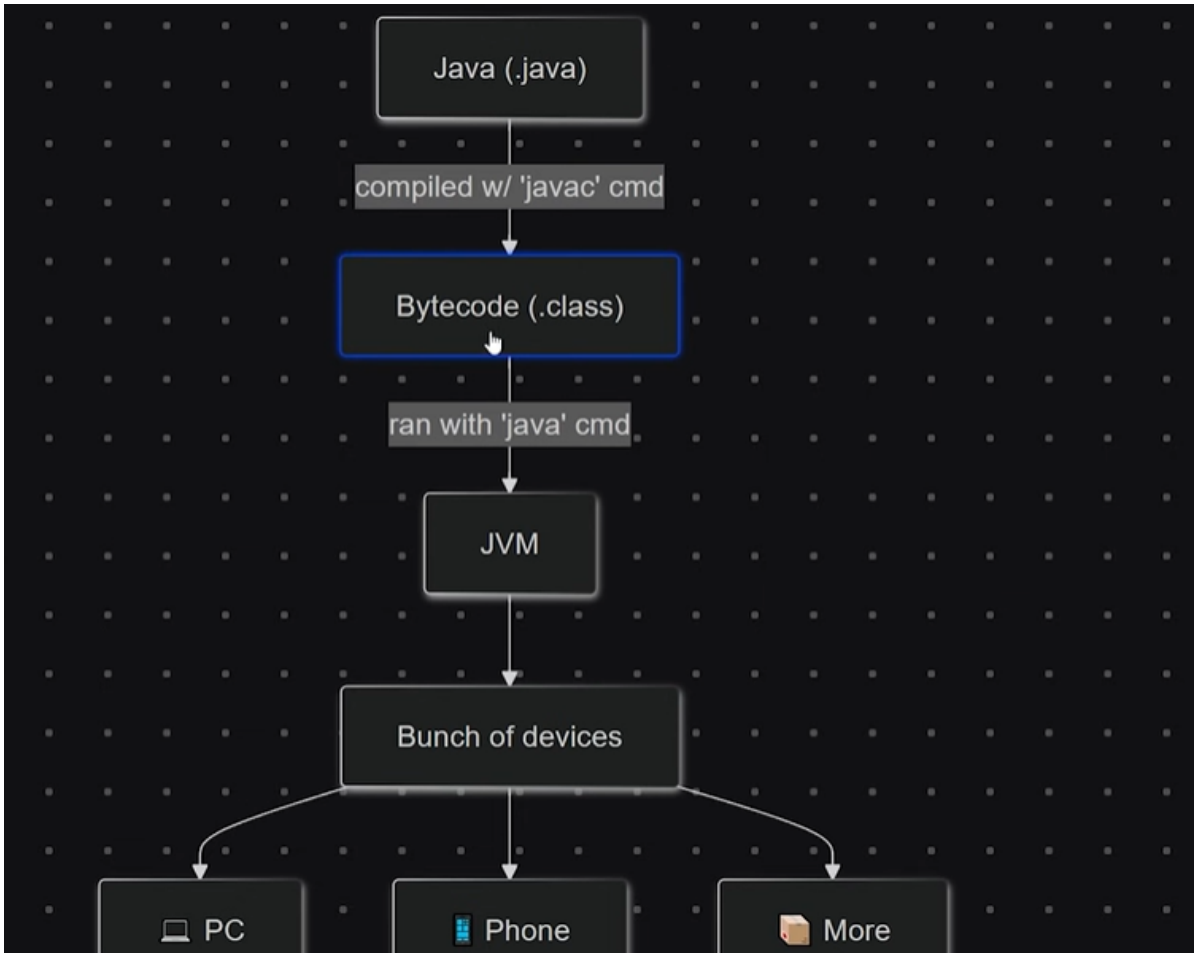
