Full Stack Study Guide

I. Back End Development
  A. Java
  B. Spring
    i. Boot
      a. Request Lifecycle
        1. Dispatcher Servlet intercepts the request
        2. Dispatcher Servlet determines the correct handler for the request
        3. Dispatcher Servlet maps the incoming request to a method in that controller
        4. The controller method is executed
          A. Controller should call a method from the service
          B. If providing access to an entity, service should access the repository
          Γ. Repository should map the data in the DB to an entity
        5. Controller returns the requested resource
        6. Response is converted to requested format, if available
        7. Response is returned to client
  C. Hibernate
    i. Definition
      a. An open source, lightweight Object Relational Mapping (ORM) framework that implements the specifications of the Java Persistence API (JPA).
    ii. Is there a time when one would not want to nest an object?
      a. If sensitive data is contained in the object to be nested, one may choose not to nest the object, but it is almost always desirable to have objects with defined relationships mapped to one another, which is why the @JsonBackReference and @JsonIgnore annotations exist.
  D. Annotations
    i. Commonly Used Annotations
      a. Bean
        1. Package
          A. org.springframework.context.annotation.Bean
      b. ComponentScan
        1. Package
          A. org.springframework.context.annotation.ComponentScan
      c. Configuration
        1. Package
          A. org.springframework.context.annotation.Configuration
      d. EnableAutoConfiguration
        1. Package
          A. org.springframework.boot.autoconfigure.EnableAutoConfiguration
      e. SpringBootApplication
        1. Package
          A. org.springframework.boot.autoconfigure.SpringBootApplication
        2. Notes
          A. Comprised of three other annotations
            α. Configuration
            β. EnableAutoConfiguration
            γ. ComponentScan
      f. JsonBackReference

1. Package
    A. com.fasterxml.jackson.annotation.JsonBackReference

g. JsonIgnore
    1. Package
        A. com.fasterxml.jackson.annotation.JsonIgnore

h. Id
    1. Package
        A. javax.persistence.Id

i. Entity
    1. Package
        A. javax.persistence.Entity

j. Table
    1. Package
        A. javax.persistence.Table

k. Column
    1. Package
        A. javax.persistence.Column

l. Index
    1. Package
        A. javax.persistence.Index

m. Min
    1. Package
        A. javax.validation.constraints.Min

n. Max
    1. Package
        A. javax.validation.constraints.Max

o. NotNull
    1. Package
        A. javax.validation.constraints.NotNull

p. JoinColumn
    1. Package
        A. javax.persistence.JoinColumn

q. SuppressWarnings
    1. Package
        A. java.lang.SuppressWarnings

r. Embedded
    1. Package
        A. javax.persistence.Embedded

s. Embeddable
    1. Package
        A. javax.persistence.Embeddable

t. AttributeOverrides
    1. Package
        A. javax.persistence.AttributeOverrides

u. OneToOne
    1. Package
        A. javax.persistence.OneToOne

v. OneToMany
    1. Package

        A. javax.persistence.OneToMany
w. ManyToOne
    1. Package
        A. javax.persistence.ManyToOne
x. ManyToMany
    1. Package
        A. javax.persistence.ManyToMany
y. Override
    1. Package
        A. java.lang.Override
z. Service
    1. Package
        A. org.springframework.stereotype.Service
aa. Controller
    1. Package
        A. org.springframework.web.bind.annotation.Controller
ab. RestController
    1. Package
        A. org.springframework.web.bind.annotation.RestController
ac. Inheritance
    1. Package
        A. javax.persistence.Inheritance
ad. Slf4j
    1. Package
        A. lombok.extern.slf4j.Slf4j
ae. Data
    1. Package
        A. lombok.Data
af. RequiredArgsConstructor
    1. Package
        A. lombok.RequiredArgsConstructor
ag. AllArgsConstructor
    1. Package
        A. lombok.AllArgsConstructor
ah. NoArgsConstructor
    1. Package
        A. lombok.NoArgsConstructor
ai. Builder
    1. Package

