Talent Acceleration Program

Workbook

tap.kiitos.tech

Week 1

Nothing is impossible. The word itself says "I'm possible".

Welcome to **week 1** of the Talent Acceleration Program! Your journey to becoming a professional software developer starts now.

On the *Technical Skills* side, you will start at the foundation of both Frontend and Backend development.

- For Frontend, you will learn about the rise of frontend frameworks, the differences between a website and web application, static vs. dynamic pages and finally get an introduction to React (the current leading frontend framework)
- The Backend specialization will learn more about the key roles the backend fulfills: Application Programming Interfaces (API), the architectural pattern Representational State Transfer (REST) and lastly, database management (and its most important operations: CRUD).

On the *Soft Skills* side you'll start off by working on your mindset, the place where all growth starts. Important topics that we'll cover are growth vs. fixed, locus of control, antifragility, self-image & proactivity.

Good luck!

The TAP Team

Part 1: Technical Skills

Jump ahead to Frontend or Backend

Frontend Track

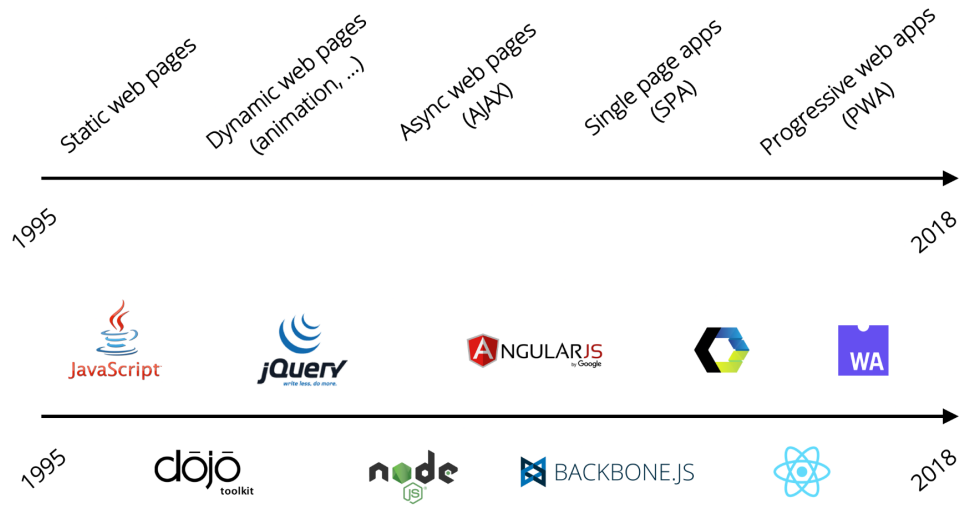
Week	Topics
1	Introduction to Specialization
Frontend	Frontend Frameworks
Backend	APIs, REST & DBMS
2	Core Concepts Specialization
3	Agile Fundamentals
4	Project Management Fundamentals

Learning goals:

- ❏ Learn about the evolution of frontend development
- ❏ Grasp the difference between websites and web applications
- ❏ Understand the 2 types of web content: static and dynamic
- ❏ Get introduced to the most popular Frontend Framework: *React.js*

FRONTEND DEVELOPMENT ROADMAP

The field of frontend development is ever-evolving. However, this is only a recent phenomenon. It was only until 2009 when things got interesting. Node.js was born and JavaScript could now also be executed on the server side!



From that time on an enormous amount of frontend tools has been created. Unit testing JavaScript code started to become important. AngularJS came out and we are seeing more technologies that help us create single page applications (SPAs). Nowadays we even have Progressive web apps (PWA), micro frontends and WebVR!

With these innovations nowadays it's necessary to become familiar with at least 1 modern framework (i.e. : React.js, Vue.js or Angular) in order to even get started as a professional frontend developer. And that's exactly what you'll be learning in the following weeks!