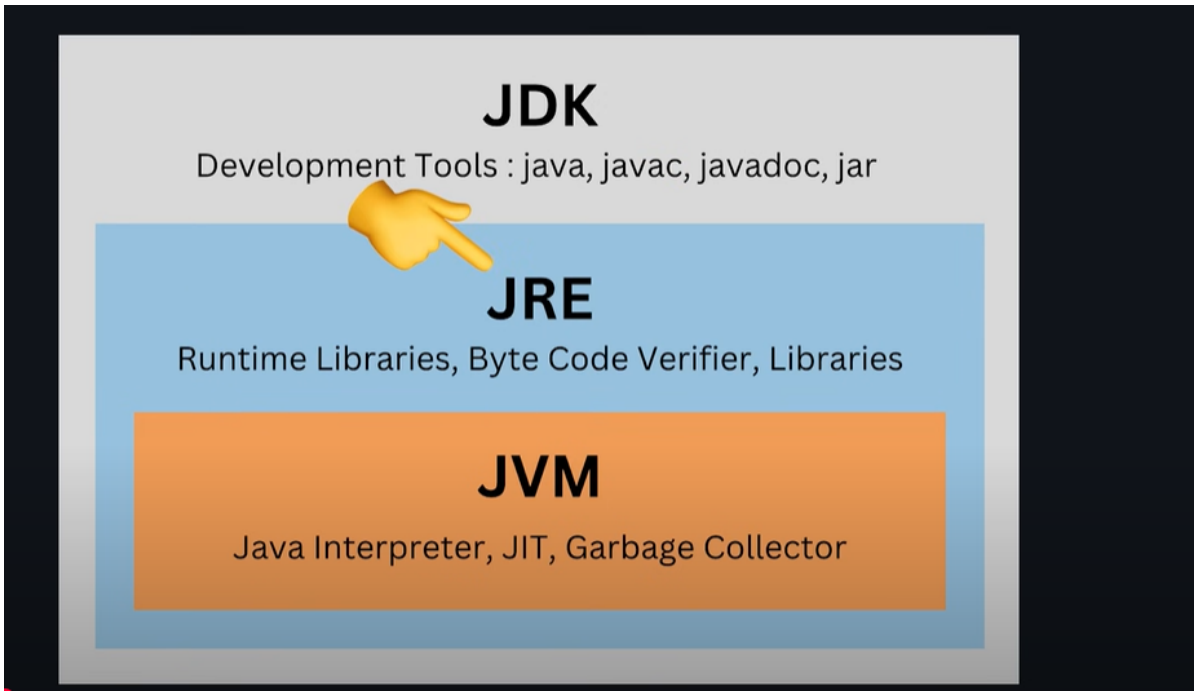
JAVA

