технический путеводитель

как устроен «поток»

понятно о сайте, данных, публикации, безопасности и проверках качества.

актуально на 4 октября 2026 · публичный гид

1. главная идея

«поток» - это гибридный веб-сервис. его основа заранее собирается в обычные файлы и размещается на Timeweb. поэтому публичная часть быстрая, недорогая в эксплуатации и продолжает работать даже при проблемах с базой данных. поверх этой основы браузер запрашивает у Supabase свежие опубликованные материалы и аккуратно заменяет ими встроенный резерв.

> короткая формула: Next.js собирает сайт → Timeweb раздаёт файлы → React оживляет интерфейс в браузере → Supabase хранит редакционные данные и проверяет права.

как движутся данные

сначала человек получает уже собранную страницу. затем браузер проверяет, есть ли в Supabase более свежая опубликованная версия. если она есть, сайт заменяет или добавляет нужную запись. если сеть или база недоступна, сайт остаётся полезным благодаря встроенному резерву.

главное понимание

- Next.js здесь - инструмент разработки и сборки, а не постоянно работающий сервер.
- React здесь - слой интерактивности в браузере: выбор группы, переходы, админка, запросы данных.
- Supabase здесь - готовый защищённый backend: вход, PostgreSQL, правила доступа, серверные функции.
- статический резерв - страховка доступности, а не устаревший хвост.

2. карта архитектуры

у системы пять разных мест, где что-то происходит. ошибки часто возникают, когда их мысленно смешивают.



где	что происходит	чего там нет
компьютер разработчика	редактируются TypeScript, React, CSS и контент; запускаются тесты и сборка.	пользователи сюда не заходят.
этап сборки	Next.js выполняет код, формирует страницы, CSS и JavaScript, затем маршруты локализуются.	это не запрос реального посетителя.
Timeweb	отдаёт готовые HTML/CSS/JS, изображения, PDF и другие файлы.	не запускает Next.js и не хранит черновики.
браузер	рисует страницу, запускает клиентский React, хранит выбор группы и обращается к Supabase.	ему нельзя доверять секреты или окончательное решение о правах.
Supabase	проверяет пользователя, исполняет SQL/RPC, применяет RLS, хранит черновики и публикации.	не отвечает за дизайн и маршруты сайта.

почему так

проекту нужны одновременно простота обычного хостинга и безопасная совместная редакция. полностью статический сайт даёт первое, Supabase - второе. гибрид сохраняет достоинства обоих подходов, но требует внимательно синхронизировать резерв и публикации.

3. база: что вообще делает веб-сайт

слой	простыми словами	в «потоке»
HTML	скелет страницы: заголовки, ссылки, списки, формы.	создаётся Next.js заранее.
CSS	внешний вид и адаптивность: размеры, цвет, расположение.	mobile First, хвойная рамка, брейкпоинты и типографика.
JavaScript	поведение после загрузки: клики, состояние, запросы.	React-компоненты, переходы, выбор группы, админка.
HTTP/HTTPS	правила передачи файлов и данных по сети.	Timeweb отдаёт сайт; браузер обращается к Supabase по HTTPS.
JSON	текстовый формат обмена структурированными данными.	так Supabase возвращает публикации и принимает редакторские payload.

build time и runtime

build time - момент команды `pnpm build`. в этот момент Next.js читает исходники и заранее создаёт файлы. runtime - момент, когда студент открывает сайт. на Timeweb runtime почти целиком происходит в браузере, потому что отдельного Node-процесса нет.

что такое hydration

браузер сначала получает уже видимый HTML. затем загружается JavaScript, и React связывает разметку с обработчиками кликов и состоянием. этот этап называется гидратацией. поэтому страница может быстро появиться, а затем стать интерактивной.

> полезная аналогия: HTML - уже собранная сцена, CSS - декорации, JavaScript - механика, React - способ организовать эту механику, Next.js - цех, который всё собирает.

почему не мобильное приложение

веб-сервис открывается по ссылке, не требует установки и одинаково доступен на телефоне и компьютере. для учебного справочника с частыми короткими визитами это существенно снижает порог входа. нативное приложение имело бы смысл при офлайн-синхронизации, push-уведомлениях и глубокой интеграции с устройством.