Learning Materials

- Frontend Roadmap
- A Brief History of Frontend Frameworks

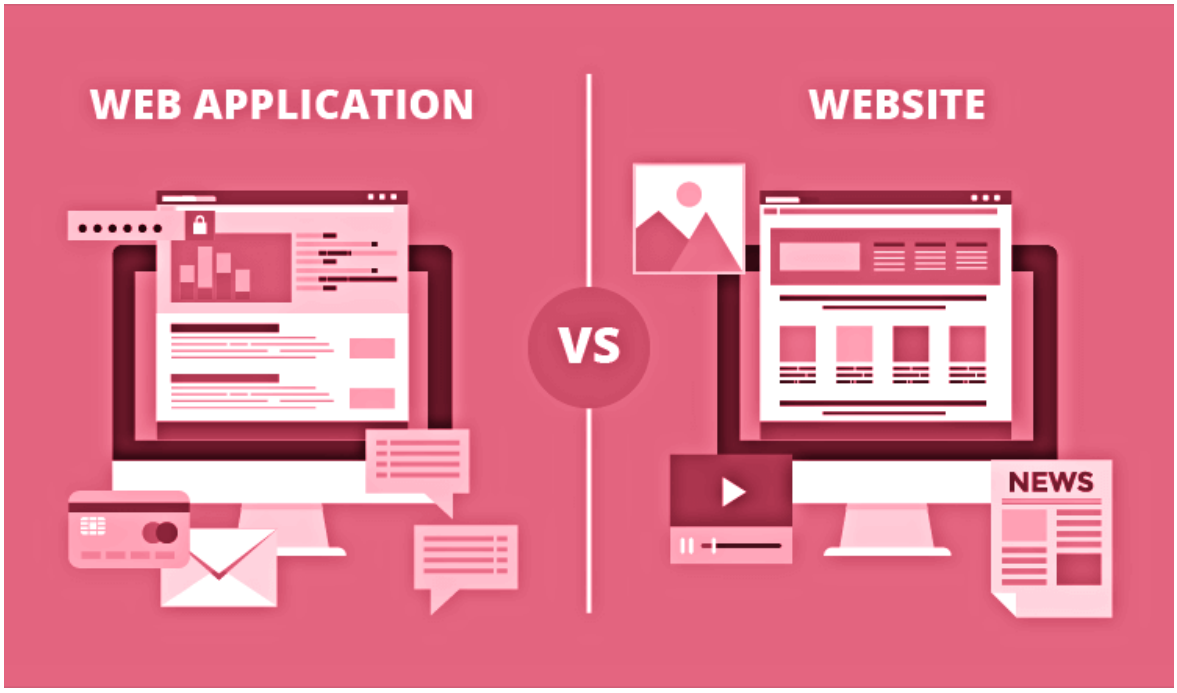
WEBSITES VS. WEB APPLICATIONS

Before diving into frontend frameworks, it's worth mentioning the thing we're actually using it for: websites. Or is it web applications?

On a technical level there's a difference between those two and it's important to know what those are. Why? Knowing the difference will allow you to choose what type of application you need to build in order to address the client's actual needs.

Generally speaking, websites are simpler:

- From the user's perspective, they are meant to be *informational* only, with little to no way to interact (and thus no way to change the data shown). Common examples of websites are Wikipedia or Google
- From the developer's perspective, they are usually a collection of HTML, CSS and JavaScript files. They are served by a web server that has the single job of serving web assets only



Web applications, on the other hand, allow for more complexity:

- From the user's perspective, they are meant to be *interacted* with and are designed to be fully fledged digital products. Common examples of web applications are Facebook or Google Docs
- From the developer's perspective, they are way more complex and usually require a database, API and frontend framework to handle state changes.

Learning Materials

- Website vs Web App

Another distinction worth knowing about is the division between static and dynamic websites. The terms static and dynamic refer to the type of data the website provides.

- **Static websites** contain data that *does not change*. There is only one state and it's always the same. The primary purpose of static websites is to be informational.
- **Dynamic websites** contain data that *changes* depending on user interaction. The primary purpose of dynamic websites are meant to be interacted with.

Learning Materials

- Static vs. Dynamic Websites

A final distinction worth making is the difference between the Multiple Page Application (MPA) and Single Page Application (SPA).

On the surface these terms are confusing: don't SPAs also contain multiple pages? They do, but that's not what the term is referring to. When we talk about single versus multiple, we're talking about HTML files.

In a SPA, there's only one HTML file: *the index.html*. When a client requests for a SPA, the server sends back the single index.html file, which will then render the complete web application once the browser executes the JavaScript.

In a MPA, there's an HTML file corresponding to every route the web application has: a request for */about* will have the server send back an *about.html* file.

Learning Materials

- Dynamic Websites vs Static Pages vs Single Page Apps (SPAs)
- SPA vs MPA: What's the difference?
- What Is a Single Page Application (SPA)?

GETTING STARTED: REACT.JS

In order to get you started with learning about modern frontend development, we'll start by focusing on the leading technology today: React.js!

React is a technology that allows us to create a SPA. Strictly speaking, it's not a framework but a JavaScript library with only one job: *to build modular user interfaces*. It's a library because React is concerned only with rendering the UI and leaves other modern frontend features out of the picture (i.e. file

bundling, unit testing, routing, etc.), leaving it up to you to decide how you want to handle these things. However, it's treated as a framework because React does force you to "think in the React way"!

Learning Materials

- React in 100 Seconds
- A High Level Overview of React
- React: The Big Picture

Backend Track

Week	Topics
1	Introduction to Specialization
Frontend	Frontend Frameworks
Backend	APIs, REST & DBMS
2	Core Concepts Specialization
3	Agile Fundamentals
4	Project Management Fundamentals

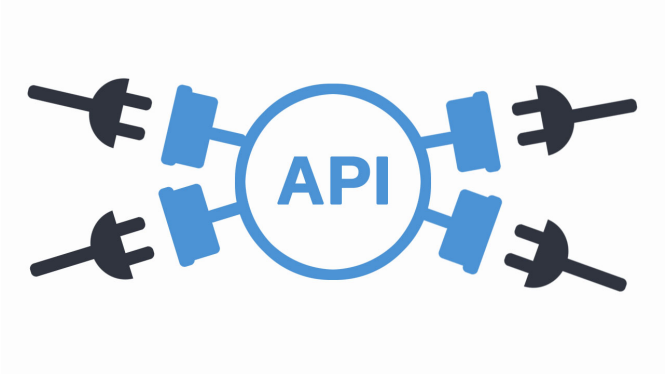
Learning goals:

- ❏ Learn about the essential role of Application Programming Interfaces (API)
- ❏ Understand the architectural pattern Representational State Transfer (REST)
- ❏ Recognize the importance of database management

CORE CONCEPT: APPLICATION PROGRAMMING INTERFACE

One of the most important concepts in backend development is called the application programming interface (or API for short).

