**REST Services and KoaJS**

**RESTful services**

* REST-Representational State Transfer Protocol.
* Based in resource and build things or resources
* URI use to identify the resource and multiple URIs may refer the same resoce.
* Use to client and server communication
* JSON and XML use to pass data
* That is Lightweight, scalable and maintainable.

**HTTP Messages**

HTTP verbs

|  |  |  |
| --- | --- | --- |
| Verb | Operation | Safe |
| GET | Fetch a resources | Yes |
| POST | Insert new resource | - |
| PUT | Replacing existing resources | Idempotent |
| PATCH | Update resources | Idempotent |
| DELETE | Remove a resources | No |
| OPTION | Get all allowed option | Yes |
| HEAD | Get only the response header | yes |

**6 Constraints**

* Uniform Interface
  + Base on the HTTP specification. HTTP verbs are GET, PUT, POST, DELETE
  + URIs refer to resources and HTTP verb are action done on resources.
  + HTTP request has header, body, query parameters and URI
  + HTTP response has header and body.
  + Use hypermedia to navigate between resources.
* Stateless
  + Server doesn’t have client state and client have Session state.
  + Self containing messages contains sufficient information for that particular operation.
* Client-Server
  + URI use to connect Client and server.
  + Separate user interface and services.
  + Client does not have any connection to data server
  + Server independent from different user interface.
* Cacheable
  + REST services are cacheable.
  + Server send resources are cashable.
  + Client can cash resources.
  + Server and Client can decide the caching data.
* Layered System
  + Client doesn’t access Underlying layers or complexities of the service.
  + Client only can access abstraction of resources URI and verbs
* Code on demand
  + Code on demand is optional constraint.
  + Server has some part of logic. That can transfer to clints.
  + JS is passed to execute for client.

**Compliance with REST constraint**

* Performance
* Scalability
* Simplicity
* Modifiability
* Visibility
* Portability
* Reliability

How to Operate REST

* Use plural Nouns instead of verbs and don’t use GET method to alter the state
* Use HTTP status code When error occur.
* If we need to other related resources. We can use links.
* HTTPS use to open

**HTTP Codes**

|  |  |  |  |
| --- | --- | --- | --- |
| 200 | OK | 201 | CREATE |
| 202 | Accepted | 204 | No content |
| 304 | Not modified | 400 | Bad request |
| 401 | Unauthorized | 403 | Forbidden |
| 404 | Not found | 409 | Conflict |

**KoaJS**

* NodeJS web application framework.
* Use for develop web application and API
* Minimal features but many feature are available via plugin.
* That is high performance, Test coverage, Robust routing and Reusable

**React JS**

React is a JavaScript library for building web applications' front-ends. We utilize this in our group project to construct user interfaces utilizing its components js files.

Concentrate on speed, simplicity, and dependability.

Large web applications that modify data over time without reloading the entire page are suitable.

Mobile app development with React Native

Developing VR applications with React VR

One way data transmission

* Data is created and stored in a single location that cannot be changed. It is transferred from the parent component to the child component, and then from the child component to the action flow.

Virtual DOM

* Rather than manipulating the DOM, react produces a virtual DOM tree and compares it to the displayed tree, updating only the parts that have changed.

JSX

* The syntax is the same as HTML/XML, and expressions are converted to strings to prevent cross-site scripting attacks.

Props

* Props are used to communicate data and behaviour between container and child components.
* Read only

State

* Modified data is saved in a state.
* In this stage, read and write access is available.
* To alter the status, use the setState() function.
* Because the setState function is asynchronous, the second version of the method is used to use state and props inside the setState method: setState((state,props)=>).

Component life cycle

* Constructor
* constructor in JavaScript. This declares the current state and connects life cycle events to their context. It is not necessary to call SetState. Super(props) can always be used to bind this.
* Render

- The element is rendered.

-This method is invoked when the state or props are changed.

- Returns, among other things, JSX, Portal, Fragments, Strings, and Booleans.

* componentDidMount
* Triggers when the component is added to the DOM.
* Has the ability to communicate with the DOM.
* Ajax requests and state changes
* getDerivedStateFromProps
* This is the way to utilize if you want to edit your state using props.
* This is strictly discouraged by React.
* Using props to conditionally set state is strongly discouraged.
* shouldComponentUpdate
* To assess whether or not the component should be updated.
* Only as a productivity boost if necessary.
* componentDidUpdate
* After the component's update this occur.
* The program will not run if shouldComponentUpdate returns false.
* Use this to interact with the DOM after a Component update.
* componentWillUnmount
* Start by removing the component.
* SetState should be avoided.
* Cleaning jobs
* getDerivedStateFromError/componentDidCatch
* As a result of a previous component error.
* Use getDerivedStateFromError to get error-related State.
* Use componentDidCatch for logging and side effects.

Context API

* + API is a relatively new technology.
  + The main purpose was to keep props from travelling across many tiers.
  + This usage should only be applied to context-level data, according to React. For instance, authenticated users.
  + You should not use this to represent a state.
  + It's difficult to keep the rendering under control.

Hooks

* + This feature is quite new.
  + This is a technique for integrating state and life cycles into functional components, and it's typically used to improve the functionality of such components.
  + Hooks should only be used in React functional components and top-level custom hooks (not in loops, for example).
  + The most popular hooks are useState and useEffect.
  + Additional hooks: useContext and useReducer

Babel

* + The most extensively used transpiler for JavaScript.
  + Convert JavaScript es6 features and beyond to the latest supported version.
  + Babel polyfills are provided to support new global objects. Use the core-js polyfills.
  + Babel's interoperability combined with unique features is one of the best so far.
  + Source maps make debugging easier.
  + The optimization is built into the code.
  + Support for FLOW and JSX (XML-like syntax extension) (Static type checker).

Webpack

* The most often used module bundler.
* Module bundling and the build system are both supported.
* Split the code into many files.
* Both AMD and CommonJS are available.
* Extend using loaders (transpiling, css transformation, image optimization).
* Encourages productivity enhancements (minification, uglification).
* Input, loaders, plugins, and output may all be configured in four simple steps.

Parcel

* The most efficient module bundler.
* new to the neighborhood
* Configurations take precedence over conventions.
* JSX, SCSS, LESS, SVG, TypeScript, and other languages are supported.
* Extension plugins and automatically recognized plugins