======================**Spring**=======================

Spring Context - DI container that instantiates, configures, and manages a set of instances (beans) that are defined with a configuration metadata. Allows us to manage the composition and life cycle of objects and their dependencies, dynamically. **DI/IoC** - The dependent object receives and uses the dependencies, but it does not create them. **Wiring** – establish the dependencies. Constructor Injection and Field/Prop Injection (after obj creation, requires mutable fields, obj are not ready to operate after construction; `@AutoWired lateinit var`). **Context Configuration**: XML config files; Java annotations or code (`@Component`, `@Bean` inside `@Configuration` class, `@Service`, `@Repository`, `@Controller`, …).

Spring MVC - library/framework for handling HTTP reqs, providing higher level features on top of the Java Servlet API, based on the MVC pattern. Follows the Singleton pattern, that ensures only one instance of an object exists per application (changeable with `@Scope`). Usage of an underlying servlet server, such as Jetty, Tomcat. A dispatch servlet that is registered on top of that servlet server.

Servlet - Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a req-res programming model. Allow req handlers (servlets) and intermediaries (filters) to run on multiple web servers, the so-called servlet containers.

Filter - intermediary that can perform: 1.pre-processing (before the req-res pair is passed in to the next intermediary or final handler) 2.post-processing and 3.short-circuit the req processing, by not calling the next intermediary/final handler. `HttpFilter(); doFilter; try finally; chain:FilterChain; @Order(Ordered.HIGHEST_PRECEDENCE)`

Interceptor - Spring-specific mechanism to intercept the call to the req handler, i.e. to have code executed before and after the handler is executed; can access spring-specific info, but run later in the pipeline. `HandlerInterceptor; preHandle(request:HttpServletRequest, response: HttpServletResponse,handler:Any):Boolean;afterCompletion(request,response,handler,ex:Exception?);postHandle(request,response,handler,modelAndView:ModelAndView)`

Argument binding - mechanism by which Spring MVC computes the arguments to pass to the handlers, typically based on info available in the req message. `HandlerMethodArgumentResolver;supportsParameter(parameter:MethodParameter):Boolean;parameter.parameterType; override fun resolveArgument(parameter:MethodParameter, mavContainer:ModelAndViewContainer?,webRequest:NativeWebRequest,binderFactory:WebDataBinderFactory?):Any`

Controller - class annotated with `@Controller` or `@RestController` that contains req handling methods (`@RestController` annotation is a specialization of the `@Controller` annotation, with the added behavior that the return value of each method is automatically serialized into JSON and passed back into the `HttpServletResponse` object).

Handler - instance method inside a controller class, annotated with `@<Request>Mapping`.

Message conversion - mechanism responsible for translating HTTP messages payloads into objects and vice-versa, using Jackson lib. Support for other formats using `HttpMessageConverter` interface.

`@SpringBootApplication … WebMvcConfigurer; override fun addInterceptors(registry:InterceptorRegistry){registry.addInterceptor(exampleHandlerInterceptor)}:fun main(args:Array<String>){runApplication<Ap>(*args)}`

`@ControllerAdvice` annotation allows to define a class that is going to be used to handle exceptions thrown by handlers.

==================**Web Architecture**==================

The World Wide Web (WWW) is an info space in which the items of interest (resources), are identified by global identifiers called Uniform Resource Identifiers (URI), that are universal (can be used to identify resources in any info space). **3 architectural bases**: Identification (URIs identify resources), Interaction (Web agents communicate using standardized protocols that enable interaction through the exchange of messages which adhere to a defined syntax and semantics), Formats (Most protocols used for representation retrieval and/or submission make use of a sequence of one or more messages, which taken together contain a payload of representation data and metadata, to transfer the representation between agents). `google.co.uk` (uk-TLD,`co.uk`-eTLD,`google.co.uk`-eTLD+1)

====================**HTTP Protocol**===================

HTTP is a generic interface protocol for info systems. It is designed to hide the details of how a service is implemented by presenting a uniform interface to clients that is independent of the types of resources provided. HTTP enables the use of **intermediaries** to satisfy reqs through a chain of connections. There are three common forms of HTTP intermediary: **proxy** (message-forwarding agent that is selected by the client, usually via local configuration rules, to receive reqs for some type(s) of absolute URI and attempt to satisfy those reqs via translation through the HTTP interface), **gateway** (a.k.a. "reverse proxy", is an intermediary that acts as an origin server for the outbound connection but translates received reqs and forwards them inbound to another server or servers, and **tunnel** (acts as a blind relay between two connections without changing the messages). HTTP provides a **uniform interface** for interacting with a resource, regardless of its type, nature, or implementation, via the manipulation and transfer of representations. What makes HTTP significantly different from RPC is that the reqs are directed to resources using a generic interface with standard semantics that can be interpreted by intermediaries almost as well as by the machines that originate services. Unlike distributed objects, the standardized req methods in HTTP are not resource-specific, since uniform interfaces provide for better visibility and reuse in network-based systems [REST]. Semantics of methods, status codes, headers…, does not depend on the target resource.

Identification, interaction, and representation are orthogonal concepts, meaning that technologies used for that may evolve independently – IANA manages a set of registries with Web specifications. No payload: `204 No Content, 304 Not Modified, 205 Reset Content, 416 Range Not Satisfiable`

Content negotiation is the mechanism that is used for serving different representations of a resource to the same URI to help the user agent specify which representation is best suited for the user (for example, which document language, which image format, or which content encoding).

