Talent Acceleration Program

Workbook

tap.kiitos.tech

Week 2

"Time is the scarcest resource; and unless it is managed, nothing else can be managed."

Welcome to **week 2** of the Talent Acceleration Program!

On the *Technical Skills* side,

- The Frontend specialization will go deeper into the core concepts of Single Page Applications (SPA): component-based thinking, the rendering lifecycle and state are found in any of the popular SPA frameworks. As it's the most popular at the moment, we'll use React.js as the one to illustrate these concepts with!
- 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 will get familiar with another fundamental topic: time management. More importantly, how to take ownership of your time so that you can be more productive.

Good luck!

The TAP Team

Part 1: Technical Skills

Jump ahead to Frontend or Backend

Week	Topics
1	Introduction to Specialization
2	Core Concept Specialization
Frontend	State Management
Backend	Core API Features
3	Agile Fundamentals
4	Project Management Fundamentals

Frontend Track

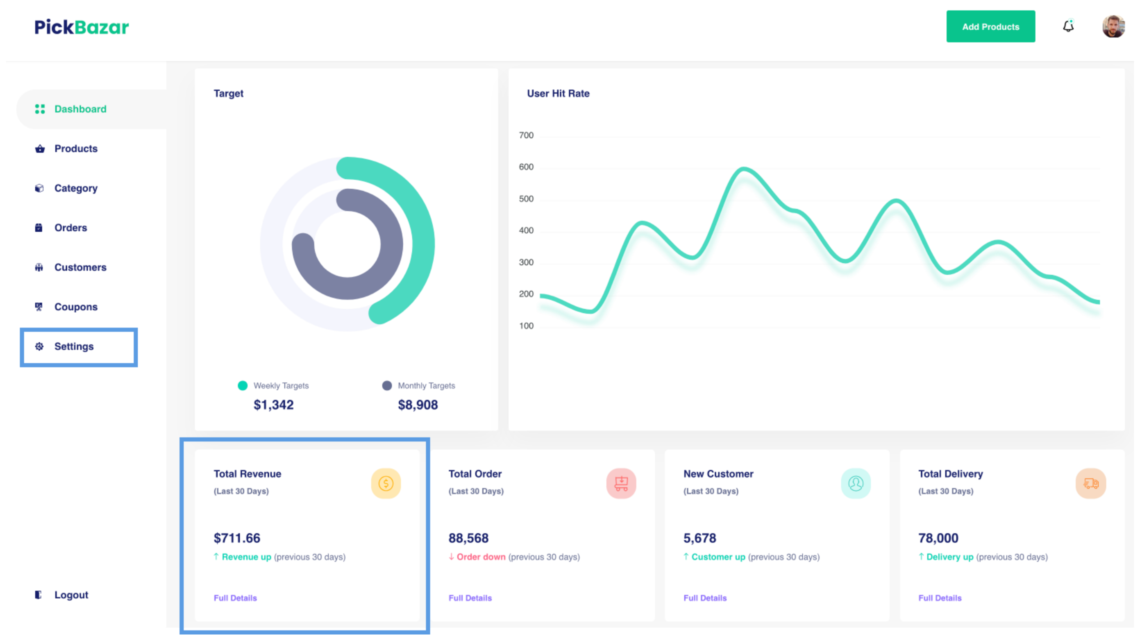
Learning goals:

- ❏ Gain awareness of a core concept of modern frontend development: component-based thinking
- ❏ Understand the lifecycle of page/component rendering
- ❏ Learn about the distinction between local and global state
- ❏ Differentiate between the use cases for various state management tools

CORE CONCEPT: COMPONENT-BASED THINKING

One of the most important concepts in modern frontend development is the idea of components.

Simply put, a component is *a reusable part of a user interface*. In the following example, you'll see 2 components highlighted that are reused in the page: the *menu item* and a *results count container*.



Knowing how to dissect any webpage in this component-based way of thinking is crucial to learning how to work with any type of frontend framework.

Learning Materials

- Component Hierarchy

CORE CONCEPT: COMPONENT LIFECYCLE

If you have no background with any frontend framework and start with React, you might've come across the concept of component lifecycle. The basic idea is that the rendering process of every component goes through roughly 3 stages: mounting, updating and unmounting.

On the surface this seems like a radical idea: rendering a page in various stages. Usually when you deal with regular HTML you just load a page into the browser and it's done, right? Well, actually no. A regular HTML page actually also has a lifecycle!

