# Talent Acceleration Program

## Workbook

tap.kiitos.tech

# Week 0: Preparation

*It's not the will to win that matters – everyone has that. It's the will to prepare to win that matters.*

Welcome to the *Preparation Module* for the **Talent Acceleration Program (TAP)**. In this module you'll learn all about how to set yourself up for success so that you can get the most out of the program!

It is divided into several sections:

1. Learning How To Learn
2. Web Development Fundamentals
3. TAP Quickstart

Good luck!

*The TAP Team*

# PART 2: WEB DEVELOPMENT FUNDAMENTALS

You may already have a degree, training or even some work experience. Then why do you have to review the fundamentals? In order to become good at anything *you must have a strong foundation*. By refreshing the basics you'll refine your mental model of how software development works in order to make sense of new concepts quicker!

## Learning goals:

- ❏ Install Visual Studio Code + GIT
- ❏ Become familiar with HTML5 (incl. ARIA and SEO)
- ❏ Learn about the latest CSS practices
- ❏ Practice with modern JavaScript

## Software development fundamentals

Before we dive into the Frontend Fundamentals, we have to set up a proper development environment.

### CODE EDITOR

Like any job, in order to start you need to set up your workspace first. Having a quality code editor is a great start. As software developers this is our tool of trade. The right editor provides features such as *autocomplete*, *formatting on save* and *debugging tools* in order to boost your productivity.

We recommend Visual Studio Code. It's a popular code editor made by Microsoft, optimized for building and debugging modern web and cloud applications.

Learning Materials

- Extensions
- VS Code Top Ten Pro Tips

**VERSION CONTROL**

Version control is the practice of tracking and managing changes to software code. When building software we always want to be making sure we can revert back to a previous state, in case we made a mistake.

Currently the industry standard for version control is GIT. It allows us to create *local repositories* that store all of our code changes within a given directory.

From there we can create alternate versions of our software project, using *branches*. You'll learn more about why those are important in the following weeks.

Learning Materials

- GIT Explained in 100 Seconds
- Top 10 Git Commands
- Introduction to GIT - Branching & Merging

Exercises:

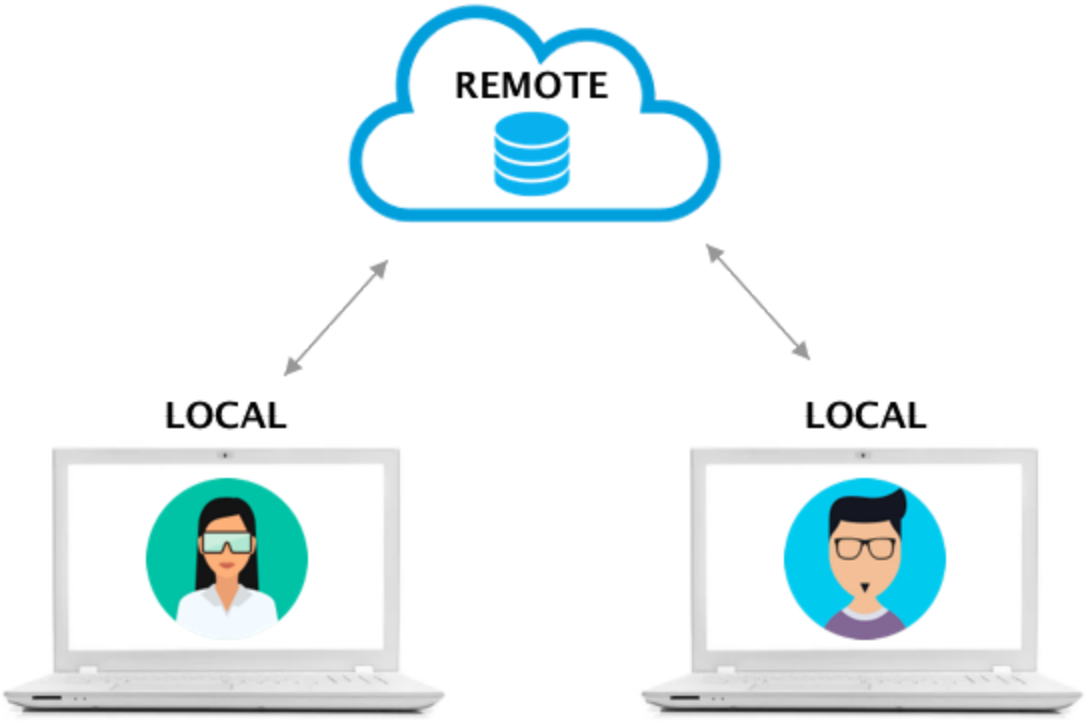
- Play Git game

**REMOTE REPOSITORIES**

In real-world software development you will work with other developers in the same project. In order to share code we use a remote repository, a GIT repository that exists on “in the cloud”.

Various companies exist to provide this service: GitHub, BitBucket or GitLab are currently the most widely used. They each allow you to host your source code projects in a variety of different programming languages and keeps track of the various changes made to every iteration. In addition, they provide other

The main reason why we want to use remote repositories is to *have one central place where we can share access to our project’s source code*. Take a look at the following:




**BUSINESS TERMINOLOGY**

As you make your transition into working life you'll be hearing a lot of unfamiliar terms. They are specific to the business side of software development. Don't worry if you don't understand most (or any) of them just yet, you'll learn more about them in the course of the program.

Agile development

This has little to do with actual code production and more with the way you work within a team in order to deliver new features to the customer at a consistent pace. The leading theory in software development is called Agile and it consists of various practices on how to work in an "iterative" and "self-organized" way.

Learning Materials

- Software Development Methodology: What is Agile?

Business logic

The way you design your data (i.e. entities) and how your application interacts with it is referred to as business logic. It flows from the requirements a certain product has to fulfill in order to reach its goals.

Learning Materials

- What is Business Logic in Software Development

Development vs. Production

As a professional software developer you build digital products. There are always at least 2 states your product is in. In *development* (also known as "dev") we're referring to the unreleased beta version of your application, which usually contains new features that haven't been shown to the customer yet. In *production* (also known as "prod") we have released a version of our application that real users can use right now.

*Note: in practice there might be a couple more stages in between (i.e "testing", "acceptance", "staging"). But the aforementioned are the 2 most important ones.*

Learning Materials

- [Difference Between Development, Stage, And Production]

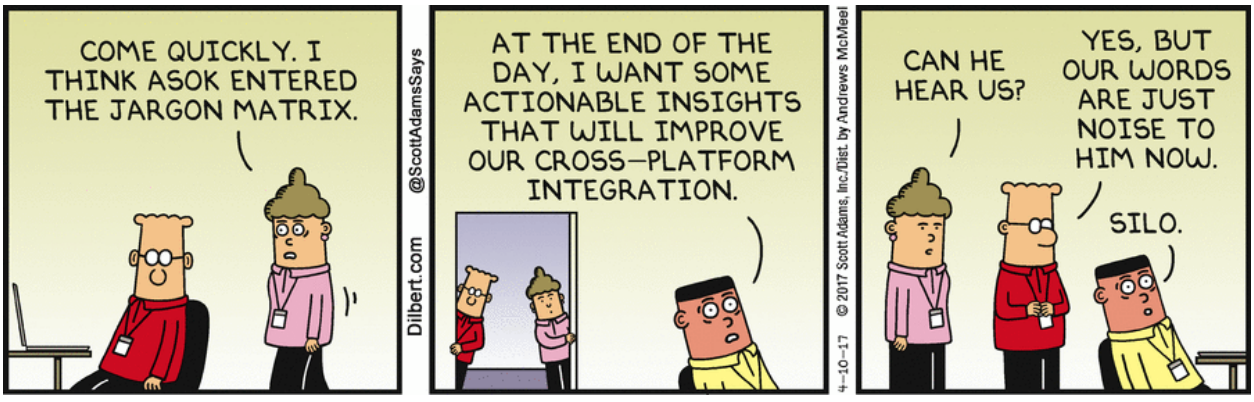
Technical debt

Also known as tech debt or code debt, this is the result of prioritizing speedy (but bloated) over perfectly written code. In business there will be many deadlines and more often than not you'll find yourself writing a first, working version and leaving it there.

