Full Stack

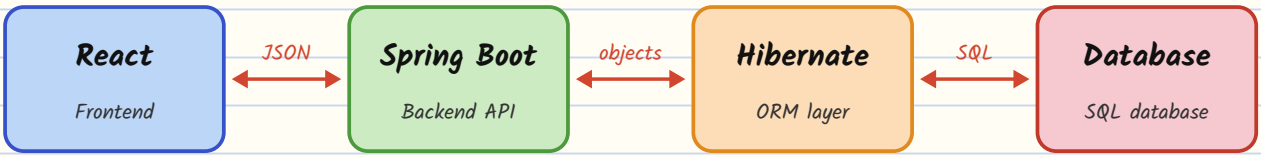
Java · Spring Boot · React · SQL, explained simply

Java Full Stack Notes · easy to understand · free for students

1. Full Stack Overview

- **Full stack** means you can build all parts of a web app: the **frontend** (what users see), the **backend** (logic and APIs) and the **database** (stored data), then deploy it.

IDEA Think of a restaurant. React is the dining hall and menu, Spring Boot is the kitchen, the SQL database is the store room, and the REST API is the waiter who carries orders.



Request goes right, response comes back left

Layer	Technology	Job
Frontend	React, HTML, CSS	Screens, forms, buttons
Backend	Java, Spring Boot	Business logic, REST APIs, security
Database	MySQL or PostgreSQL	Store and search data with SQL
Tools	Maven, Git, Postman, Docker	Build, version, test and ship

- Install: **JDK** (17 or higher), **IntelliJ IDEA** or VS Code, **Maven**, **Node.js**, **Git**, a database and **Postman** to test APIs.

2. Core OOP and Collections

- Java is **compiled** to bytecode and runs on the **JVM**, so the same code runs on any system. It is **statically typed**: every variable has a fixed type.

Name	Meaning
JVM	Runs the bytecode
JRE	JVM plus standard libraries
JDK	JRE plus compiler and tools (install this)

OOP pillar	In Java
Encapsulation	Private fields with public getters and setters
Inheritance	`class Dog extends Animal`
Polymorphism	Overloading (same name, different parameters) and overriding
Abstraction	`interface` and `abstract class`

EXAMPLE Interface and class.

```
interface Shape { double area(); }

class Circle implements Shape {
  private double r;
  Circle(double r) { this.r = r; }
  public double area() { return Math.PI * r * r; }
}
```

➤ Collections

Type	Class to use	Feature
List	ArrayList	Ordered, duplicates allowed
Set	HashSet	Unique items only
Map	HashMap	Key and value pairs
Queue	ArrayDeque	First in, first out

EXAMPLE Use List, Map and a stream.

```
List<String> names = new ArrayList<>();
names.add("Asha");
names.add("Ravi");
Map<Integer, String> map = new HashMap<>();
map.put(1, "Asha");

List<String> a = names.stream()
    .filter(n -> n.startsWith("A")).toList();
```

- **record** is a short class that only holds data: `record UserDto(Long id, String name) {}`. Great for DTOs. **Optional** avoids null errors.
- Exceptions: **checked** must be handled or declared, **unchecked** (RuntimeException) need not. Use `try / catch / finally`.

3. SQL and Databases

- **SQL** is the language used to talk to a relational database. Data lives in **tables** (rows and columns). Popular choices are MySQL and PostgreSQL.

- A **primary key** uniquely identifies a row. A **foreign key** links a row to a row in another table.

Group	Commands
DDL (structure)	CREATE, ALTER, DROP
DML (data change)	INSERT, UPDATE, DELETE
DQL (read)	SELECT
DCL (permission)	GRANT, REVOKE
TCL (transaction)	COMMIT, ROLLBACK

EXAMPLE Create a table and add a row.

```
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE,
  city VARCHAR(30)
);
INSERT INTO users (name, email, city)
VALUES ('Asha', 'asha@mail.com', 'Delhi');
```

EXAMPLE Read, update and delete.

```
SELECT name, city FROM users
WHERE city = 'Delhi' AND id > 5
ORDER BY name ASC
LIMIT 10;

UPDATE users SET city = 'Pune' WHERE id = 1;
DELETE FROM users WHERE id = 1;
```

REMEMBER Always write `WHERE` with UPDATE and DELETE. Without it, `DELETE FROM users;` removes every row.