4. что такое Next.js именно здесь

Next.js - фреймворк поверх React. он задаёт структуру страниц, сборку, оптимизацию кода и правила разделения логики. проект использует App Router: папки в `src/app` описывают отдельные экраны. точная версия библиотеки меняется, а эта роль остаётся той же.

что Next.js делает в проекте

- превращает React/TypeScript в готовые HTML, CSS и JavaScript.
- строит страницы: главная, актуальное, неделя, расписание, материалы, справка, лекция, день и админка.
- разделяет компоненты на исполняемые при сборке и клиентские с директивой `use client`.
- самохостит шрифты и разбивает клиентский код на части.
- при production-сборке использует `output: export` и создаёт каталог `out/`.

почему выбран Next.js

он даёт зрелую структуру для роста, удобные React-компоненты, предсказуемую маршрутизацию и возможность начать со статического экспорта. если позже понадобится серверный рендеринг или API, модель проекта не придётся полностью выбрасывать.

альтернатива	плюс	почему не основной выбор
чистые HTML/CSS/JS	минимум зависимостей и магии.	быстро растёт дублирование страниц и сложность интерактивности.
Vite + React SPA	очень простая клиентская сборка.	почти всё появляется после JavaScript; слабее модель заранее созданных страниц.
astro	сильный выбор для контентных статических сайтов, мало JS.	админка и сложная React-интерактивность потребовали бы отдельной архитектурной дисциплины.
Next.js с Node/Vercel	SSR, API и серверные действия.	дороже и сложнее эксплуатации, пока публичному сайту это не нужно.

5. React: из чего собран интерфейс

React позволяет описывать интерфейс как дерево компонентов. компонент получает данные и возвращает разметку. например, оболочка, навигация, строка лекции, выбор группы и редактор - отдельные части, которые можно проверять и переиспользовать.

server Components и Client Components

тип	когда выполняется	для чего
компонент без use client	в этом static export - преимущественно во время сборки.	сформировать стабильную разметку, не отправляя лишний JS.
компонент с use client	в браузере после загрузки.	состояние, эффекты, localStorage, формы, сетевые запросы, переходы.

состояние и Context

выбранная группа нужна нескольким экранам. React Context позволяет хранить её в одном провайдере и читать в дочерних компонентах без передачи через каждый промежуточный уровень. значение сохраняется в `localStorage`, поэтому остаётся после перезагрузки.

компромиссы React

- плюс: богатая экосистема, компоненты, тестирование и понятная модель роста.
- минус: клиентский JavaScript увеличивает объём загрузки и создаёт класс ошибок гидратации.
- правило проекта: интерактивность должна быть клиентской только там, где она действительно нужна.

какие были варианты

для простых фильтров хватило бы обычного JavaScript или Web Components. для глобального состояния можно было взять Redux/Zustand, но текущий объём не оправдывает дополнительную библиотеку. context + локальное состояние проще и достаточно прозрачно.

> важно: React скрывает или показывает кнопки для удобства, но не обеспечивает безопасность. любое действие пользователя всё равно заново проверяется в Supabase.

6. TypeScript и модель контента

TypeScript - это JavaScript с проверяемыми типами. он помогает заранее описать, какие поля обязаны быть у лекции, материала, объявления или записи справки. ошибка вроде пропущенной даты или неправильного вида блока выявляется до размещения на сайте.

статический слой

проверенный резерв хранится прямо в исходниках: лекции, объявления, справочник и снимок расписания. это не текст, разбросанный по страницам, а структурированные объекты, из которых интерфейс строится одинаково.

упрощённый пример

```
type Lecture = {
slug: string;
subject: string;
lessonDate: string;
topic: string;
summary: string;
materials: Material[];
};
```

почему это лучше случайного JSON

- редактор кода подсказывает доступные поля.
- сборка падает раньше, чем ошибка попадает пользователю.
- компоненты получают предсказуемые данные.
- миграция в базу может строиться детерминированно из одной модели.

