Spring Review Concepts

Q1: Annotations to tell Spring Data JPA to keep track of table mapping

@Entity

@Table just give the name of the table

@Repository creates a class with CRUD functionality (interacting with not setting up)

Q2: HTTP methods

1. safe methods (don’t affect the state of the database)
   1. GET
2. indempotent (don’t affect the state of the database on the 2nd 3rd 10th etc time it runs)
   1. POST – creates a new row each time so is NOT indempotent (the state has changed)
   2. PUT – updates a row to a given state IS indempotent (the next time we run it the state is told to “change” but since it’s not a new change it’s indempotent)
   3. GET – IS indempotent bc it never makes changes to the database
   4. DELETE – IS indempotent (if I delete ericas-account twice it is already deleted so does not have any effect)

Q3:

@RestController – what we use on the class level to not have to put @ResponseBody on every method

@ResponseBody – what we use on the method level to return json not HTML page

Q4: HTTP method categories

* same as question 2: indempotency

Q5: dependency injection – this is how we kept the controller, service, and repository layers separate by @Autowired to autowire aka inject an instance instead of using the new operator

Q5: HTTP status code categories

100s info

200s success

300s redirects

400s client error (like bad request)

500s server error

Q8: Hibernate annotation to indicate primary key is @Id

Q9: Rest principles

1. stateless - this helps us down the road use load balancing to direct requests to any server
2. cacheable
3. layered – the client only knows about one location to send requests
4. uniform interface – one endpoint per resource
5. code on demand – server can send back runnable code for the client to run

Q11: In our Controller how to parse the request

Example: **/authors/{id}**

**public myMethod(@PathVariable int id) { .. }**

**Example : /authors?name=sam**

**public myMethod(@RequestParam String name) { … }**

**Q12: The layer that handles client request is annotated with @Controller**

**Q13: /movies is better than /getmovies bc don’t use the method name**

**/movies is better than / because should be specific**

**/movies/\* is not how we want our user to have to access all (think about how you want them to type it in)**

**Q15: Spring Boot three layers:**

**controller – interacts with client**

**service – handles business logic**

**repository – handles database interactions**

**Q16: Hibernate mapping of class to table**

**@Entity**

**@Table(name = “Employees”) // normally optional if the table name and class name match exactly (NOTE: It will convert class camel-case into snake-case in your database for you)**

**Q17: Hibernate how to tell it our primary key is generated**

**GenerationType.IDENTITY**

**Q18: Hibernate what is it used for?**

**object-relational mapping (aka how to convert our table into our java class)**

**Q19: bean**

* **three rules for Spring to be able to use dependency injection with your class:**
* **1. default constructor**
* **2. getters and setters**
* **3. serializable**

**IoC refers to inversion of Control aka dependency injection**

**Q20: JPA question on how to generate queries that aren’t built in**

1. **write them yourself in @Query annotation**
2. **use the JPA keywords in your method name for it to generate ex; findByNameContaining**

**Q21:**

**@Repository   
public interface extends CrudRepository<Employee, Long> { }**

**Q22: Mockito**

**@Mock creates a mock**

**@InjectMock inject the mock into an actual**

**Q22:**

**@Autowired**

**Q23:**

**Authentication**

1. **what you have (phone message)**
2. **who you are (fingerprint)**
3. **what you know (password)**

**Authorization**

1. **role based**
2. **rule based**

**Q24: Controller layer we can user the @RestController annotation to specify this is a controller and**

**ResponseEntity – this is the class of objects we send back to the client (including http status, headers, and response body)**

**vs**

**RestController – this is the class that interacts with the client listening**

**Q25: In the controller we annotate our methods to listen on certain endpoints using**

**@RequestMapping and specify the method and endpoint**

**@RequestMapping(value = "/shop", method = RequestMethod.GET)**

**Q28: Bind the parameter to the *body* of the request**

**Example for request param: /author?name=sam here use @RequestParam**

**Example for request body: public create(@RequestBody Author author) { } // get the author data from the body**