Constraint	Meaning
PRIMARY KEY	Unique id, never null
FOREIGN KEY	Must match a key in another table
UNIQUE / NOT NULL	No duplicates / value is required
DEFAULT / CHECK	Starting value / rule the value must follow

4. Advanced SQL Queries

- A **JOIN** combines rows from two tables using a matching column.

Join	Result
INNER JOIN	Only rows that match in both tables
LEFT JOIN	All rows from left table, plus matches from the right
RIGHT JOIN	All rows from right table, plus matches from the left
FULL JOIN	All rows from both tables

EXAMPLE Users with their orders.

```
SELECT u.name, o.amount
FROM users u
JOIN orders o ON o.user_id = u.id
WHERE o.amount > 500;
```

EXAMPLE Group and filter groups.

```
SELECT city, COUNT(*) AS total
FROM users
GROUP BY city
HAVING COUNT(*) > 2
ORDER BY total DESC;
```

- Aggregates: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`. **WHERE** filters rows before grouping, **HAVING** filters groups after grouping.
- A **subquery** is a query inside a query: `SELECT * FROM users WHERE id IN (SELECT user_id FROM orders)`.
- An **index** makes searches fast but makes inserts a little slower. Add it on columns you search or join often.
- **Normalization** removes repeated data: 1NF (single values in each cell), 2NF (no partial dependency), 3NF (no dependency between non-key columns).

ACID	Meaning
Atomicity	All steps of a transaction happen, or none
Consistency	Data stays valid after the transaction
Isolation	Transactions do not disturb each other
Durability	Committed data is saved even after a crash

5. Spring Boot Basics

- **Spring Boot** makes Spring easy: it gives **starters** (ready dependency bundles), **auto-configuration** and an **embedded server** (Tomcat). Spring Boot 3 and newer need Java 17 or higher.
- Create a project at `start.spring.io`: Maven, Java, and add **Spring Web**, **Spring Data JPA**, a **database driver** and **Validation**.

IDEA **Dependency Injection**: Spring creates the objects (called **beans**) and hands them to the classes that need them. You do not write `new` yourself.

EXAMPLE Starting point of the app.

```
@SpringBootApplication
public class App {
  public static void main(String[] args) {
    SpringApplication.run(App.class, args);
  }
}
```

EXAMPLE application.properties for MySQL.

```
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/shop
spring.datasource.username=root
spring.datasource.password=secret
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Annotation	Use
`@SpringBootApplication`	Main class, turns on auto-configuration
`@RestController`	Class that handles web requests and returns JSON
`@Service`	Business logic class
`@Repository`	Database access class
`@Component @Bean`	Make any class or method result a Spring bean
`@Autowired`	Inject a bean (constructor injection is better)

Layer	Job	Talks to
Controller	Receive request, send response	Service
Service	Business rules, transactions	Repository
Repository	Read and write the database	Database

- Run with `mvn spring-boot:run` or the run button in the IDE. The API opens at `localhost:8080`.

6. REST APIs with Spring

- **REST** uses HTTP methods on URLs and sends data as **JSON**. One URL stands for one resource, like `/api/users`.

Method	Purpose	Example	Success code
GET	Read	GET /api/users/5	200 OK
POST	Create	POST /api/users	201 Created
PUT	Replace or update	PUT /api/users/5	200 OK
PATCH	Change part	PATCH /api/users/5	200 OK
DELETE	Remove	DELETE /api/users/5	204 No Content

EXAMPLE A CRUD controller.

