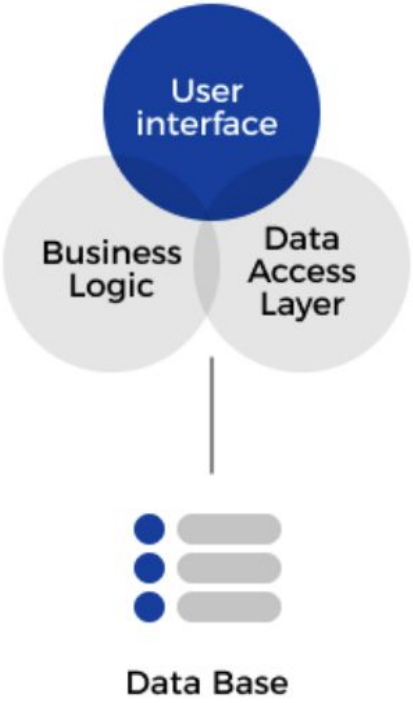
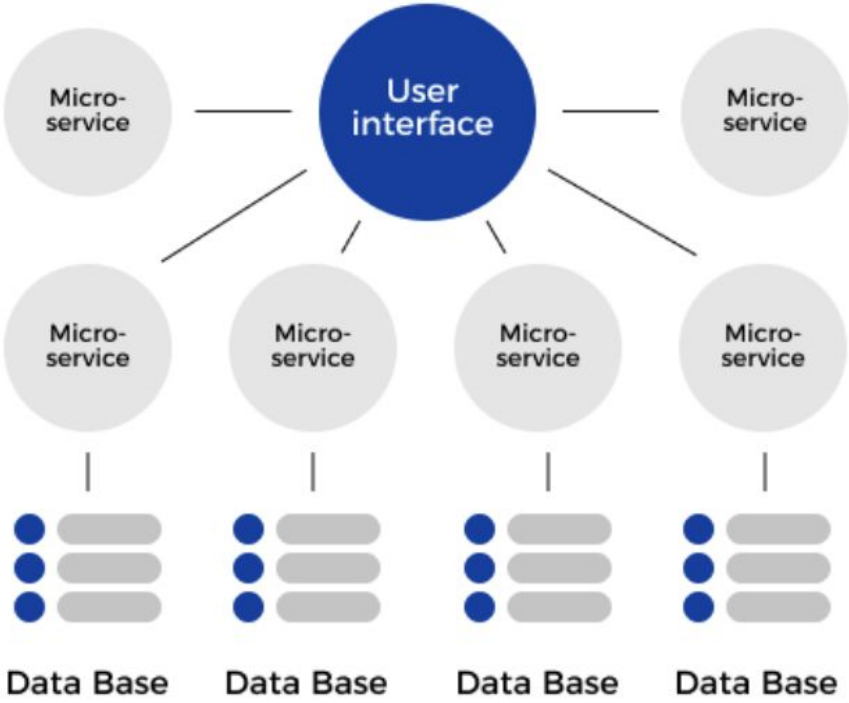
Lecture 2

Illustrate concepts

MONOLITHIC ARCHITECTURE



MICROSERVICE ARCHITECTURE



Spring framework

Features:

- Core technologies: dependency injection, resources, validation, data binding, AOP.
- Data Access: transactions, DAO support, JDBC, ORM, Marshalling XML.
- Integration: remoting, JMS, scheduling, cache.