        A. lombok.Builder
aj. ControllerAdvice
    1. Package
        A. org.springframework.web.bind.annotation.ControllerAdvice
ak. ExceptionHandler
    1. Package
        A. org.springframework.web.bind.annotation.ExceptionHandler
al. Aspect
    1. Package
        A. org.aspectj.lang.annotation.Aspect
am.Component
    1. Package
        A. org.springframework.stereotype.Component
an. Repository
    1. Package
        A. org.springframework.stereotype.Repository
ao. Before
    1. Package
        A. org.aspectj.lang.annotation.Before
ap. After
    1. Package
        A. org.aspectj.lang.annotation.After
aq. Request Mapping
    1. Package
        A. org.springframework.web.bind.annotation.RequestMapping
ar. GetMapping
    1. Package
        A. org.springframework.web.bind.annotation.GetMapping
as. PostMapping
    1. Package
        A. org.springframework.web.bind.annotation.PostMapping
at. PutMapping
    1. Package
        A. org.springframework.web.bind.annotation.PutMapping
au. DeleteMapping
    1. Package
        A. org.springframework.web.bind.annotation.DeleteMapping
av. PathVariable
    1. Package

A. org.springframework.web.bind.annotation.PathVariable

aw. RequestBody
1. Package
A. org.springframework.web.bind.annotation.Requestbody

ax. RequestParam
1. Package
A. org.springframework.web.bind.annotation.RequestParam

ay. Produces
1. Package
A. javax.ws.rs.Produces

az. Consumes
1. Package
A. javax.ws.rs.Consumes

ba. PreAuthorize
1. Package
A. org.springframework.security.access.prepost.PreAuthorize

bb. Autowired
1. Package
A. org.springframework.beans.factory.annotation.Autowired

bc. EnableGlobalMethodSecurity
1. Package
A. org.springframework.security.config.annotation.method.configuration.Enable GlobalMethodSecurity

bd. EnableWebSecurity
1. Package

A. org.springframework.security.config.annotation.web.configuration.EnableWebSecurity

be. Description
1. Package
A. org.springframework.context.annotation.Description

bf. Transactional
1. Package
A. org.springframework.transaction.annotation.Transactional

II. Front End Development

A. HTML

i. Semantic Markup

a. Definition
1. Markup elements with names that convey the content's structure or meaning, as opposed to its presentation
2. e.g. <article>, <section>, <form>, <table>

b. Uses
1. Accessibility
A. Allows a screen reader to identify content and deliver a more coherent and comprehensible experience
2. Search Engine Optimization (SEO)
A. e.g. <h1> content is given more importance than <p> content
3. Maintainability
A. It is easier for a developer to recognize the structure and content of the page and update it accordingly

ii. Accessible Rich Internet Applications (ARIA)

a. A powerful technical specification for adding accessibility information to elements that are not natively accessible
1. If a semantic HTML tag is available, use that, not ARIA
2. For dynamic content, use ARIA roles, alerts, and frameworks that specifically support accessibility

B. CSS

i. Box Model

a. A model that conceives of each HTML element as a box comprised of four layers: the content in the middle of the box, surrounded by the padding, contained by the border, which is cushioned by the margins.

ii. Organization

a. Principles
1. Follow the project's style guide
2. Keep format and structure consistent
3. Format CSS to be readable
4. Create logical sections of the stylesheet
5. Place block comments between logical sections
6. Avoid overly-specific selectors
7. Break large stylesheets into multiple smaller ones

b. BEM

C. JavaScript

D. React

i. Definition

a. A JavaScript library that allows the developer to build dynamic, single page applications (SPAs) using JavaScript XML (JSX) to simplify creating HTML elements.