Learning Materials

- [Technical Debt]

While this list is not exhaustive, it'll give you a headstart into what you might hear in the workplace. Be sure to always keep your ears open to anything new!



## Frontend fundamentals

There are a large number of JavaScript frameworks and libraries out there that can be used for Frontend Development. Some of these you could have heard of such as React, Angular or Vue.

However, before diving into those it's important to know the foundation of the web: HTML and CSS.

**1. HTML5**


In order to build for the web you need to know the foundational building block: HTML5. It gives us the power to add text, images and videos, thereby providing the *structure* of any webpage in existence.

As professional web developers we should know the internal logic, most commonly used elements and best practices in order to build semantic web pages.

In a business environment, this also includes an awareness of things like accessibility (ARIA) and Search Engine Optimization (SEO) so that any given website becomes more user-friendly and more easily found in search engines, respectively.

Learning Materials

- Learn HTML5 in 1 Hour
- Web Accessibility Guidelines

- What is ARIA & Why it's Important to Use
- SEO for Developers

**2. CSS3**

While HTML by itself could do the job in providing an interface, it doesn't provide the best user experience. Therefore, CSS was invented. With CSS3 (the latest version) the potential for a frontend developer to organize and style user interfaces became even bigger.

Features like media queries, flexbox and keyframes made it possible to make the web even more accessible to many different users on many different devices!

In the evolution of CSS, some developers thought about reducing repeated CSS rules by creating frameworks to be used in the frontend. These consist of many predefined style rules that are assigned to class names you can reuse in your HTML tags.

CSS frameworks (i.e. Bootstrap or TailwindCSS) have gained popularity in business since. Why? (1) It significantly reduces the amount of time it takes to style any webpage as the developer doesn't have to think about which styling rules to apply and (2) it makes the code more readable and maintainable since the framework is continuously updated by an external party!

Learning Materials

- Most Important CSS Properties with Example
- The Box Model
- Complete guide to flexbox
- A Complete guide to CSS Grid
- Responsive Design
- CSS Animation in 100 Seconds
- Vanilla CSS vs. Bootstrap vs. TailwindCSS

Exercises:

- Play Flexbox Froggy
- Play CSS Grid Garden

**3. JAVASCRIPT**


JavaScript began as a browser scripting language. What started as a 10 day experiment has now evolved into one of the most popular languages in the world.

The main use for JavaScript is to make your webpage interactive: for example, if you click a button it will open a popup. Or if you scroll over an image, it changes its color. This is called DOM manipulation.

Learning Materials

- Fundamentals

    - Introduction to JavaScript
    - JavaScript and the DOM
- Tools
    - ESLint
    - Browser DevTools
- Best practices
    - KISS, YAGNI, DRY, SOLID
    - JavaScript Best Practices and Naming Conventions
- Package managers
    - What is NPM?
    - NPM vs Yarn

# ASSIGNMENT: The Pokédex App (FRONTEND)

Description

Build a Pokédex user interface that allows a user to (1) view many different Pokémon, (2) view their details and (3) record whether or not they've captured them. It should be possible to (4) search by name and the (5) data should be persistent after refreshing the page.

Feature list

- Search by name
- List Pokemon by Pokédex #
- Records if the Pokemon has been captured (store data in LocalStorage)
- Popup when clicked on Pokemon to show details
- Persistent data after page refresh

Acceptance criteria

- Uses the following public API: PokéAPI
- Uses vanilla JavaScript and a CSS framework MaterializeCSS
- Handles validation, errors and placeholders
- Includes a header and footer (it should look like a landing page!)
- Always lists 5 (randomized) Pokemon by default at every refresh
- Adds documentation on how to setup the project for viewing locally

How to submit

Submit your application to the TAP-Cohort-4 repository, on BitBucket. Make sure to:

- Create a branch (i.e. "razan-week0-coursework")
- Make a pull request to "main"
- Assign the Program Manager (Razan) to it

**Deadline: Saturday, Feb 19, 2022 23:59**

## Backend Fundamentals

In any web application, the part that a user can interact with is called the frontend (also known as the user interface). These could be elements like buttons, dropdown menus or videos.