```
@RestController
@RequestMapping("/api/users")
public class UserController {
  private final UserService service;
  public UserController(UserService service) { this.service = service; }

  @GetMapping
  public List<User> all() { return service.findAll(); }

  @GetMapping("/{id}")
  public User one(@PathVariable Long id) { return service.findById(id); }

  @PostMapping
  public ResponseEntity<User> create(@Valid @RequestBody User u) {
    return ResponseEntity.status(201).body(service.save(u));
  }

  @DeleteMapping("/{id}")
  public void delete(@PathVariable Long id) { service.delete(id); }
}
```

Annotation	Reads from
`@RequestBody`	JSON in the request body, turned into a Java object
`@PathVariable`	Part of the URL, like /users/**5**
`@RequestParam`	Query string, like ?page=1
`@Valid`	Checks rules like @NotBlank and @Email

- Status codes: `400` bad request, `401` not logged in, `403` not allowed, `404` not found, `500` server error.

EXAMPLE One place for all errors.

```
@RestControllerAdvice
public class ApiErrors {
  @ExceptionHandler(NotFoundException.class)
  public ResponseEntity<String> notFound(NotFoundException e) {
    return ResponseEntity.status(404).body(e.getMessage());
  }
}
```

- Return **DTOs** (small data classes), not entities, so you never leak private fields. Test endpoints with Postman or `curl`.

7. Spring Data and Hibernate

- **ORM** maps Java classes to tables. **JPA** is the standard, **Hibernate** is the engine behind it, and **Spring Data JPA** gives ready repositories with almost no code.

EXAMPLE Entity class becomes the users table.

```
@Entity
@Table(name = "users")
public class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @NotBlank
  private String name;

  @Column(unique = true)
  private String email;
  // getters and setters
}
```

EXAMPLE Repository with no SQL written.

```
public interface UserRepository extends JpaRepository<User, Long> {
  List<User> findByCity(String city);
  Optional<User> findByEmail(String email);

  @Query("SELECT u FROM User u WHERE u.name LIKE %:q%")
  List<User> search(@Param("q") String q);
}
```

- Free methods: `save`, `findAll`, `findById`, `deleteById`, `count`, `existsById`. Method names like `findByCityAndAge` become queries.

Relation	Example
@OneToMany / @ManyToOne	One user has many orders
@ManyToMany	Students and courses
@OneToOne	User and profile

EXAMPLE Many orders belong to one user.

```
@ManyToOne
@JoinColumn(name = "user_id")
private User user;
```

- Use `@Transactional` on service methods that change data. For big lists use pagination: `PageRequest.of(0, 10)`.

REMEMBER The N+1 problem: loading a list and then one extra query per item. Fix it with `JOIN FETCH` in the query.

8. Security and Tokens

- **Authentication** checks who you are. **Authorization** checks what you may do. **Spring Security** protects your endpoints.
- Never store plain passwords. Save a **BCrypt** hash. It is one-way, so even you cannot read it.
- **JWT** flow: user logs in, server returns a signed token, React sends it in every request as `Authorization: Bearer token`, and the server checks it.

JWT part	Holds
Header	Algorithm used to sign
Payload	Claims like user id, role, expiry time
Signature	Proof that the token was not changed

EXAMPLE Basic security setup (Spring Security 6).

```
@Configuration
public class SecurityConfig {
  @Bean
  SecurityFilterChain chain(HttpSecurity http) throws Exception {
    http.csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(a -> a
            .requestMatchers("/api/auth/**").permitAll()
            .anyRequest().authenticated());
    return http.build();
  }
  @Bean
  PasswordEncoder encoder() { return new BCryptPasswordEncoder(); }
}
```

- A custom **JWT filter** runs before the default login filter. It reads the token and tells Spring who the user is. Protect by role with `.hasRole('ADMIN')`.

REMEMBER Tokens in `localStorage` are easy but can be stolen by XSS. httpOnly cookies are safer. Keep token life short.

9. React Frontend Basics

- **React** builds UI from small **components** (functions that return JSX). Start with `npm create vite@latest web -- --template react`.

Concept	Meaning
Component	Reusable function returning UI. Name starts with a capital letter
Props	Read-only data from parent to child
State	Data that changes; changing it redraws the component
useEffect	Run code after render, like fetching data
Router	React Router gives pages without reloading

EXAMPLE State and a list.