Spring Framework

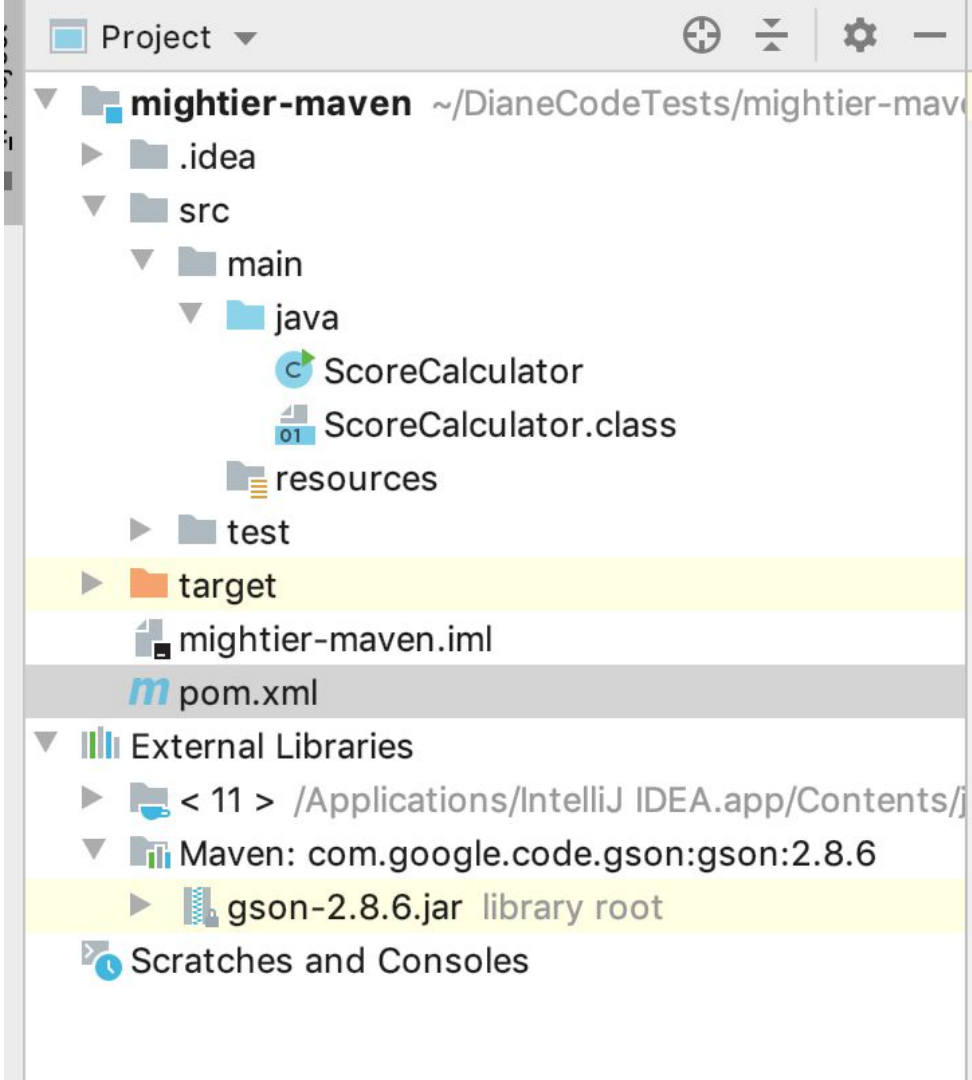
Developer(s)	Pivotal Software
Initial release	1 October 2002; 18 years ago
Stable release	5.2.7.RELEASE / June 9, 2020; 4 months ago[1]
Preview release	5.3.0-RC2 / October 14, 2020; 1 day ago[2]
Repository	Spring Repository
Written in	Java
Platform	Java EE
Type	Application framework
License	Apache License 2.0
Website	spring.io

Spring boot “Hello world” Example:

Building an Application with Spring Boot
https://spring.io/guides/gs/spring-boot/

MAVEN

- Maven is a software project management and comprehension tool.