вариант	когда подходит	компромисс
обычный JavaScript	маленький одноразовый сайт.	ошибки структуры проявляются позже.
JSON + Zod	данные приходят извне и нужна runtime-валидация.	ещё одна схема и зависимость.
CMS	много нетехнических редакторов и сложные медиа-процессы.	отдельная система, стоимость и интеграция.
только PostgreSQL	контент полностью динамический.	исчезает встроенный резерв и усложняется локальная работа.

7. русские адреса и маршруты

у Next.js папки страниц технически названы ASCII-именами: `current`, `week`, `lecture`. пользователь при этом видит понятные адреса: `/актуальное/`, `/неделя/`, `/конспект/123/`.

зачем два слоя

ASCII-папки надёжнее для инструментов сборки. русские публичные адреса лучше соответствуют продукту. после Next.js-сборки специальный скрипт переименовывает каталоги в `out/`. в development-режиме эквивалент обеспечивают rewrites.

единая точка правды

конспекты имеют глобальный неизменяемый номер, предметы — постоянное русское имя. старые ссылки переводят на новый адрес; неизвестный номер возвращает 404. все компоненты получают адреса из `src/lib/routes.ts`. это защищает от разных вариантов слэшей, старых адресов и двойного URL-кодирования.

задача	правило
обычная страница	использовать routes.current, routes.schedule и другие готовые значения.
день	использовать routes.day(date), чтобы ключ дата и формат query не расходились.
лекция	routes.lecture(slug) выдаёт /конспект/номер/. номер назначает БД; старые ключи сохраняются.
кодирование	не кодировать routes вручную. нативные переходы используют publicHref и сериализацию URL браузером.
материалы	файлы внутри /materials/ сохраняют ASCII-имена и не проходят локализацию маршрутов.

альтернативы

можно было оставить английские URL - технически проще, но хуже для продукта. можно было делать динамические русские сегменты средствами сервера, но у проекта нет постоянного Next.js-сервера. текущая локализация после сборки - осознанный компромисс для обычного файлового хостинга.

> самая опасная ошибка здесь - кодировать уже закодированный URL. тогда символ % превращается в %25, и страница перестаёт находиться.

8. сборка и публикация на Timeweb

команда `pnpm build` обновляет редакторский снимок расписания и реестр номеров, получает публичный каталог, запускает Next.js, локализует маршруты, создаёт календари и service worker. для этого нужен доступный Supabase. итоговый `apps/web/out/` - самодостаточный набор файлов, который загружается в веб-корень хостинга.

шаг	результат	зачем
снимки и адреса	обновляются расписание, номера, публичный каталог.	при ошибке Supabase сборка останавливается.
next build	создаются HTML со встроенным CSS, JS и ресурсы.	оптимизация и проверка production-режима.
localize-routes	технические папки получают русские публичные имена.	сохраняется контракт адресов.
загрузка out/ целиком	Timeweb получает одну согласованную версию.	нельзя публиковать выборочные HTML из разных сборок.

почему обычный Timeweb подходит

- не нужен постоянно работающий Node.js-процесс.
- меньше точек отказа и простая стоимость.
- обычный веб-сервер быстро раздаёт готовые неизменяемые файлы.
- публичная основа работает независимо от Supabase.

ограничения static export

- нет Next.js API routes, middleware и server actions во время запроса.
- персонализация и свежие данные выполняются клиентом.
- секрет нельзя спрятать в статической сборке: всё клиентское потенциально видно пользователю.

когда стоило бы перейти на сервер

если появятся операции с секретными внешними API, сложные персональные страницы, серверная генерация для каждого запроса или HttpOnly-сессии, понадобится Node/VPS/Vercel/Cloudflare либо отдельный backend. до этого static export экономически разумнее.

9. расписание, материалы и обновление данных

расписание ИГЭУ не запрашивается каждым браузером. команда `pnpm schedule:sync` заранее получает данные, проверяет таблицы, дубликаты и пустые группы, после чего сохраняет снимок в проекте. официальный каталог охватывает доступные факультеты, курсы и группы. поправки ответственных в Supabase хранятся отдельно; их публикация обновляет браузер, но статические календарные подписки требуют новой сборки.