Everything else that's needed to make it actually work the way it does, is called the backend. But have you ever wondered why there is this distinction?

**SEPARATION OF CONCERNS**

In software design (that is, how we think we should build any particular software before we actually build it) we always want to do what's most effective: how to write code that's reusable, easy to change and modular.

Why? Because it's the most cost-effective way to create software.

**THE BUSINESS SIDE OF SOFTWARE DEVELOPMENT**

Imagine a customer comes along and offers to pay you for an enhancement (also known as "feature") to their software: a user login system. In order to get paid, you will need to change your program to add the feature. But what constitutes how much money you can ask?

- How much code you have to change
- How easy it is to make the changes
- How likely you are to break existing features that are being used by other customers
- How much you can reuse the existing model/architecture

From a business perspective, all of these points should result in a time estimation: how long will it take to deliver upon the enhancement?

A fundamental principle that allows us to design and build cost-effective software is called separation of concerns.

In simple terms this means: each part of the application should only have 1 job.

- The 1 job of the frontend is to be an interface for the user to interact with. (i.e. easy to understand, good user experience of the company's product)
- The 1 job of the backend is to process data in whatever way is necessary (i.e. store/retrieve data from the database, connect with external APIs)

By having the job (or "concern") clearly defined for each part, it's easier to think about, design and implement the feature.

Let's take a look again at the 4 points that affect the budget of a project, and see how this relates to the concept of separation of concerns:

- **How much code you have to change**
  If all of the code for a particular behaviour of the application is separated out, then you will only have to change code directly associated with your new feature. Which should be less code to change.

- **How easy it is to make the changes**
  If the behaviours you are interested in are neatly separated from the

rest of the application it is more likely you will be able to swap in a new implementation without having to fully understand or manipulate the rest of the program. It should also be easier to find out which code you need to change.

- **How likely you are to break existing features that are being used by other customers**
  Code that you do not have to change is less likely to break than code that you do change. So splitting up the concerns helps you to avoid breakage in unrelated features by preventing you from having to change code that they could call. If your features are mixed up together you might change the behavior of one by accident while trying to change another one.

- **How much you can reuse the existing model/architecture**
  If your architecture doesn't depend on the technical solutions you're using (i.e. libraries or middlewares) or the business logic (operations that handle data according to what services the product offers) then you don't need to modify your architecture when implementing a new feature.

Learning Materials

- What is Separation of Concerns in Software Architecture

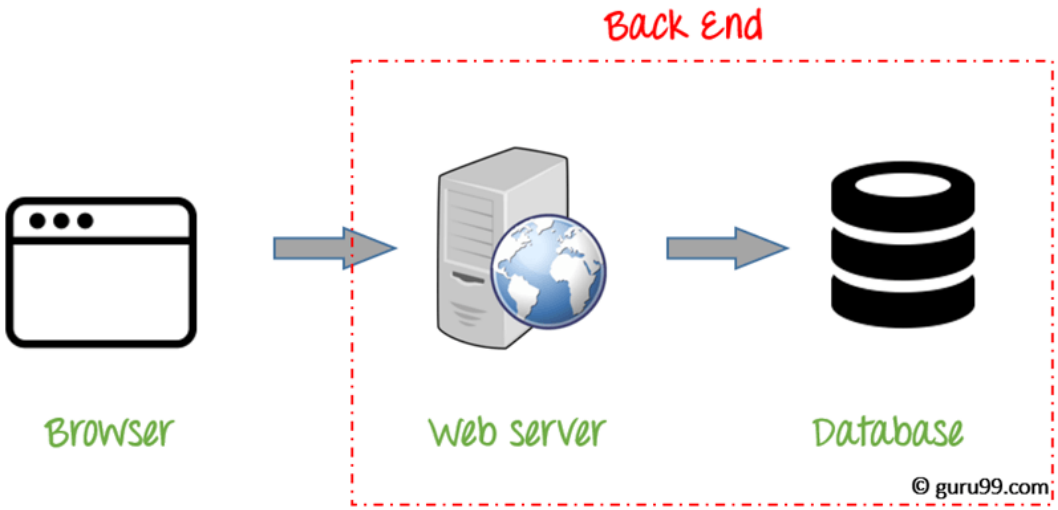
**WHAT IS BACKEND**

Now that we know more about the distinction between frontend and backend, let's dive deeper into the concept of backend itself.