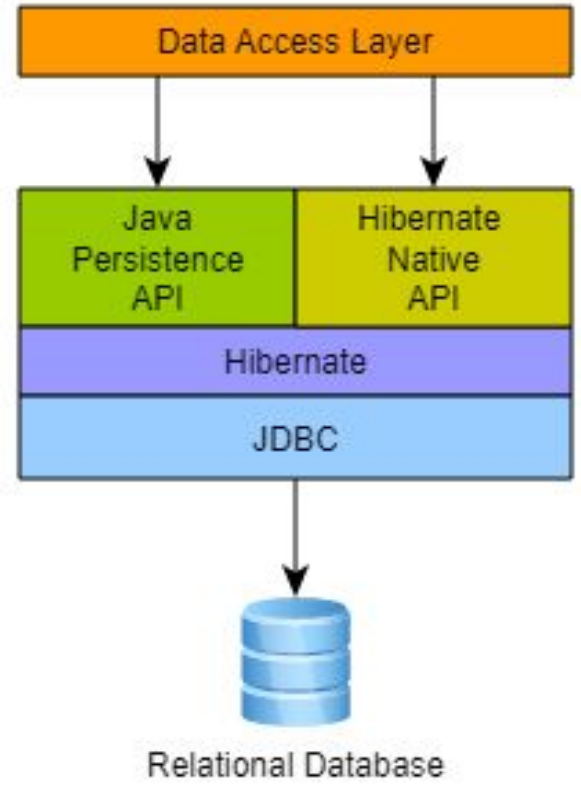


Maven Dependency example

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.3.4.RELEASE</version>
</dependency>
```

Layers between App and Database

- Application
- Jpa
- Jpa implementation (Hibernate/..)
- Jdbc driver (mysql/mssql/oracle)
- DB(mysql/mssql/oracle)



Entity example

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  private Long id;
  private String firstName;
  private String lastName;

  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
```

Repository Example

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

  List<Customer> findByLastName(String lastName);

  Customer findById(long id);
}
```

You can get repository object by:

- Autowiring it
- Pass it in constructor in bean

```java
@Bean
public CommandLineRunner demo(CustomerRepository repository) {
  return (args) -> {
    // save a few customers
    repository.save(new Customer("Jack", "Bauer"));
    repository.save(new Customer("Chloe", "O'Brian"));
    repository.save(new Customer("Kim", "Bauer"));
    repository.save(new Customer("David", "Palmer"));
    repository.save(new Customer("Michelle", "Dessler"));

    // fetch all customers
    log.info("Customers found with findAll():");
    log.info("-------------------------------");
    for (Customer customer : repository.findAll()) {
      log.info(customer.toString());
    }
    log.info("");

    // fetch an individual customer by ID
    Customer customer = repository.findById(1L);
    log.info("Customer found with findById(1L):");
    log.info("--------------------------------");
    log.info(customer.toString());
    log.info("");

    // fetch customers by last name
    log.info("Customer found with findByLastName('Bauer'):");
    log.info("--------------------------------------------");
    repository.findByLastName("Bauer").forEach(bauer -> {
      log.info(bauer.toString());
    });
    // for (Customer bauer : repository.findByLastName("Bauer")) {
    //  log.info(bauer.toString());
    // }
    log.info("");
  };
}
```

Entity With relations Example

```java
@Entity
public class Library {

    @Id
    @GeneratedValue
    private long id;

    @Column
    private String name;

    @OneToOne
    @JoinColumn(name = "address_id")
    @RestResource(path = "libraryAddress", rel="address")
    private Address address;

    // standard constructor, getters, setters
}
```

Entity with relations Example 2

```java
@Entity
public class Address {

    @Id
    @GeneratedValue
    private long id;

    @Column(nullable = false)
    private String location;

    @OneToOne(mappedBy = "address")
    private Library library;

    // standard constructor, getters, setters
}
```

Types of Relations

- @ManyToOne Relation (Single Object)
- @OneToMany Relation (List of objects)
- @OneToOne Relation(Single Object)
- @ManyToMany Relation(List of objects)

Changing the Database step(1)

Change configuration in file :application.properties