▼ **process of running java program**

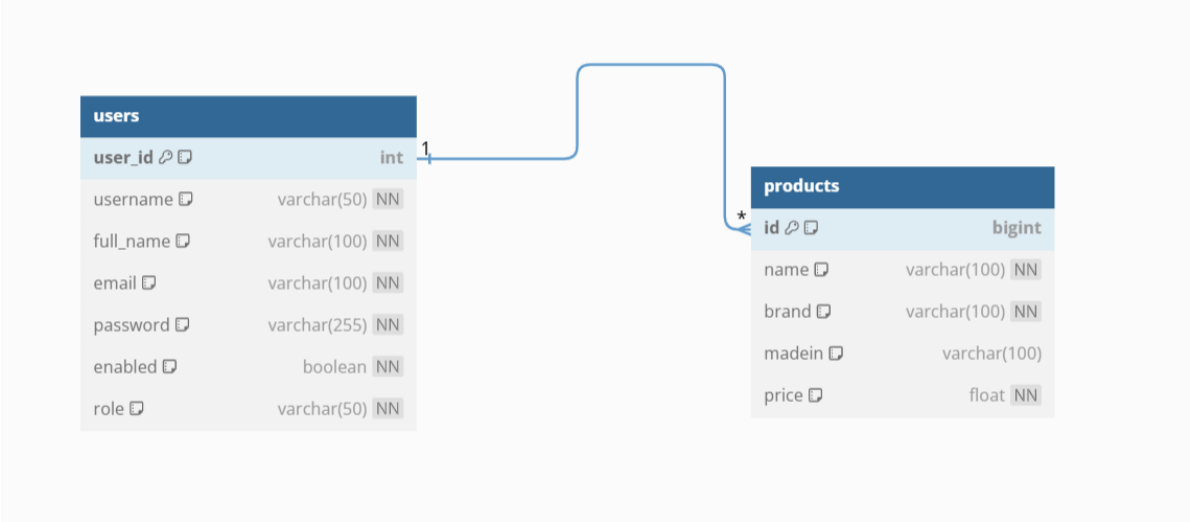




Mavin

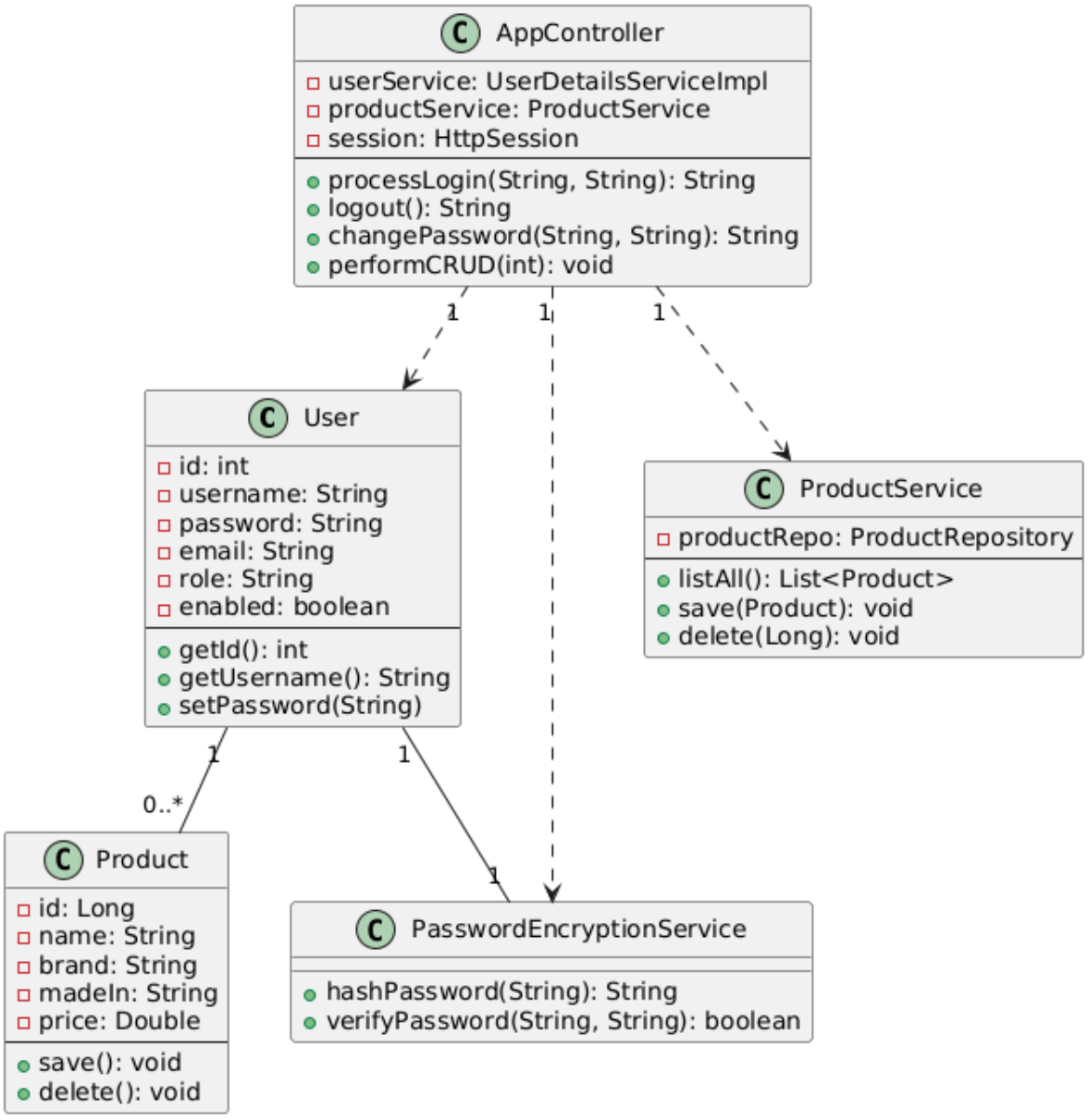
▼ project er-diagram

```
// -Person can register or login
//- Session is maintained
// -after logout the cookie are flushed and session is termi
// - Admins can perform full CRUD operations on poducts.
// - Users can only read product data; no modification or de
//  -User/admin can also change his password
```