In web development the term backend can be boiled down to 3 components:

- **A server (hardware)**: a computer that is connected to other computers, which runs an application (see below) that allows for sharing and managing services (like a calculator or word processor) and resources (like images or text files).
- **A database (software)**: manages and saves sensitive data in a structured way (i.e. lists of names, email address, etc.).
- **An application (software)**: contains code that allows it to interact with and manipulate the server (i.e. the filesystem, database and other applications).

In web development, the application part of the backend is often referred to as a web server application: it *serves* resources over the Internet to some client, like the browser.




Learning Materials

- A Beginners Guide to Backend Development

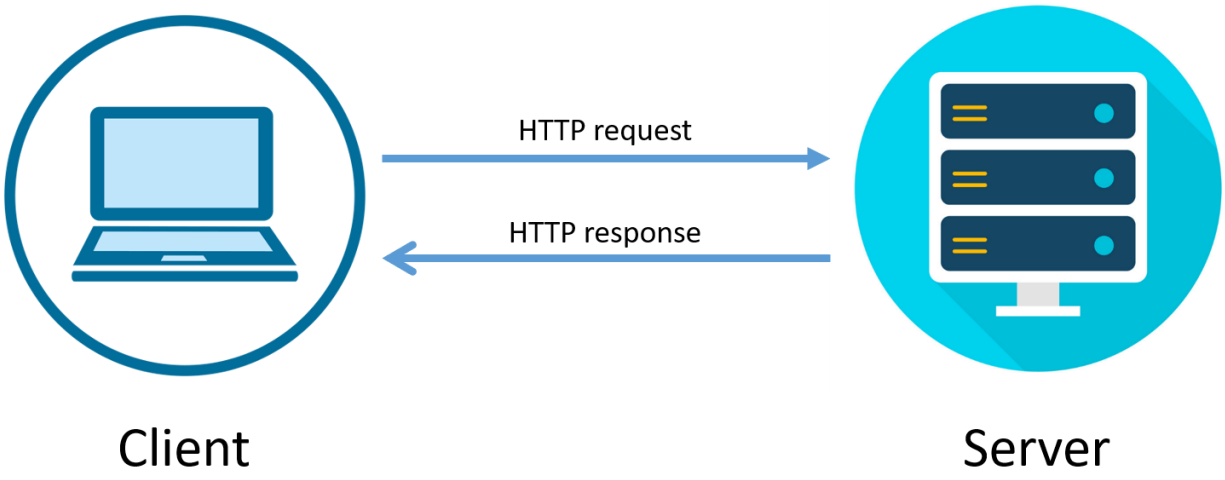
**WEB SERVERS**

A web server application (usually shortened to “web server” or, simply, “server”) is an application that runs on a computer. It contains code that instructs the computer to receive an HTTP request and handle it accordingly. The request comes from a client (usually a browser, but this could also be another application), usually to retrieve, manipulate or store data.

The web server interprets the request and performs what is asked for. Whether or not it has been successful, the web server always sends back a response to the client.

In the case that things didn’t go as expected, the response usually contains an error message (also known as an exception). If things did succeed it sends back as a response what was asked for.

This request-response cycle between a client and server is referred to as the client-server model.



Learning Materials

- What Is a Web Server?
- Client-Server Model
- Web Application Architecture

**USING JAVASCRIPT IN THE BACKEND**

Nowadays there's a way to create backend applications with JavaScript, called Node.js. This software allows us to execute JavaScript files without the need for a browser to interpret it.

Through Node's internal package manager, NPM, we can access many different modules (predefined functionalities other developers have made) that can help us perform essential functions on the operating system of your computer.

Learning Materials

- Software Download
- Learn Node.js in 1 Hour

With Node.js, we can also make web server applications (or web server, for short). There are two ways of doing so: using a combination of native modules (such as *http*) or a web framework that will do most of the essential configuration for us.

Whichever one you choose depends on your business needs: is it an internal project or a client project? How tailor-made should it be? What's the budget and timeline?