Learning Materials

- Page: DOMContentLoaded

So the idea of a lifecycle isn't actually new. React just took the concept and reduced its scale from a complete page to a component of a page.

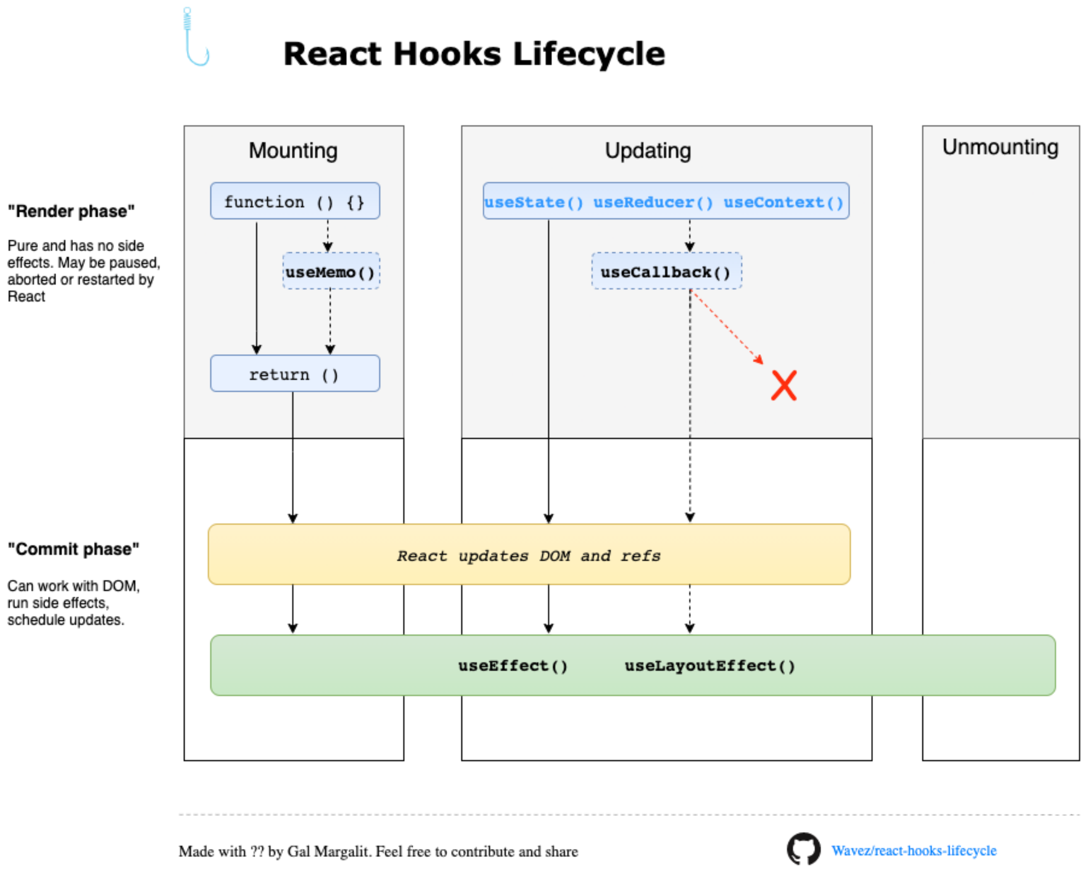
Learning Materials

- The React Lifecycle of a Functional Component

CORE CONCEPT: LIFECYCLE HOOKS

The concept of a React hook is misleading. Isn't it just a function that deals with state management in some way? Yes it is. However, the term "hook" is actually short for lifecycle hook. And what does it "hook into"? *It hooks into one of the stages in the rendering process of a component.*

In other words, it's a special function that should be executed in a certain stage (Mounting, Updating or Unmounting) in order to (1) render the right data at the right time or (2) destroy that data after it has served its purpose so that browser computing power can be freed for rendering something else.



React Hooks Lifecycle
Mounting
Updating
Unmounting
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React
function () {}
useMemo()
return ()
useState() useReducer() useContext()
useCallback()
"Commit phase"
Can work with DOM, run side effects, schedule updates.
React updates DOM and refs
useEffect() useLayoutEffect()
Made with ?? by Gal Margalit. Feel free to contribute and share
Wavez/react-hooks-lifecycle

The diagram above shows which specific hooks are best used in which stage. Effectively, you only need to use useState (to create and manage local state) and useEffect (to deal with side effects) and the others are just there for optimization!