▼

▼ Class diagram



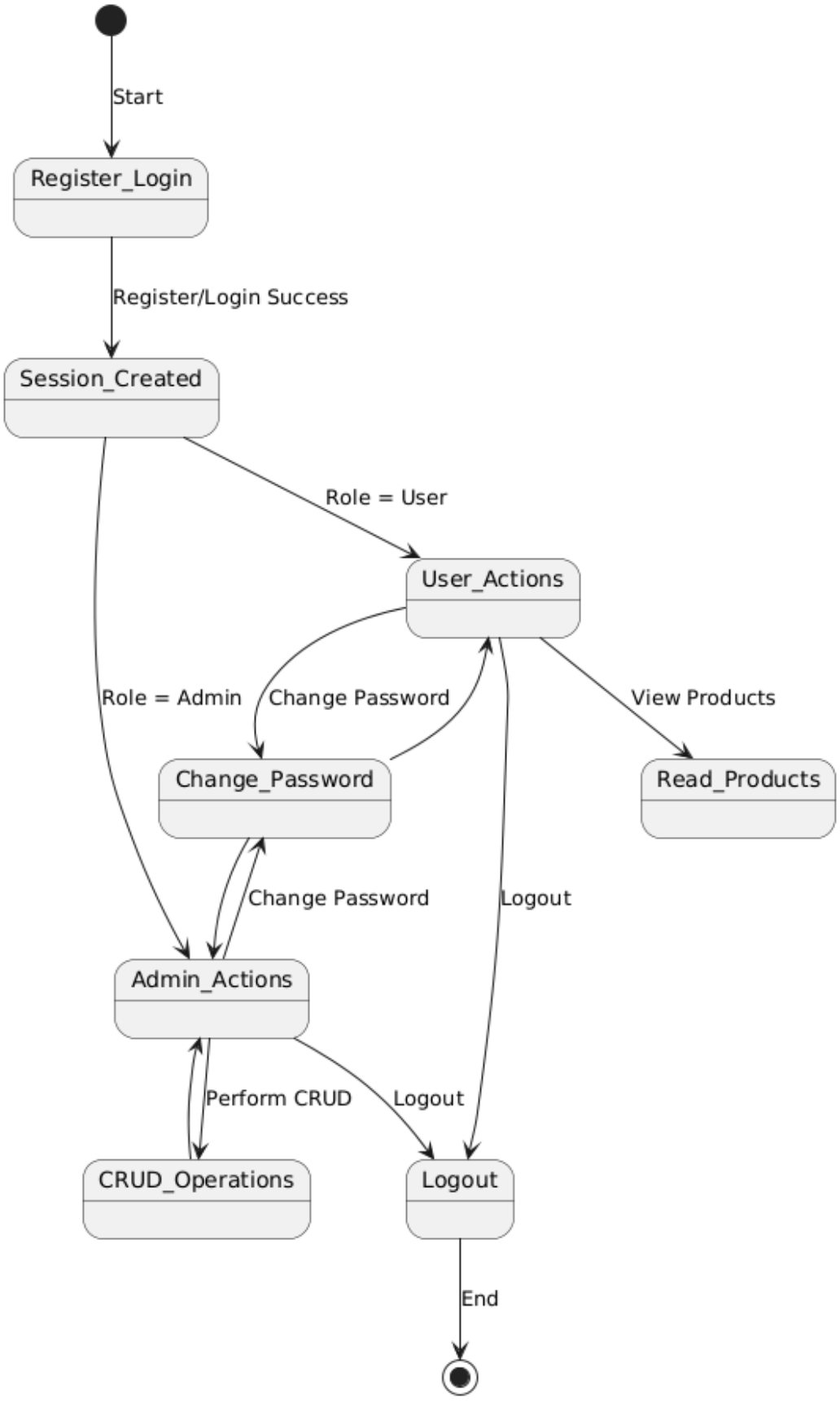
▼ umlcodefor above

```
@startuml
class User {
  - id: int
  - username: String
  - password: String
  - email: String
  - role: String
```

```
  - enabled: boolean
  --
  + getId(): int
  + getUsername(): String
  + setPassword(String)
}

class Product {
  - id: Long
  - name: String
  - brand: String
  - madeIn: String
  - price: Double
  --
  + save(): void
  + delete(): void
}

class PasswordEncryptionService {
  + hashPassword(String): String
  + verifyPassword(String, String): boolean
}

class AppController {
  - userService: UserDetailsServiceImpl
  - productService: ProductService
  - session: HttpSession
  --
  + processLogin(String, String): String
  + logout(): String
  + changePassword(String, String): String
  + performCRUD(int): void
}

class ProductService {
  - productRepo: ProductRepository
```

```
  --
  + listAll(): List<Product>
  + save(Product): void
  + delete(Long): void
}

User "1" -- "0..*" Product
User "1" -- "1" PasswordEncryptionService
AppController "1" ..> User
AppController "1" ..> ProductService
AppController "1" ..> PasswordEncryptionService
@enduml
```

▼ STATE DIAGRAM


Start
Register_Login
Register/Login Success
Session_Created
Role = User
User_Actions
Role = Admin
Change Password
View Products
Change_Password
Read_Products
Change Password
Logout
Admin_Actions
Perform CRUD
Logout
CRUD_Operations
Logout
End