**EXPRESS.JS**

In software development we're always striving to make our jobs as simple to code as possible, while still being able to customize when needed. A web framework helps a lot with this.

A popular web framework for Node.js is Express.js. It contains all the essential features needed to rapidly build a web server, but still gives us the freedom to add any other functionality (i.e. by adding native modules included in Node.js, or any module found in the NPM ecosystem) easily.


Learning Materials

- Express.js

**ARCHITECTURAL PATTERNS: MODEL-VIEW-CONTROLLER**

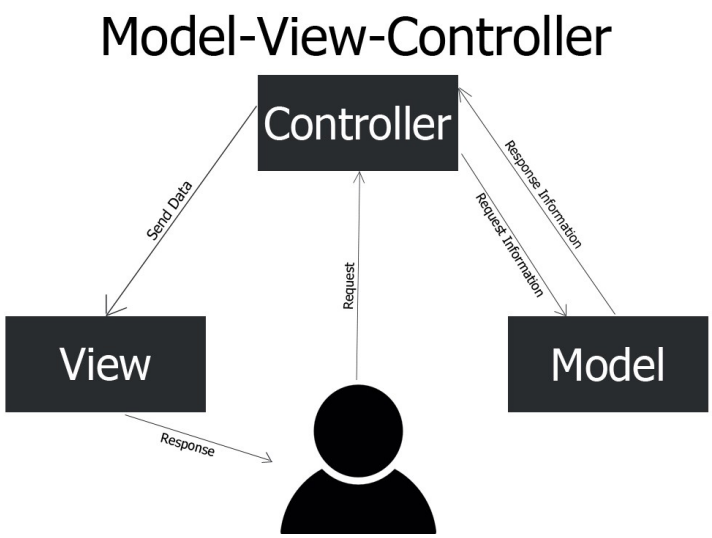
Once you've decided to build a web server, it's important to decide *how* to organize your project folder. Do you put everything into a single file? Or does it make more sense to group features according to functionality?

This is where an architectural pattern comes into play. That's a fancy way of saying: organizing your functions and variables in a scalable and readable way by putting them in files and folders that make sense.

A common architectural pattern is called the model-view-controller (MVC).

The model represents your data definitions or schemas; the blueprint of any particular data entity in your application. A common example of this is a User or Product model.

The view represents the user interface or frontend of your application. It's the place where the data will be shown in a user-friendly way.




The controller (also known as a request handler) represents the part of the web server that moves data from one place to another. It's essentially a function that receives a request and proceeds to "handle" it, depending on what is required (i.e. store/retrieve data from the database, validate inputs from the view, etc.).

Learning Materials

- Separation of Concerns: MVC
- The Big Picture of JavaScript REST APIs

**API TESTING**

A big part of backend development is API testing. You always want to make sure that your backend logic works as intended.

Normally speaking, you'd use the browser or terminal for this. Send a request to an endpoint within your running web server and you should be able to see if it works. However, this is not efficient as not every browser works the same way.


Postman is a graphical user interface that's designed to solve one problem: make API testing as easy as possible. It removes all unnecessary operations the browser or command line might require, plus gives you handy features that make it easier to "test your endpoints".

Learning Materials

- Postman Beginner's Course

# ASSIGNMENT: The Pokédex App (BACKEND)

Description

Using Express.js, create a web server that will (1) serve the frontend from last week and (2) supply the Pokémon data in JSON format (replacing the previously used public API) and (3) update the JSON when a Pokémon is captured.

Feature list

- Serve frontend (from last week)
- Get 5 random Pokémon
- Get Pokémon details
- Update Pokémon based on capture status
- Host on Heroku

Acceptance criteria

- Uses Express.js
- Uses fs for reading and writing JSON
- Includes at least 1 GET endpoint
- Includes at least 1 PUT endpoint
- Incorporates MVC pattern
- Makes use of the following static data set

How to submit

Submit your application to the TAP-Cohort-4 repository, on BitBucket. Make sure to:

- Create a branch (i.e. “razan-week0-coursework”)
- Make a pull request to “main”
- Assign the Program Manager (Razan) to it

**Deadline: Saturday, Feb 26, 2022 23:59**