Learning Materials

- 10 React Hooks Explained

- The useEffect Hook and its Lifecycle

CORE CONCEPT: STATE

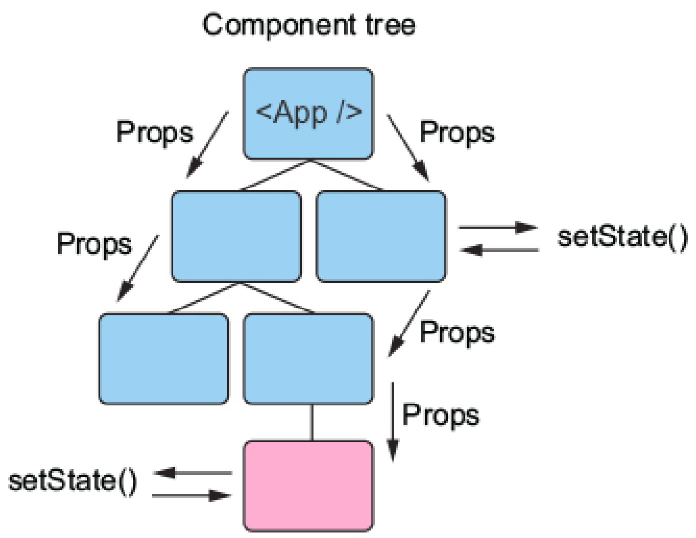
One of the most expensive and complex operations that happens in the frontend is updating the UI state. In this context the term "state" refers to all the (dynamic) data contained with the application.

Learning Materials

- What is "State" in Programming?
- The deepest reason why modern JavaScript frameworks exist

React is one solution that's created in order to simplify this process. All you have to do is:

- Build up the page by creating custom components
- Nest components inside components, to create a tree-like structure
- Pass data in a unidirectional (from top to bottom) way



Learning Materials

- React State vs. Props

- Understanding unidirectional data flow in React

When it was first released, React didn't come with an efficient way to pass data far down the component tree. If you had some kind of shared state you would like to use anywhere else in your application, you had to pass it down from one level to another. In large scale React applications with 100's of components, you can imagine this is very hard to maintain.

This pattern of passing down data many levels downwards is also referred to as prop drilling. This only works in small scale applications with several components. However, as software developers we should also think about building our software in a scalable way: *would this solution still work if our application was 10X its size*?

Learning Materials

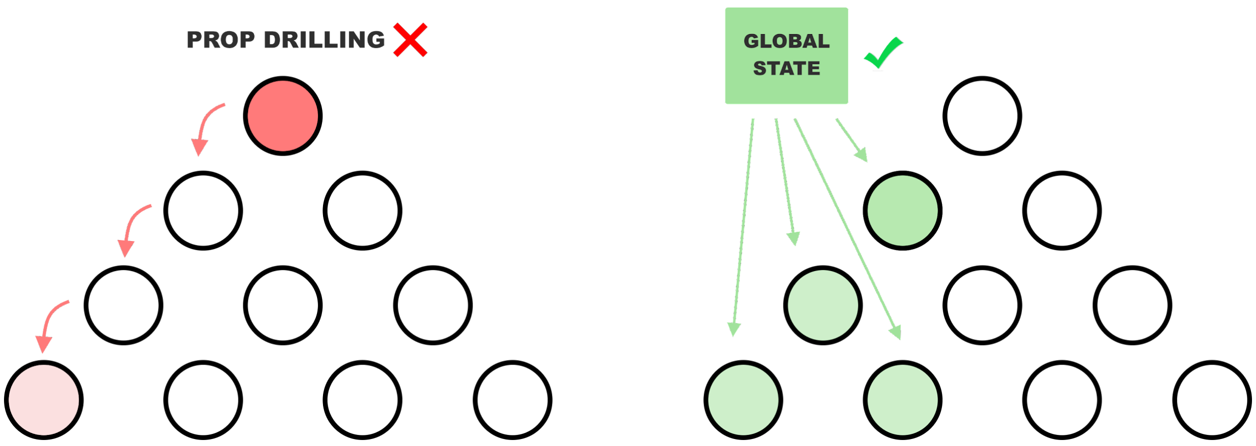
- What is Prop Drilling?

