BACKEND РАЗРАБОТЧИК

Чеклист Backend разработчика

Практическая база для backend-разработки: API, данные, транзакции, безопасность, производительность, наблюдаемость, деплой и эксплуатация.

Архитектура

Backend начинается с понимания домена, границ ответственности, контрактов и того, где система должна быть простой, а где масштабируемой.

Определить границы сервиса

Понять, за что сервис отвечает, а что остается вне его ответственности.

Что это дает: Четкие границы уменьшают связность, упрощают поддержку и помогают избежать сервиса, который знает слишком много.

Как выполнить:

1. Опишите доменные сущности и операции сервиса.
2. Зафиксируйте внешние зависимости.
3. Согласуйте, какие данные сервис хранит сам, а какие получает извне.

Критерии приемки:

- Границы описаны.
- Зависимости перечислены.
- Команда понимает, где источник истины.

Зафиксировать архитектурное решение

Описать важный технический выбор в ADR: контекст, варианты, решение и последствия.

Что это дает: ADR сохраняет инженерный контекст и помогает будущей команде понимать, почему выбран именно этот путь.

Как выполнить:

1. Опишите проблему и ограничения.
2. Сравните 2-3 варианта.
3. Зафиксируйте выбранное решение и последствия.

Критерии приемки:

- ADR опубликован.
- Решение связано с задачей.
- Последствия и trade-off описаны.

Спроектировать модель ошибок

Определить типы ошибок, формат ответа и правила логирования.

Что это дает: Единая модель ошибок упрощает frontend-интеграцию, поддержку и диагностику production-инцидентов.

Как выполнить:

1. Разделите validation, auth, business и system errors.
2. Опишите формат error response.
3. Не раскрывайте чувствительные детали во внешнем ответе.

Критерии приемки:

- Формат ошибок документирован.
- Коды ошибок стабильны.
- Логи содержат диагностический контекст.

API и интеграции

API должен быть предсказуемым для клиентов: с понятными ресурсами, статусами, ошибками, версиями и документацией.

Описать API-контракт

Зафиксировать endpoints, параметры, схемы ответов, ошибки и авторизацию.

Что это дает: API-контракт снижает трение между backend, frontend, QA и внешними интеграторами.

Как выполнить:

1. Опишите ресурсы и методы.
2. Добавьте request/response schemas.
3. Укажите status codes, ошибки и требования авторизации.

Критерии приемки:

- OpenAPI или аналог опубликован.
- Контракт покрывает успешные и ошибочные ответы.
- Frontend и QA используют документацию.

Настроить валидацию входных данных

Проверять входные данные на границах системы до бизнес-логики.

Что это дает: Валидация защищает доменную модель, снижает количество неожиданных ошибок и улучшает пользовательскую обратную связь.

Как выполнить:

1. Опишите обязательные поля, типы, длины и форматы.
2. Разделите syntactic и business validation.
3. Верните понятные ошибки для клиента.

Критерии приемки:

- Некорректные данные не попадают в бизнес-логику.
- Ошибки валидации имеют стабильный формат.
- Покрыты граничные значения.

Продумать версионирование API

Определить, как будут выпускаться несовместимые изменения и поддерживаться клиенты.

Что это дает: Версионирование снижает риск сломать мобильные приложения, партнерские интеграции и старый frontend.

Как выполнить:

1. Определите правила backward compatibility.
2. Выберите подход к версиям: URL, header или contract version.
3. Подготовьте deprecation policy.

Критерии приемки:

- Правила версионирования описаны.
- Breaking changes не ломают текущих клиентов без плана.
- Deprecation communicated.

Данные

Работа с данными требует аккуратности: схема, индексы, миграции, консистентность и восстановление часто важнее красивого кода.

Спроектировать схему данных

Определить таблицы, связи, ограничения, индексы и правила владения данными.

Что это дает: Хорошая схема данных сохраняет консистентность и упрощает запросы, миграции и развитие продукта.

Как выполнить:

1. Опишите сущности и связи.
2. Добавьте constraints: not null, unique, foreign keys.
3. Проверьте запросы, которые будут самыми частыми.

Критерии приемки:

- Схема покрывает бизнес-правила.
- Индексы соответствуют запросам.
- Есть план миграции.

Писать безопасные миграции

Планировать изменение схемы так, чтобы не сломать production и не остановить сервис надолго.