▼ umlcodeforabove

```
@startuml
[*] --> Register_Login : Start
Register_Login --> Session_Created : Register/Login Succes
Session_Created --> User_Actions : Role = User
Session_Created --> Admin_Actions : Role = Admin
User_Actions --> Read_Products : View Products
Admin_Actions --> CRUD_Operations : Perform CRUD
User_Actions --> Change_Password : Change Password
Admin_Actions --> Change_Password : Change Password
Change_Password --> User_Actions
Change_Password --> Admin_Actions
CRUD_Operations --> Admin_Actions
User_Actions --> Logout : Logout
Admin_Actions --> Logout : Logout
Logout --> [*] : End
@enduml
```

▼ @Entity

Purpose: Marks a class as a JPA entity, meaning it maps to a table in a database.

Key Features:
• Each instance of the class represents a row in the database table
• Works with JPA to provide Object-Relational Mapping (ORM)
• Requires an @Id annotation to specify the table's primary key

▼ @Controller

Purpose: Marks a class as a Spring MVC Controller, enabling it to handle web requests.

Key Features:
• Processes incoming HTTP requests and maps them to specific handler methods
• Works with @RequestMapping and similar annotations to define request paths

• Can be used with @ResponseBody to write responses directly to the HTTP response body

▼ Lombok

Purpose: A Java library that reduces boilerplate code by generating commonly used methodslike getters, setters, constructors, etc., at compile time using annotations.

Key Features:
• Simplifies code maintenance and readability
• Common annotations:
  • @Getter / @Setter: Generates getter/setter methods
  • @NoArgsConstructor / @AllArgsConstructor: Generates constructors

▼ JpaRepository

Purpose: A Spring Data interface that provides advanced CRUD and pagination operations forJPA entities.

Key Features:
• Provides methods like findAll, findById, save, deleteById, and custom query methods based on method naming conventions
• Reduces the need for boilerplate code when interacting with databases

▼ @Service

Purpose: Marks a class as a service component in Spring's service layer.

Key Features:
• Indicates that the class contains business logic or service-level functionality
• Often used to implement the core business logic of an application
• Works with dependency injection to enable seamless integration with other components

▼ @Bean

Tells Spring to create an object and manage itBeans can be create inside class marked by annotation @ConfigurationSpring makes it available for dependency injection elsewhere

▼ @Autowired

Purpose: Used in Spring to enable dependency injection. It automatically wires a bean intoanother bean by type.Key Features:Can be applied to constructors, fields, or setter methods.

▼ @Mock

Purpose: Used in unit testing (with frameworks like Mockito) to create mock objects.Key Features:Simulates the behavior of real objects without relying on external systems like databases orAPIs.Commonly used in isolated unit tests to avoid dependencies.

▼ @InjectMocks

Purpose: Used in testing to inject mock objects (created using @Mock) into a class under test.Key Features:Automatically injects mocks into the annotated class by type or name.Simplifies setup for unit tests.

▼ @BeforeEach

Purpose: Marks a method to be executed before each test method in a test class.Key Features:Used for setting up common test data or configurations.

▼ @Test

Purpose: Marks a method as a test method to be executed by the testing framework (e.g.,JUnit).when(...).then(...)Purpose: Part of the Mockito framework, used to define the behavior of mock objects.Key Features:when(...): Specifies the condition or invocation of a mock method.then(...): Specifies what the mock should do when the condition is met (e.g., return a specificvalue or throw an exception).

▼ useful git commands

git checkout -b branch_name ⟹ Creates a new branch with branch name.
<<Make all the changes >>
git add file_name → Optional . Do this if you are creating new files. add the files to the github.
git add directory_name → Optional . Do this if you are creating new directory. Adds all the files of that directory to github.
git diff → Optional. What all changes you made but have not committed

git commit -a -m "commit message" ⟹ commits all the changes in your local branch.
git status → Optional . Provides list of files that are new/modified/deleted.
git diff → Optional. What all changes you made but have not committed
git diff branch_name..master ⟹ Optional. compares branch with master. make sure that you commit your changes first.
git push origin branch_name ⟹ pushes the branch_name to remote(online). make sure you run git commit first.

Other useful commands:
git branch → Check which branch you have in your local. Highlights the branch with * that you are currently working on
git branch -m old_name new_name ⟹ renames local branch from old to new name. You need to run git push command to push it to remote
git reset --hard ⟹ dangarous command. wipes out all changes in your local that you have not committed
git checkout branch_name. ⟹ switch to new branch that is already present. you will have to either run get reset command or git command to switch branches

▼