

Данный алгоритм можно использовать при каждом проектировании API.

При этом, в разном контексте API требования могут отличаться (разработка API для блокчейн-биржы отличается от API Госуслуг).

1. Сбор требований: начните со сбора User Story и диаграмм бизнес-процессов, которые помогут вам понять контекст, цели и ограничения интеграции.
2. Выделение функциональных требований: проанализируйте диаграммы бизнес-процессов, User Story и определите функциональные требования, необходимые для интеграции, такие как обмен данными, синхронизация или преобразование.
3. Выделите важные нефункциональные требования для создания качественной интеграции.
 1. Учтите необходимость масштабирования, производительности и безопасности.
 2. Система stateless или stateful (грубо - нужно ли хранить на сервере данные о запросах, или нет?).
4. Выберите подходящий архитектурный стиль и протокол.
 1. Рассмотрите использование REST, GraphQL или gRPC, в зависимости от вашей ситуации и предпочтений.
 2. Выберите подходящий протокол для передачи данных (HTTP, HTTPS, TCP, WebSockets и т.д.).
5. Анализ потока данных: создайте диаграммы потоков данных, которые показывают, как данные перемещаются между системами, и определите закономерности потоков данных.
6. Определите формат обмена данными (JSON, XML, Protobuf).
7. Разбейте ваш API на логические ресурсы и коллекции, либо эндпоинты если это RPC технология.
8. Разработка сценариев интеграции (Use Case): на основе потоков данных определите сценарии использования, которые описывают, как системы будут взаимодействовать в конкретных сценариях, обеспечивая охват всех возможных ситуаций.
9. Создание диаграмм деятельности UML: сопоставьте сценарии использования с диаграммами деятельности. унифицированного языка моделирования (UML), которые обеспечивают четкую визуализацию последовательностей действий в каждом сценарии.
10. Определите методы, которые будут использоваться (для REST) и данные (параметры) запросов и ответов.
 1. Учтите варианты пагинации, сортировки и фильтрации данных.
 2. Организуйте проверку входящих данных на валидность и целостность.
11. Обработка ошибок и коды состояния.

1. Возвращайте подходящие HTTP коды состояния для успешных и неуспешных запросов.
2. Предоставляйте информацию об ошибках в структурированном формате (JSON, XML и т.д.).

12. Безопасность и авторизация.

1. Реализуйте аутентификацию и авторизацию пользователей (OAuth, API ключи, JWT и т.д.).
2. Защитите ваш API от распространенных угроз и атак (SQL-инъекции, XSS, CSRF и т.д.).
3. Учтите применение политик безопасности (CORS, Content Security Policy и т.д.).

13. Рассмотрите инструменты управления производительностью, кеширование, rate limit.

14. Версионирование API.

1. Планируйте изменения и обновления API, предусматривая версионирование.
2. Используйте соглашения по именованию версий в URL или заголовках запросов.

15. Документация.

1. Создайте четкую и понятную документацию для вашего API, описывающую методы, параметры, коды состояния и ошибки и всю другую необходимую информацию. Также приложит примеры запросов и ответов.
2. ЗадOCUMENTИРУЙТЕ ваш веб-сервис.

16. Обратная связь и поддержка.

1. Обеспечьте каналы связи для получения обратной связи от пользователей и разработчиков, использующих ваш API.
2. На основе отзывов улучшайте ваше API.
3. Настройте мониторинг вашего работающего API, чтобы быстро получать информацию о проблемах.