подход	плюсы	минусы
текущий: проверенный снимок	быстро, стабильно, нет зависимости от доступности источника при каждом визите.	официальный снимок требует синхронизации; редакторские поправки читаются отдельно.
прямой запрос из браузера	всегда самые свежие данные при идеальном API.	CORS, нестабильность источника, формат может измениться, сложнее контроль.
свой сервер-прокси	можно кэшировать и нормализовать данные централизованно.	нужен постоянно работающий backend и мониторинг.

материалы

локальные PDF, документы и оптимизированные изображения лежат в публичном каталоге. в контенте сохраняются ссылки и описание источника. для редакторских ссылок разрешены внешние http/https или безопасные пути строго внутри `/materials/`; обход каталогов через `..` запрещён.

почему не автоматический Telegram-парсер

«поток» позиционируется как проверенный слой, а не зеркало чата. автоматический импорт переносил бы ошибки, личные данные и неактуальные сообщения. поэтому человек подтверждает дату, группу, источник и актуальность до публикации.

> архитектурный принцип проекта: автоматизировать доставку и проверку формы, но не отменять редакционную ответственность.

10. Supabase: готовый backend

Supabase объединяет PostgreSQL, авторизацию и HTTP-интерфейс. это позволяет статической админке безопасно работать с базой без собственного Node-сервера, если права корректно закреплены в самой базе.

часть Supabase	роль в «потоке»
Auth	вход по личному email и паролю; у помощника - выданный случайный логин. проверка пользователя и сессии.
storage	закрытый bucket note-images: фотографии и PNG рисунков. публичная выдача только изображений из опубликованных снимков.
PostgreSQL	черновики лекций, материалы, объявления, справка, профили, области доступа, ревизии.
PostgREST	автоматический HTTPS API поверх разрешённых таблиц и представлений.
RPC	серверные PostgreSQL-функции для атомарного сохранения, публикации, архива и управления доступом.
RLS	row Level Security: база решает, какие строки конкретный пользователь может читать и изменять.

почему Supabase

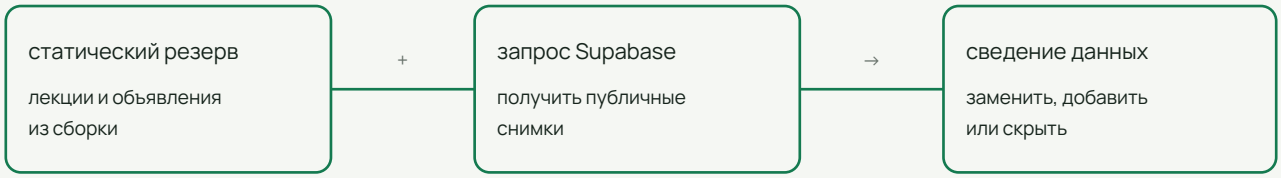
для небольшой команды он снимает большой объём инфраструктуры: не нужно отдельно поднимать API, систему пользователей и управляемую PostgreSQL. при этом логика остаётся в стандартной SQL-базе, а не только в закрытом визуальном конструкторе.

альтернатива	когда лучше	цена перехода
собственный Node/Nest backend	уникальная бизнес-логика, интеграции, полный контроль.	разработка API, auth, деплой, логи, обновления и безопасность.
Firebase	realtime-first, мобильная экосистема Google.	другая модель данных и правил, меньше преимуществ SQL.
headless CMS	контентная команда хочет готовую редакцию.	подстройка модели прав и публикаций под CMS.
Git-only	редактирует один технический владелец.	нельзя безопасно делегировать старостам без доступа к репозиторию.

11. как статический резерв сливается с публикациями

публичная страница начинает со встроенного редакционного резерва, затем запрашивает таблицу публичных снимков. специальная функция overlay сводит два источника в один список.

что делает публичная страница при открытии



Supabase недоступен

показать резерв

published + source_key

заменить старую запись

archived + source_key

скрыть старую запись

что такое source_key

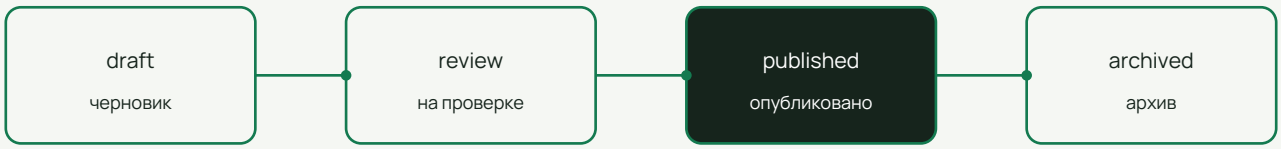
это неизменяемый технический идентификатор исходной статической записи, например `legacy:lecture:<slug>`. он связывает публикацию из базы с её резервной версией и предотвращает дублирование.