ADVANCED STATE MANAGEMENT

So far we've established that we can manage it *locally* by passing down data as props, custom attributes given to components, in order to access them within its structure.

However, if we have a large-scale application this will not work. There will be too many levels of components to pass through in order to get the state to the right place.

So this is where we have to start thinking about a concept called global state: the idea of moving all of our data into a globally accessible object and then connecting that object directly to the particular component that needs to grab data from it.



This is where state management tools come in: libraries that have been developed to solve the problem of prop drilling. Whichever way you go doesn't matter that much.

The various state management tools that exist ultimately all solve the same basic problem of prop drilling, by allowing us to (1) create a central place that holds all of the state (which we call global state, or store) and (2) provide a way to connect that state directly to the component.

Learning Materials

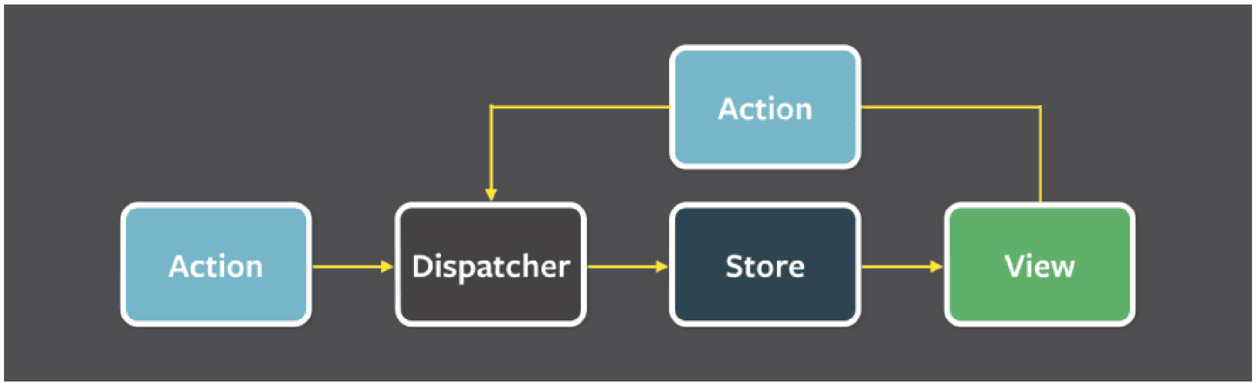
- [How I Manage State in React]

ARCHITECTURAL PATTERN: FLUX

This idea of “global state” came from the need to simplify state changes within a large-scale application. The dominant way to implement this comes from an architectural pattern known as Flux.

Flux consists of 4 parts:

1. **Action**. These are customly defined browser events (i.e. “Add items to shopping cart” or “Log in user”) that contains data which will change the application’s state.
2. **Dispatcher**. This is the function that will take the data from the action and puts it in the store
3. **Store** (also known as “global state”). The central place where we store all of our application’s state).
4. **View**. The user interface, composed of the component hierarchy, will receive state in various places.



The diagram above shows the way data flows in a Flux architecture. As you may notice, it goes in 1 direction only (this is also known as unidirectional data flow). This means that a SPA must follow this very specific procedure, namely creating functions and variables in certain places, in order to update its state. It’s not allowed to update the state anywhere else.

Learning Materials

- Flux Architectural Pattern
- Flux Explained in 3 Minutes

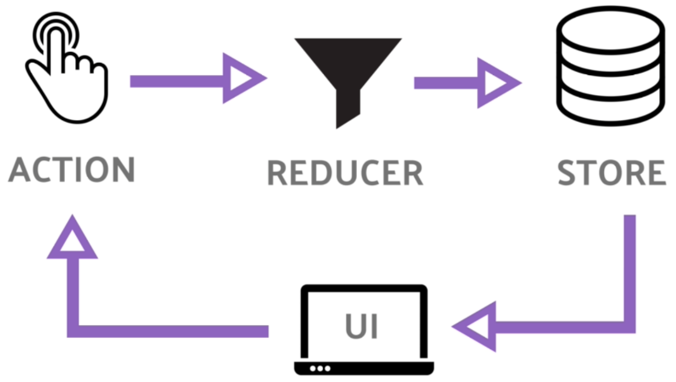
STATE MANAGEMENT TOOLS

When you're building modern frontends, the way you do state management becomes essential. How do you decide on which approach to take? There are 3 basic questions that need to be answered:

1. Size of application
   The bigger the size (i.e. amount of components), the more layers the application state has to pass through.
2. Frontend framework
   Each framework has communities that have built tools in order to make state management easier.
3. Team expertise
   The strategy to use should match with the skill level of the development team that builds the application, as well as those that will maintain it in the future.

