Spring Boot

a. Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities
    1. Autoconfiguration
    2. An opinionated approach to configuration

b. An extension of the spring framework.
c. "Simplifies spring application development by automatically pre-configured virtually all third-party libraries"
d. This pre-configuration is done by default.
e. Auto-configuration - dependencies of an application configured automatically.
f. Opinionated Approach
g. In software development, it refers to an approach that intentionally reduces flexibility by making choices on behalf of the developer.
h. For example: programmers should decide third party libraries and their versions but Spring Boot will make decisions for programmers related to libraries and it will be applicable for the entire application.

***Spring VS Spring Boot*:**

Spring	**Spring Boot**
Spring supports several build systems like maven, ant and gradle.	Spring Boot recommends maven and gradle rather than ant. Works best with a build system that can pull artifacts published to Maven central.
Spring needs detailed dependencies to be specified by the developer in order to get started.	Spring Boot provides starter dependencies, so developers can get started with a single dependency. For example: Starter dependencies for spring web application which encapsulated - servlet, Apache dependencies and more without conflicts.

***Spring Boot - starter dependencies*:**

1. Starter dependencies are the kind project – which consists set of dependencies required for respective spring framework application.
2. ***List of starter dependencies***:
    a. Spring-boot-starter-web: for development of web applications.
    b. Spring-boot-starter-test: to test application.
    c. Spring-boot-starter- thymeleaf: to develop web applications with thymeleaf, etc.

How to create Spring Boot Project:

- Go to browser and search for "Spring initilizer".
- Select Maven and language as Java. Then select Spring Boot version.
- Specify Group Id & Artifact Id, select required dependencies.
- Click on generate, zip file will get download.
- Extract zip file and open respective file in IDE platform.

Spring Data JPA

1. It significantly reduces the amount of boilerplate code required to implementations for the most commonly used CRUD (create, read/retrieve, update and delete) operations.
2. While using JDBC API directly involves setup data source connection, Statement, Executions of Queries, Process of Resultant Data.
3. Most applications will require a standard set of operations on data (query by name, Id, etc.). Spring data repositories take care of these implementations.
4. By default, implements select * (findAll) and select by ID (findById) methods.
5. Will implement other basic SELECT operations if provided with a corresponding method signature. (Derived query method)
6. E.g. Create JPA repositories by mapping with "Student" class then Define method - **List<student> findByStream(String stream) [Generate query implicitly]**
   Above method allows retrieving all data of students based on column stream and translates query results into instances of the class.

7. Derived query methods can also perform ordering and search on multiple fields.
8. Create, Update and Delete operations can also be implemented. Above task will be performed with JPQL queries.
9. Custom queries can be mapped to methods using **@Query** annotations.
10. For Update, delete operation specify **@Transactional** above JpaRepository and **@Modifying** above user-defined methods.
11. PagingAndSortingRepository: extends CrudRespository and provides support for paging and sorting.
12. JpaRepository: extends Paging and sorting repository and provides JPA methods.

For example:

application.properties

```
spring.application.name=Spring-Data-JPA
#database properties
spring.datasource.driver-class-name= com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/spring-data-jpa?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=sql123

#hibernate properties
#to add dialect
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Entity

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Expense {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        private double amount;
        private String category;
        private String description;
        private LocalDate date;

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

}
```

Repository

```
@Repository
@Transactional
public interface ExpenseRepository extends JpaRepository<Expense, Integer> {

	@Query(value = "SELECT e1 FROM Expense e1 WHERE e1.user.userId = ?1")
	List<Expense> findAllExpenseByUserId(int userId, Sort sort);

	List<Expense> findByCategory(String category, Sort sort);

	List<Expense> findByDateBetween(LocalDate startDate, LocalDate endDate, Sort sort);

	List<Expense> findByAmountBetween(int minAmount, int maxAmount, Sort sort);

	@Query(value = "UPDATE Expense e1 SET e1.amount = ?1 WHERE e1.id = ?2")
	@Modifying
	Expense updateAmountByExpenseId(double amount, int expenseId);

	@Query(value = "SELECT e1 FROM Expense e1 WHERE e1.user.userId = ?1")
	Page<Expense> findAllExpenseByUserIdWithPageable(int userId, Pageable pageable);
}
```

Dao

```
@Component
public class UserDao {
	@Autowired
	private UserRepository userRepository;
	@Autowired
	private ExpenseRepository expenseRepository;

	public User register(User user) {
		return userRepository.save(user);
	}

	public User login(String email, String password) {
		return userRepository.findByEmailAndPassword(email, password).orElse(null);
	}

	public Expense addExpense(Expense expense) {
		return expenseRepository.save(expense);
	}

	public List<Expense> displayAllExpense(int userId){
		//sort based on amount
		Sort sort = Sort.by("amount");
		return expenseRepository.findAllExpenseByUserId(userId, sort);
	}
	public List<Expense> firstPageRecords(){

		//to divide records with respective pages
		Pageable pageable = PageRequest.of(0, 3);

		//find records only for page 1
		Page<Expense> expense = expenseRepository.findAll(pageable);

		//to get all records of first page
		List<Expense> expenses = expense.getContent();
		return expenses;
	}
}
```