Message payload = payload headers + payload body

NGINX is a web server that can also be used as a reverse proxy, load balancer, mail proxy, and HTTP cache.

=====================**Hypermedia**===================

Hypermedia: way of linking related resources in a web application by using URLs or URIs, allowing the client to discover new resources and actions by following links (journey; autonomy). It's often used in RESTful web services and is one of the key principles of the REST architectural pattern.

Web Linking: specification defines a model for the relationships between resources on the Web ("links") and the type of those relationships ("link relation types") - *link context has a link relation type resource at link target, which has target attributes*. HAL (tem templates, Siren não), Siren, Collection+JSON. **Siren**: hypermedia specification for representing entities. Link relations define a relationship between two resources. Classes define a classification of the nature of the element, be it an entity or an action, in its current representation. Sub-entities exist to communicate a relationship between entities, in context. Links are primarily navigational and communication ways clients can navigate outside the entity graph. Actions represent the operations that can be performed on an entity (*show available behaviors an entity exposes*).

============**The Browser Application Platform**===========

The browser is a platform to execute applications - host environment. A **module** is a mechanism to organize and isolate code (by default, on the browser all the code runs in the top-level scope, which is the global scope - multiple script elements exist in the same scope). A module system provides isolation, reusability, code execution on initialization, export behavior and data other modules. Node.js uses CommonJS and Browser uses ES6. However, if the application has a lot of modules, the browser will need to load a lot of files, which will impact the performance of the app(gera bottleneck na rede). We should not need to reduce the module count for efficiency purposes. The solution to this is **bundling** multiple modules into a single load level module; this means that a build process is required to bundle the modules into a single file. **Webpack**: module bundler, creates a dependencies graph based no q é exported. Tbm compila TS e JSX em JS; é necessário loaders;

TypeScript is a typed superset of JS that compiles to plain JS – structural type system. The type info in the source files is checked and erased on the compiled files - type system is not reified.

`require()` funciona com Webpack e vai buscar ao `node_modules`

=======================**React**=======================

React is a JS library to manipulate trees that represent user interfaces, including the browser's **DOM tree**. The UI for a browser window is defined by the contents of the DOM (Document Object Model) tree, composed by DOM nodes (element nodes or text nodes). The way React is used to control the UI is based on the concept of virtual DOM, that reflects the end goal for the UI state (this is done independently of the current state of the DOM tree). React then reconciliates (creating, removing, or changing existing DOM trees) the state of the virtual DOM tree with the state of the real DOM tree. The `key` property, which needs to be added when creating sequences of elements of the same type (when a new sequence is represented in the virtual tree, React will use the `key`'s value to determine which elements represent are new UI things and which represent things that were already there). Virtual DOM tree elements are never reused; a new virtual tree is always composed by newly created elements. However, these newly created elements can represent already existing UI things.

Virtual tree nodes are created by using the `React.createElement(type, props, [...children])` function or by using the JSX syntax. React elements (i.e. the virtual tree nodes) are simple JS objects and are much less costly to create than real DOM nodes. `type` is a string or a React component; `props` contain the props of the element, null is like an empty object (`ref` and `key` are special- instead of `element.props.key`, `element.key`). `document.createElement` returns a DOM element. A component is made of elements and other components. An element is a plain object describing a component instance or a DOM node.

React is divided into a host-independent part (used to create tree representations, before they are applied to a host specific tree – reconciler – react) and a host-dependent part (knows how to manipulate a specific tree – renderer – react-dom).

Hooks are functions that let you hook into React state and lifecycle features from function components. *Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns. By following this rule, you ensure that Hooks are called in the same order each time a component renders.*

`useEffect(setup, dependencies?);` `[]` uma vez, `undefined` sempre

Context provides a way to pass data through the component tree without having to pass props down manually at every level. `const Context = React.createContext('abc'); <Context.Provider value='cba'>…; const a = React.useContext(Context).`

React Router is a collection of navigational components that compose declaratively with your application, and we use it to create single-page applications with React, using Client-Side Routing (a plain HTML doc can link to other doc and the browser handles the history stack itself; enables devs to manipulate the browser history stack without making a req to the server). `Link` usa CSR, muda o URL, render, mas ñ refresh

The **History** interface allows manipulation of the browser session history, that is the pages visited in the tab or frame that the current page is loaded in. Mudar `location`, tirando `hash`, faz refresh, History não faz. **Deep-linking** é um conceito que permite que uma SPA seja acessível através de URLs, ou seja, que seja possível navegar entre diferentes páginas da SPA através de URLs. Para que uma SPA suporte deep-linking, é necessário que o cliente utilize a Location e a History API (que permite que o browser navegue entre diferentes URLs sem recarregar a página). React Router suporta deep-linking.

======================**Other**======================

`ThreadLocal<T>(init);get();set();remove();initialValue()`

`preventDefault()` previne o comportamento padrão de um evento. **Server-Sent Events (SSE)** is a technology for sending real-time updates from a server to a web client. It is a simple, lightweight protocol that allows a web page to receive updates from a server in real-time, without the need for long polling or continuously opening and closing connections. SSE is built on top of HTTP, and uses a single, long-lived connection to send updates to the client. This allows for low-latency communication and reduces the load on both the client and the server. SSE can be used for a variety of applications, such as live updates, notifications, and real-time data streaming.