ответ базы	поведение
ошибка или сеть недоступна	показывается статический резерв. сайт остаётся полезным.
published с source_key	публикация заменяет соответствующую статическую запись.
published без source_key	появляется новая запись, которой не было в исходниках.
archived с source_key	техническая tombstone-запись подавляет старый статический материал.
archived без source_key	новая запись исчезает из публичного результата.

компромисс отказоустойчивости

если Supabase недоступен, браузер временно не узнает даже об архивной отметке и покажет резерв. это сознательный выбор в пользу доступности. для срочного юридического удаления пришлось бы также убрать запись из статического слоя и пересобрать сайт.

12. редакционный процесс и безопасность



редактор сохраняет черновик или отправляет его на проверку. публикация разрешается сервером по роли, разделу и области. при публикации создаётся отдельный snapshot; последующие правки черновика не меняют публичную версию до нового утверждения.

где реально проверяются права

- владелец (owner) имеет все права и единственный назначает системных администраторов. системный администратор (coordinator) управляет редакцией по всему ИГЭУ, но не меняет владельца и не назначает других администраторов.
- RPC-функции проверяют статус, владельца, версию записи и допустимые поля.
- староста - editor с временным назначением группы: проверяет и публикует в ней. обычный editor получает точные права на раздел, действие и область. право на расписание назначается отдельно.
- помощник (contributor) редактирует свои черновики и отправляет старосте. приглашённый editor группы может менять материалы других авторов в разрешённых разделах, но публикацию утверждает староста.
- история версий создаётся серверными триггерами; прямое изменение истории запрещено.
- оптимистическая блокировка через `updated_at` не даёт молча затереть чужую свежую правку.

ключи и токены

данные	можно в браузер?	почему
publishable key	да	это публичный идентификатор клиента. безопасность обеспечивают RLS и grants.
access/refresh token	да, но осторожно	подтверждает сессию пользователя; сейчас хранится в localStorage.
service-role key	никогда	обходит RLS и даёт привилегированный доступ.

текущий компромисс localStorage

для полностью статической админки хранение токенов в localStorage просто, но при XSS вредоносный JavaScript смог бы их прочитать. поэтому критичны отсутствие небезопасного HTML, строгая обработка ссылок и зависимости без подозрительных скриптов. более сильная модель - собственный backend/BFF с HttpOnly Secure cookies, но она отменяет часть простоты static-only.

что добавлено к первоначальной версии

конспект и редакция

конспект поддерживает текст, ссылки, формулы KaTeX, таблицы, фотографии, редактируемые рисунки и десять видов структурированных схем. локальный черновик и изображения хранятся в IndexedDB. синхронизация отправляет изображения в закрытый Storage и сохраняет серверный черновик, но не публикует его. PDF и произвольные файлы через редакцию не загружаются. предпросмотр и печать в PDF не сохраняют запись.

команда группы и личные конспекты

староста создаёт до пяти помощников со случайными логинами и приглашает до пяти редакторов по личному email. делегирование прекращается при отзыве или окончании назначения. личные конспекты доступны без входа, остаются в браузере и переносятся файлом .potok с фотографиями и исходниками рисунков. очистка данных сайта удаляет локальные записи.

справка и расписание

справка имеет пять аудиторий: группа, курс факультета, факультет, курс всего ИГЭУ и весь университет. право на конкретную область назначается отдельно. изменения расписания требуют отдельного доступа к группам. поддерживаются перестановки, переносы, замены, отмены и добавления; делегированный редактор отправляет их на утверждение старосте.

установка, связь и внешний вид

pWA запускается в актуальном, сохраняет офлайн-копию и различает связь с сайтом и доступность свежих данных. уведомления добровольные. в настройках есть светлая, тёмная и системная тема, размер текста и уменьшение движения; значения сохраняются в браузере.