```
function Users() {
  const [users, setUsers] = useState([]);
  return (
    <ul>
      {users.map(u => <li key={u.id}>{u.name}</li>)}
    </ul>
  );
}
```

- **Controlled input**: value from state and `onChange` updates it. Always give list items a unique `key`. Call hooks only at the top level.

10. Connecting React and Spring

- React calls the Spring API with `fetch` or `axios`. Fetch data in `useEffect` and save it in state.

EXAMPLE GET users.

```
useEffect(() => {
  fetch("http://localhost:8080/api/users")
    .then(res => res.json())
    .then(data => setUsers(data));
}, []);
```

EXAMPLE POST a new user, with a token.

```
fetch("http://localhost:8080/api/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`
  },
  body: JSON.stringify({ name, email })
}).then(res => res.json()).then(u => setUsers([...users, u]));
```

REMEMBER React runs on `localhost:5173` and Spring on `localhost:8080`. These are different origins, so the browser blocks the call until the server allows it (CORS).

EXAMPLE Allow React in Spring (CORS).

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {
  public void addCorsMappings(CorsRegistry r) {
    r.addMapping("/api/**")
     .allowedOrigins("http://localhost:5173")
     .allowedMethods("GET", "POST", "PUT", "DELETE");
  }
}
```

- Keep the API address in an env variable like `VITE_API_URL`. Show a **loading** message and an **error** message while waiting for the server.

II. Project Structure and Deployment

EXAMPLE Clean folder layout.

```
backend/src/main/java/com/shop/
  controller/   service/   repository/
  entity/       dto/       config/      exception/
frontend/src/
  components/   pages/     api/         App.jsx
```

Step	Command or action
Build backend	`mvn clean package` makes a jar in target/
Run backend	`java -jar target/app.jar`
Build frontend	`npm run build` makes the dist folder
Version control	Push code to GitHub, never push passwords
Config per place	`application-prod.properties` and environment variables

EXAMPLE Dockerfile for the backend.

```
FROM eclipse-temurin:21-jre
COPY target/app.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- Deploy the backend on Render, Railway or AWS, the frontend on Vercel or Netlify, and use a managed MySQL or PostgreSQL database.

12. Projects and Practice

Project	What you practice
1. Student management	CRUD, entity, repository, React table
2. Todo app with login	JWT, Spring Security, protected pages
3. Blog platform	Users, posts, comments, relations, joins
4. E-commerce store	Products, cart, orders, transactions, payments
5. Job portal	Roles (admin, user), search, pagination
6. Library system	Many-to-many, validation, error handling

➤ Common mistakes

Mistake	Fix
Forgetting CORS	Allow the React origin in Spring
Plain text passwords	Hash with BCrypt
Sending entities directly	Use DTOs
No validation	Use @Valid and @NotBlank
Passwords inside the code	Use environment variables
UPDATE or DELETE without WHERE	Always add WHERE

➤ Practice questions

1. Difference between JDK, JRE and JVM. Explain the four pillars of OOP.
2. ArrayList vs LinkedList. HashMap vs HashSet. How does HashMap work?
3. Explain INNER JOIN vs LEFT JOIN. WHERE vs HAVING.
4. What are ACID properties and normalization?
5. What is Spring Boot? What is Dependency Injection and a bean?
6. Explain the Controller, Service and Repository layers.
7. What is JPA and Hibernate? What is the N+1 problem?
8. How does JWT authentication work?
9. What is CORS and how do you fix it?
10. Build a CRUD app with React, Spring Boot and MySQL.

- Build one full CRUD app first. Then add login and deploy it. That one project teaches more than ten tutorials.