Что это дает: Безопасные миграции снижают риск downtime, блокировок таблиц и потери данных.

Как выполнить:

1. Разделяйте destructive changes на несколько релизов.
2. Проверяйте время выполнения на похожем объеме данных.
3. Готовьте rollback или forward-fix план.

Критерии приемки:

- Миграция протестирована.
- Нет долгих блокировок без плана.
- Rollback/forward-fix описан.

Анализировать медленные запросы

Находить запросы, которые замедляют API, блокируют базу или плохо масштабируются.

Что это дает: Оптимизация запросов улучшает latency, снижает нагрузку на базу и уменьшает риск отказов при росте трафика.

Как выполнить:

1. Соберите slow query log.
2. Проверьте план выполнения через EXPLAIN.
3. Добавьте индекс или измените запрос только после проверки.

Критерии приемки:

- Медленные запросы измерены.
- EXPLAIN показывает улучшение.
- Нет лишних индексов без пользы.

Безопасность

Безопасность backend держится на корректной аутентификации, авторизации, валидации, защите секретов и безопасной обработке ошибок.

Реализовать надежную аутентификацию

Проверить, что система корректно устанавливает личность пользователя или сервиса.

Что это дает: Authentication является основой доступа: если она сломана, остальные проверки безопасности теряют смысл.

Как выполнить:

1. Используйте проверенные механизмы: sessions, OAuth2, JWT по необходимости.
2. Настройте срок жизни токенов и refresh flow.
3. Защитите brute force и credential stuffing.

Критерии приемки:

- Auth flow описан.
- Секреты не попадают в код.
- Неуспешные попытки контролируются.

Проверить авторизацию на уровне ресурса

Гарантировать, что пользователь может работать только с разрешенными ему данными.

Что это дает: Authorization защищает от IDOR и утечек данных между пользователями, командами и организациями.

Как выполнить:

1. Проверяйте доступ не только к endpoint, но и к конкретному объекту.
2. Покройте тестами чужие ID и разные роли.
3. Держите правила доступа рядом с доменной логикой.

Критерии приемки:

- Чужие ресурсы недоступны.
- Роли и permissions описаны.
- Есть тесты на отрицательные сценарии доступа.

Защитить секреты и конфигурацию

Хранить ключи, пароли и токены вне репозитория и логов.

Что это дает: Утечка секретов часто приводит к компрометации данных, инфраструктуры или платежных систем.

Как выполнить:

1. Используйте env и secret manager.
2. Проверяйте, что секреты не логируются.
3. Настройте rotation для важных ключей.

Критерии приемки:

- Секретов нет в репозитории.
- Production secrets хранятся отдельно.
- Есть процесс замены ключей.

Эксплуатация

Сервис должен быть не только написан, но и наблюдаем, диагностируем, безопасно выкатываем и понятен тем, кто будет поддерживать его после релиза.

Добавить структурированные логи

Логировать важные события с request id, пользователем, операцией и контекстом ошибки.

Что это дает: Структурированные логи позволяют быстро расследовать инциденты и связывать действия между сервисами.

Как выполнить:

1. Добавьте correlation id.
2. Логируйте бизнес-события и ошибки.
3. Исключите персональные данные и секреты.

Критерии приемки:

- Логи можно фильтровать.
- Есть request/correlation id.
- Чувствительные данные не попадают в лог.

Настроить health checks

Сделать проверки состояния сервиса и зависимостей для мониторинга и оркестрации.

Что это дает: Health checks помогают балансировщику и команде понять, жив ли сервис и готов ли он принимать трафик.

Как выполнить:

1. Разделите liveness и readiness.
2. Проверяйте критические зависимости осторожно.
3. Не делайте health endpoint тяжелым.

Критерии приемки:

- Health endpoint работает.
- Readiness учитывает критические зависимости.
- Мониторинг использует checks.

Подготовить runbook сервиса

Описать, как деплоить, диагностировать, откатывать и поддерживать сервис.

Что это дает: Runbook снижает зависимость от одного разработчика и ускоряет реакцию на инциденты.

Как выполнить:

1. Опишите основные команды и dashboards.
2. Добавьте типовые симптомы и действия.
3. Укажите владельцев и каналы эскалации.

Критерии приемки:

- Runbook доступен команде.
- Есть инструкция rollback.
- Новый участник может выполнить базовую диагностику.