предложения и метрики

публичная кнопка предложения материала открывает Telegram автора с контекстом пары. отправка сообщения остаётся действием пользователя; публикация требует редакционной проверки. аналитика работает после согласия: собственные события Supabase и Яндекс Метрика. малые когорты скрываются; число браузеров не равно точному числу студентов. закрытый отчёт и отправка push доступны владельцу.

постоянные адреса и миграции

новый конспект получает номер из БД. утверждение снимка само по себе не создаёт новый HTML: нужен полный build и доставка out/. старые SQL-миграции не переписывают. до общего db push нужно сверить локальную и удалённую историю: известные расхождения описаны в ADMIN.md.

> этот гид описывает рабочий код на 4 октября 2026. он не является подтверждением текущего состояния хостинга или успешного свежего QA. результаты и ограничения проверок - в docs/QA.md. содержательная достоверность материалов проверяется отдельно; состояние кода не подтверждает первичные учебные источники.

13. проверки качества

ни один вид теста не доказывает всё. проект использует несколько слоёв, каждый ловит свой класс ошибок.

инструмент	что проверяет	что не заменяет
TypeScript	структуру данных, аргументы функций и согласованность типов.	поведение в реальном браузере.
ESLint	опасные или нестабильные паттерны React/Next.js и стиль кода.	функциональные сценарии.
Vitest + jsdom	быстрые unit- и component-тесты, overlay, маршруты, контент.	полный движок браузера и реальную навигацию.
Playwright	chrome/Firefox, мобильные размеры, клики, ссылки, переполнения, фокус.	логику базы данных.
production build	совместимость static export, генерацию страниц и финальную упаковку.	ручную оценку смысла контента.
SQL-тесты	RLS, RPC, импорт, anon/editor-доступ и транзакционные границы.	внешний вид интерфейса.

почему тесты особенно важны здесь

у проекта есть необычные русские маршруты, статическая локализация, fallback-overlay, сложные права базы и mobile-first интерфейс. это именно те границы, где ручная проверка быстро становится ненадёжной.

что значит «сборка прошла»

это означает, что исходники удалось превратить в production-файлы. но сборка не гарантирует, что ссылка ведёт туда, куда надо, кнопка доступна с клавиатуры, редактор не видит чужую группу или публикация правильно заменяет резерв. для этого существуют остальные слои.

> правильная формула готовности: типизация + lint + unit + build + e2e + SQL-проверки + визуальная и содержательная проверка.

14. сравнение архитектурных вариантов

вариант	сложность	надёжность	редакция	когда выбирать
текущий гибрид	средняя	высокая для публичной основы	Supabase	сейчас: малый продукт, обычный хостинг, делегирование.
только статические файлы	низкая	очень высокая	через Git/агента	один редактор, редкие обновления.
React SPA + API	средняя	зависит от API	любая	приложение почти целиком интерактивное.
Next.js SSR	выше средней	зависит от сервера	встроенная или внешняя	SEO и данные должны формироваться на каждый запрос.
свой backend	высокая	контролируемая	собственная	много интеграций, секретов, сложные процессы.
headless CMS	средняя	зависит от CMS	готовая	большая контентная команда и стандартный workflow.

почему текущий вариант разумен

- публичный сервис не должен падать вместе с редакцией.
- нет необходимости оплачивать и обслуживать постоянно работающий Next.js-сервер.
- старостам можно делегировать точные области, не выдавая Git-доступ.
- переход к более серверной модели остаётся возможным позже.

цена гибрида

- два источника правды требуют source_key, overlay и чётких правил миграции.
- некоторые обновления видны только после запроса Supabase, поэтому есть состояние догрузки.
- статическая админка вынуждена работать с клиентскими токенами.
- русские маршруты требуют отдельного post-build шага и тестов.

главный критерий будущих решений

не «что технологичнее», а какую новую пользовательскую задачу невозможно разумно решить текущей моделью. сервер, CMS или мобильное приложение должны появляться из конкретной потребности, а не из желания усложнить стек.

