интерфейса в React.

\* Связывание UI-элементов (слайдеров, кнопок) со стейт-менеджером и звуковым движком.

\* Верстка, адаптивность, анимации.

4. \*\*Разработчик визуализаций / Full-stack:\*\*

\* Создание системы визуализации звука (Canvas/WebGL).

\* Развертывание минимального бэкенда (если нужен).

\* Настройка пайплайна сборки и деплоя.

\* Оптимизация производительности.

---

### \*\*6. Пошаговый план реализации (Roadmap)\*\*

\*\*Фаза 1: Прототип (2-3 недели)\*\*

\* Настройка React-приложения.

\* Создание базового `AudioContext`.

\* Реализация простого монофонического синтезатора, который играет одну ноту.

\* Добавление слайдера для изменения частоты среза фильтра.

\*\*Фаза 2: Основной функционал (3-4 недели)\*\*

\* Разработка полифонического синтезатора (можно использовать `Tone.js` для ускорения).

\* Реализация загрузки и воспроизведения барабанных семплов.

\* Создание простейшего генератора паттернов (зацикленный массив нот).

\* Разработка основного UI с кнопкой play и несколькими слайдерами.

\*\*Фаза 3: Процедурная генерация (3 недели)\*\*

\* Реализация Марковской цепи для генерации мелодий.

\* Создание генератора аккордовых последовательностей.

\* Привязка параметров генерации к элементам UI ("сложность", "настроение").

\*\*Фаза 4: Визуализация и полировка (2 недели)\*\*

\* Интеграция `AnalyserNode` и создание визуализатора.

\* Добавление предустановок ("настроений").

\* \*Реализация экспорта в WAV.\*

\* Тестирование, оптимизация, деплой.

---

### \*\*7. Основные технические вызовы и риски\*\*

1. \*\*Сложность Web Audio API:\*\* API низкоуровневый и сложный для новичков. Риск увязнуть в деталях.

2. \*\*Производительность:\*\* Генерация звука и сложная визуализация могут нагружать процессор. Необходима тщательная оптимизация, использование Web Workers для тяжелых вычислений (генерация паттернов).

3. \*\*Музыкальная теория:\*\* Членам команды потребуются базовые знания о теории музыки (гаммы, аккорды, ритм).

4. \*\*Субъективность результата:\*\* Сложно алгоритмически оценить, "приятно" ли звучит результат. Необходимо много тестирования и итераций.

Этот проект — не просто "еще одно веб-приложение". Это глубокое погружение в цифровой звук, алгоритмы и создание уникального пользовательского опыта. Успешная реализация будет смотреться в портфолио невероятно мощно.