This Flux architectural pattern has been the inspiration for many libraries that aim to bring an implementation to our modern frontend application. When it comes to tools the industry standard has been Redux, which is an independent library available that can be used with React, Angular & Vue.

To the right you'll find the way Redux implements the Flux architectural pattern. Notice how it's very much similar. The only exception being the reducer, which is actually just a more descriptive term to describe what the dispatcher would be doing: *modify the state in order to update the store.*



Nowadays the modern frameworks have evolved to now have their own built-in state management solution (being heavily inspired by Redux). React has the [Context API](), Angular has [NgRx]() and Vue has [Vuex](). Different implementations that work more closely with the framework, but that ultimately solves the same problem.

Learning Materials

- Top 5 React State Management Libraries
- Is Redux Dead?

Backend Track

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

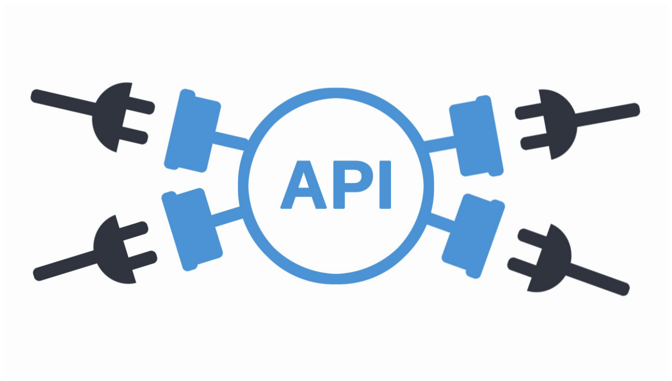
Learning goals:

- ❏ 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 in what ways users can interact with it!

Learning Materials

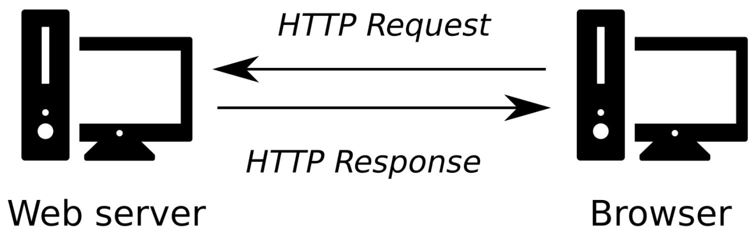
- What is an API?
- RESTful APIs in 100 Seconds
- APIs for Beginners

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*:

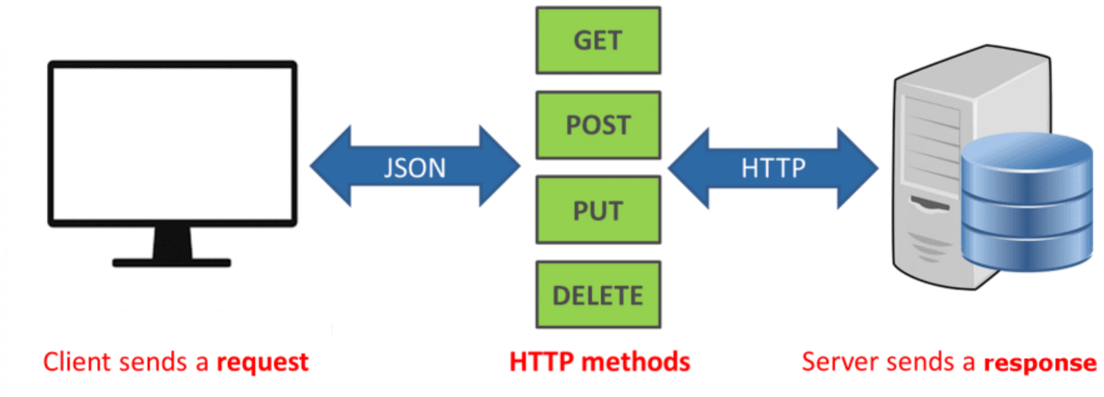
they are 2 separate applications that communicate with each other through an API using requests and responses.



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