15. риски и направления развития

риск	почему важен	что делать
XSS и токены в localStorage	украденный токен даёт права пользователя.	CSP, безопасный рендеринг, аудит зависимостей; при росте - BFF и HttpOnly cookie.
расхождение static и DB	резерв может показывать старую или архивную запись при сбое сети.	чёткий source_key, тесты overlay, процедура аварийного удаления и периодическая пересборка.
обновления зависимостей	новая версия может неожиданно изменить поведение.	версии уже закреплены точно; lockfile, контролируемые обновления и CI защищают обновления.
ручной деплой	можно загрузить не весь out/ или смешать версии.	автоматизировать публикацию каталога целиком и добавить smoke-check после деплоя.
рост схемы прав	сложные RLS/RPC трудно держать в голове.	SQL-тесты, документация ролей, минимальные grants и регулярный security review.
зависимость от внешнего расписания	источник может изменить формат.	fail-safe синхронизация, мониторинг и явная дата снимка.

логичная эволюция

- CI уже выполняет test, lint, build и Chromium audit. доставку целого out/ и живые URL нужно проверять после публикации.
- затем: мониторинг доступности Supabase и сроков актуальности расписания.
- история версий и редакторские проверки уже есть; новые процессы добавляются по конкретному запросу.
- при появлении секретных интеграций: небольшой BFF/API, не обязательный полный перенос сайта.
- при высокой динамичности: пересмотреть static fallback и решить, нужен ли SSR/ISR.

> хорошая архитектура - не та, где больше технологий. хорошая архитектура делает текущий продукт понятным и надёжным, а следующий шаг - возможным.

16. словарь без программистского тумана

термин	что означает
frontend	код и интерфейс, которые пользователь видит и запускает в браузере.
backend	серверная логика, база, авторизация и защищённые операции. здесь эту роль во многом выполняет Supabase.
framework	набор правил и инструментов, который задаёт устройство приложения. Next.js - framework для React.
static export	сборка сайта в готовые файлы без обязательного application server.
hydration	подключение React-поведения к уже показанному HTML.
component	самостоятельная часть интерфейса со своими данными и поведением.
state	данные, которые меняются во время работы интерфейса: выбранная группа, открытая форма, сессия.
API	формальный способ одной программы запросить данные или действие у другой.
PostgreSQL	реляционная база данных, где структура и связи задаются таблицами и SQL.
RLS	правила на уровне базы, ограничивающие строки для конкретного пользователя.
RPC	вызов серверной функции как операции API.
snapshot	зафиксированная опубликованная версия, отделённая от изменяемого черновика.
fallback	резервный вариант, используемый при отсутствии свежих данных.
source_key	мост между статической записью и её редакторской версией в базе.
migration	версионированное изменение структуры или данных базы.
E2E-тест	проверка реального пользовательского сценария в настоящем браузере.

17. как внести изменение

изменение в «потоке» начинается не с кода, а с ответа на два вопроса: какую задачу решаем и какой слой системы за неё отвечает. это не даёт правке случайно расползтись по всему проекту.

карта основных мест

что меняется	где смотреть	что не забыть
публичная страница	src/app и компоненты	мобильная версия, маршруты, клавиатура.
адрес	routes.ts, localize-routes.mjs, rewrites	обновить все три точки и тесты.
контент	админка; исходники — legacy-резерв	публикация не равна сохранению черновика.
права	миграции, RLS, RPC	интерфейс не заменяет серверную проверку.
расписание	официальный снимок и отдельные поправки	проверять каталог, права группы, конфликты и календарные файлы.

перед публикацией

- проверить типы, lint, unit- и component-тесты.
- пройти ключевой сценарий в реальном браузере.
- собрать production-версию и убедиться, что русские адреса созданы.
- загружать на Timeweb весь свежий `out/`, а не отдельные файлы.
- после публикации проверить разделы, числовой конспект, файлы, календарь и 404. Git push не доказывает доставку сайта.

> главное: менять нужно минимальный слой, но проверять весь путь до публичного сайта.

основано на текущих исходниках и документации проекта: PROJECT_CONTEXT.md, apps/web/README.md, docs/ADMIN.md, docs/URLS.md, docs/SCHEDULE_EDITOR.md, docs/ANALYTICS.md, next.config.ts, routes.ts, package.json и SQL-миграциях Supabase.