As the name indicates, it refers to an *interface* that applications can connect with in order to communicate with each other. The API is the label used for the code that specifies *where* a frontend can interact with a backend and is usually defined by an architectural pattern such as REST, SOAP or GraphQL.



We don't want a frontend to directly interact with the database or web server, because *this leaves the backend exposed for threats*. Instead we want to define exactly in what ways clients (whether they be browsers, command line interfaces or otherwise) can interact with it!

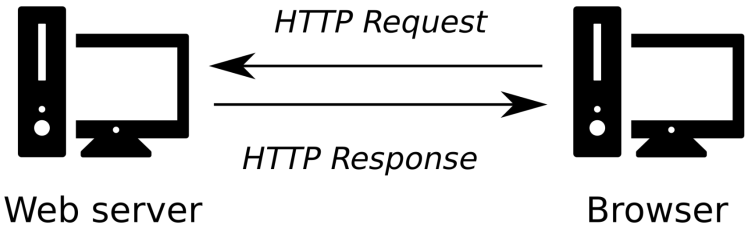
Learning Materials

- What is an API?
- RESTful APIs in 100 Seconds
- APIs for Beginners

CORE CONCEPT: REPRESENTATIONAL STATE TRANSFER

In order to build APIs that can be used by as many users, it's important to have a *standardized* way of building them. If everyone builds them in the same way, it will be a lot easier to connect with them!

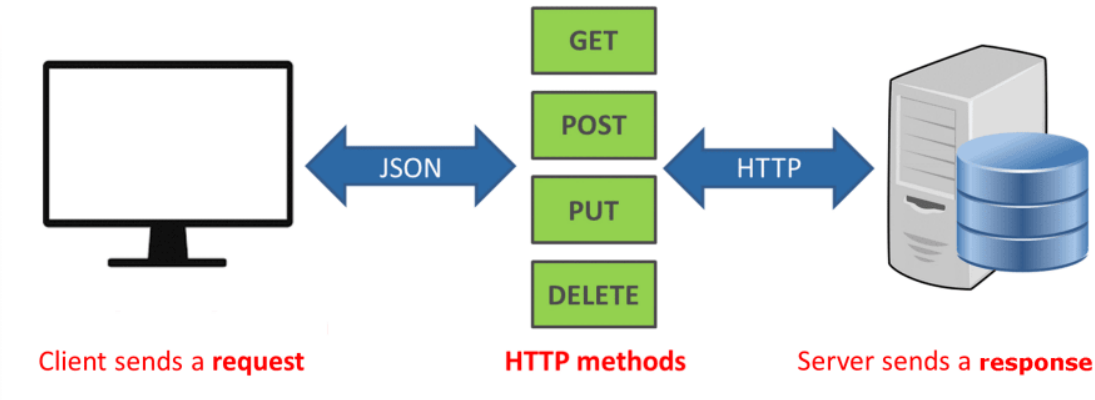
One of the most popular ways of doing this today is by using Representational State Transfer (or REST for short). In this approach, we build applications whereby the *client and server are independent from each other*: they are 2 separate applications that communicate with each other through an API using requests and responses.



The idea of REST is implemented in code through Hypertext Transfer Protocol (or HTTP). This defines the communication standard an API has to conform to in order to be able to understand requests (also known as HTTP/network/API requests) from a client.

Concretely, this takes shape in the following HTTP methods:

- GET: retrieve data from the server
- POST: upload data to the server
- PUT: update data in the server
- DELETE: delete data from the server



Learning Materials

- What is REST?
- HTTP Made Easy: Understanding the Web Client-Server Communication
- REST API Best Practices

CORE CONCEPT: DATABASE MANAGEMENT

Any modern application needs data storage. For example, it wouldn't be possible to have an account at any social media platform (i.e. Facebook) without it. You'd have to create a new account every time you wanted to use it!

Data storage is done by a database (or DBMS, which stands for database management system), an organised collection of structured data stored in a computer. In order to communicate with it a request from the client has to go through the API.

The database, generally speaking, performs *4 essential functions*. Let's take a simple example of a UserAccount to illustrate these:

- Create: when using Facebook I want to be able to create a UserAccount
- Read: when logging in I want to be able to see my UserAccount
- Update: when changing my password I want to update my UserAccount
- Delete: when I decide to end my account I want to be able to delete my UserAccount

If you connect the dots, it's clear that the 4 HTTP methods correspond effortlessly with the 4 essential database operations!

Learning Materials

- REST vs. CRUD