ii. Conditional Rendering
    a. condition && component/element
iii. List rendering
    a. {list.map((item) => <List key={“item-” + item} value={item} />)}
    b. Key
        1. Should not be index, because the key is used to check if the elements have changed, so if a modification changes the keys to all elements, all elements will need to be re-rendered.
iv. Components
    a. Class Component Lifecycle Methods
        1. Mount
            A. Definition
                α. Placing an element in the DOM
            B. Methods
                α. constructor()
                    A. Called first and is the natural place to setup the initial state of the component
                β. getDerivedStateFromProps()
                    A. Called right before rendering the component and is a good place to set state based on the initial props
                γ. render()
                    A. Always called; this method outputs the HTML to the DOM
                δ. componentDidMount()
                    A. Called after render() and is the place to put statements that require the component to already be placed in the DOM.
        2. Update
            A. Definition
                α. Updating a component when there is a change in the component’s state or props
            B. Methods
                α. getDerivedStateFromProps()
                    A. first method called when a component gets updated
                β. shouldComponentUpdate()
                    A. Returns a boolean describing whether React should continue with the rendering or not; default is true
                γ. render()
                    A. Outputs the HTML to the DOM
                δ. getSnapshotBeforeUpdate()
                    A. Allows you to access the props and state before the update, meaning that even after the update, you can still check what those values were before the update
                ε. componentDidUpdate()
                    A. Called after the component is updated
        3. Unmount
            A. componentWillUnmount()
                α. Called when the component is about to be removed from the DOM.

b. Pure Components
    1. Class components that extend React.PureComponent
    2. Includes a default implementation of shouldComponentUpdate() that checks the new and previous state and props to determine if component should be re-rendered

v. React Hooks
    a. State Hook
        1. Allows you to set and access state in a function component
        2. React will preserve this state between re-renders.
    b. Effect Hook
        1. Adds the ability to perform side effects from a function component
        2. Serves the same purpose as
            A. componentDidMount()
            B. componentDidUpdate()
            Γ. componentWillUnmount()
        3. May return a function to specify how to clean up after the effect
    c. Context Hook
        1. Exposes access to the context from a function component
    d. Memo Hook
        1. Cache the return value and return that value if the same arguments are passed in.
    e. Use Callback Hook
        1. Takes an inline callback and an array of dependencies and returns a memoized function
    f. UseRef Hook
        1. Persists a value across renders; does not trigger a rerender.
    g. UseReducer
        1. Provides a more structured approach to updating state than useState does.

vi. Context API
    a. Create context with createContext()
    b. Wrap context provider around component tree
    c. Put desired value on context provider with value property
    d. Read that value within any component using the context consumer

vii. Share logic across components
    a. Higher order components
    b. Render props pattern
    c. Custom hooks

E. Redux
    i. Definition
        a. A state management library often used with React
    ii. Redux Store
    iii. Actions
    iv. Action Creators
    v. Reducers
    vi. How does control flow between the store, actions, action creators, and reducers?
    vii. What does the connect function do?
    viii. What do mapStatetoProps and mapDispatchToProps do?
    ix. Why should you dispatch an action to update the state and note update the store directly?

x. In a reducer, why should you return a new object as state and not modify the existing state?

III. DevOps
  A. Containers
    i. Docker
  B. Cloud
    i. Amazon Web Services

IV. Version Control Systems
  A. Git

V. Persistence
  A. SQL
    i. Each record has a fixed schema
    ii. Columns must be decided before data entry
    iii. Vertically scalable
    iv. ACID compliant
  B. NoSQL
    i. Dynamic schemas
      a. Columns can be added on the fly
      b. Rows don't have to contain each column
    ii. Horizontally scalable
    iii. Sacrifice ACID compliance for performance and scalability
    iv. Cloud-based computing & storage: designed to be scaled across multiple data servers
    v. Types of NoSQL databases
      a. Key-Value Stores
        1. Dynamo
        2. Redis
        3. Voldemart
      b. Document Databases
        1. MongoDB
        2. CouchDB
      c. Wide-Column Databases
        1. Cassendra
        2. HBase
      d. Graph Databases
        1. Neo4J
        2. InfiniteGraph
  C. Miscellaneous
    i. ACID
      a. Atomic
        1. Either the entire transaction succeeds or none of it does
      b. Consistent
        1. Saved data cannot violate the database's integrity; interrupted changes are rolled back to the state they were in prior to the transaction
      c. Isolation
        1. A transaction is not affected by any other transactions taking place
      d. Durable
        1. Once a transaction is committed, it remains committed regardless of subsequent system failures
    ii. Choosing a SQL or a NoSQL database

