**API (Application Programming Interface)** - Описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой

Юзеры взаимодействуют с Frontend через кнопки, ползунки, поля. Это все интерфейсы для взаимодействия программы с пользователем. Frontend в свою очередь взаимодействует с Backend, но ему не нужны кнопки и ползунки, ему нужны endpoints.

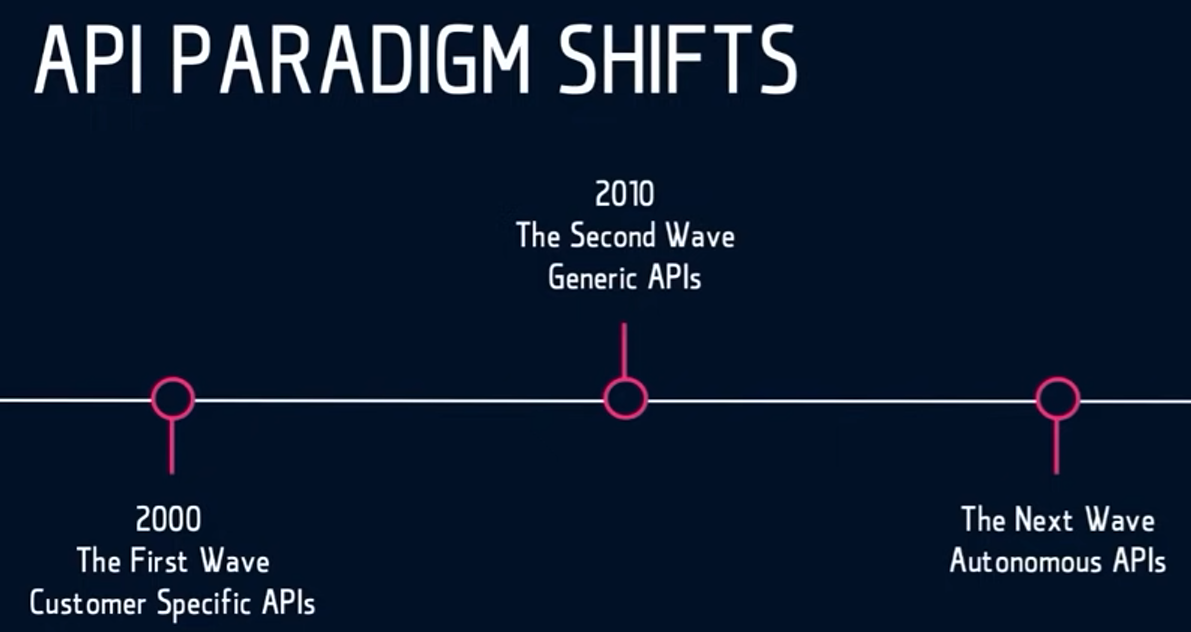
* По сути Frontend общается с backend через API (посылает запросы к каким то endpoints – “**somesite.ru/api/v1/users/createuser**”)
* Backend общается с другим backend этого приложения через API (посылает запросы к каким то endpoints – “**api/backend/calculation**”
* Backend нашего приложения общается с backend другого приложения в сети через API (посылает запросы к каким то endpoints
  + **http://api.cdek.ru/calculator/calculate\_price\_by\_json.php**
  + **https://www.express.ru/api/office/calculateFragile=FalseItems=1box\_height=1box\_length=30box\_width=20cargo\_form=2cargo\_type=2country\_from=Россияcountry\_to=Россияdistrict\_from=district\_to=fias\_from=6600000100000fias\_to=7700000000000officeRegion=Москваplace\_from=Екатеринбург гplace\_to=Москва гregion\_from=Свердловская облregion\_to=weight=0.5lWAaIEaG98HmSuV9CYawh7BWJ4fIeRn1**
  + **http://79.174.68.35:8099/JSONWebService.asmx/Calc**

Если нужно узнать, что наш бэк отправляет другому бэку или компании-интеграции. То нужно смотерть дамп работы этого бэка. Под бэком понимается какой то микросервис или монолит. Если мы хотим посмотреть что бэк шлет клиенту (FrontEnd), то смотрим DevTools. Если же бэк шлет что то другому бэку, то смотрим дампы на сервере. По сути мы можем послать и через POSTMAN. Только нужно слать на нужный endpoint. Frontend шлет на первый бэк запрос на создание отправления. Бэк уже шлет другому бэку и тд. И на каждом бэке находятся свои ендпоинты, вплоть до ендпоинта компании-интеграции.

**Разница** **между Веб-сервисом и API** в том, что API описывает способы межпрограммного обеспечения, API может быть не только у Веб-сервиса, но и у интернет-протокола или, например, у функций операционной системы.

**Веб-сервис** - это набор стандартов или протоколов для обмена информацией между двумя приложениями или устройствами. Различные приложения используют разные языки и технологии. Этим приложениям сложно общаться друг с другом. Веб-сервисы предоставляют решение этой проблемы. Он создает общую платформу для различных приложений для обмена информацией друг с другом. Например, приложение PHP может взаимодействовать с приложением .NET или Java через веб-сервис.

Веб-сервис предоставляет информацию в RAW виде (только данные, без UI). Например JSON если это REST или XML если это SOAP. Эти данные могут быть использованы различными разработчиками, для создания различных друг от друга приложений. По сути веб-сервисы это способ передачи сырых данных (RAW data) используя веб технологии (HTTP, SMTP, UDP, …). Например API сервера который содержит в себе данные о походе может быть использован и как для создания обычного UI приложения, где будет показываться погода на сегодня, так и те же RAW данные, переданные в JSON формате, могут быть использованы для программиров ания систем умного дома, дабы включать отопление, открывать окна и тд.





**API определяет интерфейс, предоставляющий данные сервиса другим приложениям**. Выбор правильного формата API крайне важен. Бизнес не всегда учитывает все факторы, выбирая формат. В результате не хватает возможности для добавления новых фич, которые могут понадобиться в дальнейшем.

Первую группу API, которую мы рассмотрим, можно выделить как Request-Response API. Основные отличия данной группы:

* Интерфейс предоставляется через веб-сервер на основе HTTP протокола.
* API определяет набор эндпоинтов.
* Клиенты отправляют запросы для работы с данными(получить/удалить/изменить) на данные эндпоинты.
* Ответ в формате JSON или XML.

Самые популярные request-response API: REST, SOAP, RPC и GraphQL.

Which API pattern fits your use case best?

Every API project has different requirements and needs. Usually, the architectural choice depends on

* the programming language in use,
* the environment in which you’re developing, and
* the resources you have to spare, both human and financial.

Knowing all the tradeoffs that go into each design style, API designers can pick the one that’s going to fit the project best.

With its tight coupling, **RPC** works for internal microservices but it’s not an option for a strong external API or an API service.

**SOAP** is troublesome but its rich security features remain irreplaceable for billing operations, booking systems, and payments.

**REST** has the highest abstraction and best modeling of the API. But it tends to be heavier on the wire and chattier – a downside if you’re working on mobile.

**GraphQL** is a big step forward in terms of data fetching but not everyone has enough time and effort to get the hang of it.

At the end of the day, it makes sense to try a few small use cases with a particular style, and see if it fits your use case and solves your problems. If it does, try expanding and see if it fits more use cases.