a. SQL
    1. Schemas are not likely to change
    2. Data is structured
    3. ACID compliance is a priority
b. NoSQL
    1. Schemas are dynamic
    2. Storing large volumes of data with no fixed structure
    3. Data needs to be stored across servers in different regions or across multiple servers

VI. Programming Paradigms

A. Object Oriented Programming

i. Four Pillars of OOP
    a. Abstraction
    b. Encapsulation
    c. Inheritance
    d. Polymorphism

ii. What is a class?
    a. A blueprint for an object being represented within the system

iii. What is an interface?
    a. An interface is a contract establishing a set of methods that must be provided by any class implementing that interface.

iv. Describe the difference between inheritance and composition.
    a. Inheritance establishes a relationship between a generalization and a specialization, which can be simplified with the phrase, “is-a relationship,” while composition establishes an associative relationship of distinct objects, which can be simplified with the phrase, “has-a relationship.”

B. Aspect Oriented Programming (AOP)

i. Definition
    a. A programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.

ii. Explanation
    a. Any consideration (usually, if not always, nonfunctional requirements) which need to be implemented in many places independently, can be addressed via AOP by creating the aspect, implementing the desired logic, and then specifying the join point.

iii. Example
    a. Logging
        1. Create the logging aspect
        2. Define the function to log that a method has been called
        3. Specify the join point at which to call that function
            A. e.g. right before the function is called

VII. Application Programming Interface (API)

A. REST

i. Definition
    a. Representation State Transfer

ii. Naming RESTful URL Endpoints
    a. All elements of URL should be nouns describing resources
    b. Http methods provide the verbs describing the action to take place
        1. GET

A. Retrieves a resource for viewing
2. POST
A. Creates a resource
3. PUT
A. Updates a resource in its entirety
4. PATCH
A. Updates part of a resource
5. DELETE
A. Deletes a resource

B. SOAP
i. Definition
a. Simple Object Access Protocol

C. Comparison
i. REST
a. REST allows greater variety of data formats, whereas SOAP only allows XML
b. Coupled with JSON, which offers faster parsing, REST is often easier to work with
c. Because of JSON, REST offers better support for browser clients
d. REST offers better performance through caching for information that isn't altered and isn't dynamic
e. REST is generally faster and uses less bandwidth
f. REST is easier to integrate with existing websites

ii. SOAP
a. More robust security because it supports WS-Security and identity verification through intermediaries as well as point-to-point, which is supported by both REST and SOAP
b. SOAP offers built-in retry logic for failed communications
c. SOAP's standard HTTP protocol makes it easier to operate across firewalls and proxies without modification to the protocol, itself.
d. SOAP can be ACID compliant
e. SOAP can be less complex than REST when a web service must maintain content and context
f. SOAP is highly extensible through other protocols and technologies, such as WS-Security, WS-Addressing, WS-Coordination, WS-ReliableMessaging, and so on

VIII. Design Patterns

A. Singleton Pattern

B. Builder Pattern
i. Returns an object by providing piecemeal specifications using a fluent API (object.builder().property1(arg1).property2(arg2).build()).

C. Factory Pattern
i. Returns a desired object by providing instructions as arguments to a factory class (shapeFactory.create(Shapes.CIRCLE, 3))

D. Model-View-Controller (MVC)
i. Definition
a. A pattern where the application is separated into three distinct spheres of responsibility

ii. Breakdown
a. Model
1. The representation and management of the data represented within the system.
b. View

    1. The user interface or presentation layer of the application which is responsible for displaying the application's output
c. Controller
    1. Responds to user input and sends that input to the model. May apply validations to that input.