```
spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_example

spring.datasource.username=springuser

spring.datasource.password=ThePassword
```

Changing the Database step(2)

Changing JDBC dependency:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Spring boot with JPA examples:

https://spring.io/guides/gs/accessing-data-jpa/

https://www.baeldung.com/spring-data-rest-relationships

https://www.tutorialspoint.com/jpa/jpa_entity_relationships.htm

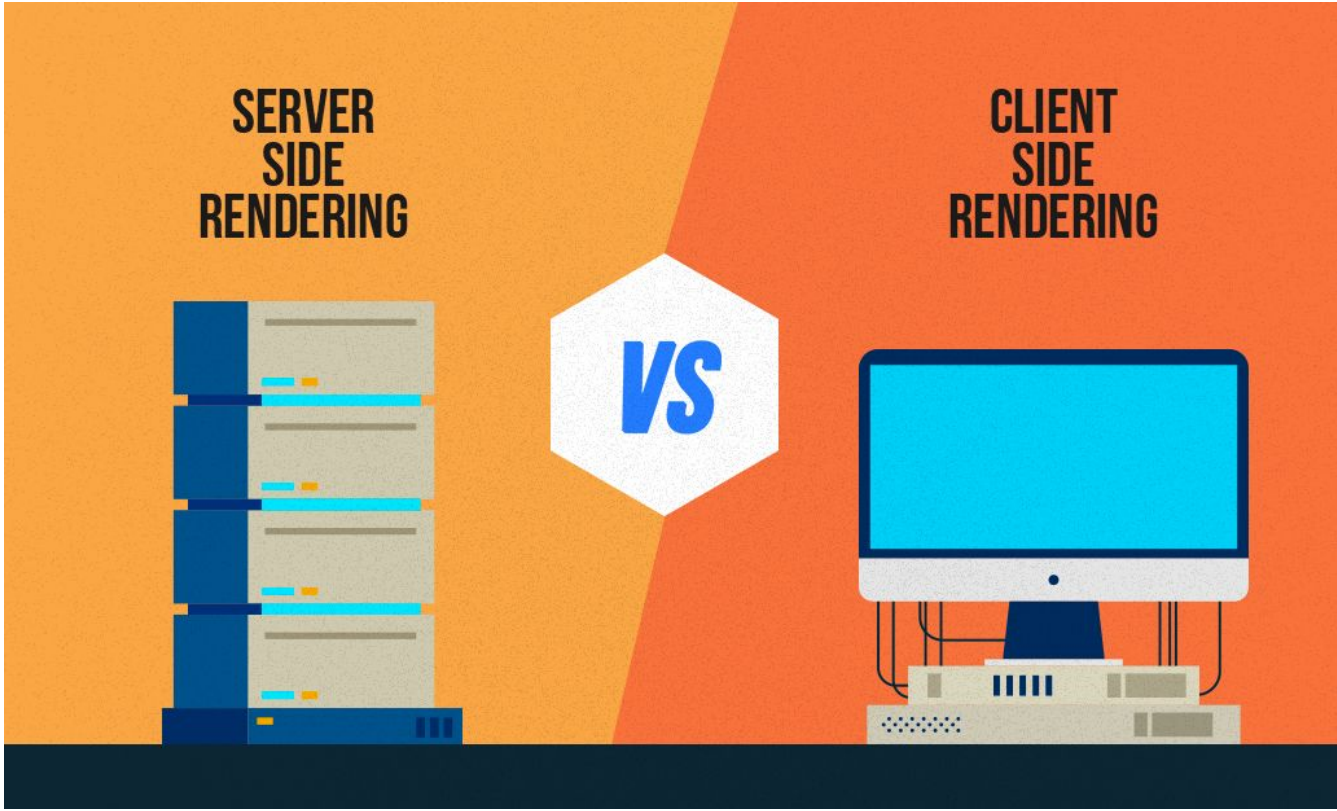
Frontend Languages

- web applications(javascript frameworks)
  - **Angular**
  - **Vue.js**
  - **React**
  - **Knockout**
- mobile applications(native / hybrid)
  - Java
  - Kotlen
  - Swift
  - Objective-C
  - Flatter
  - react.js

Client-side vs. server-side rendering

SSR like WordPress

CSR like Angular



Client-side vs. server-side rendering 1

Server-side pros:

1. Search engines can crawl the site for better SEO.
2. The initial page load is faster.
3. Great for static sites.

Client-side vs. server-side rendering 2

Server-side cons:

- Frequent server requests.
- An overall slow page rendering.
- Full page reloads.
- Non-rich site interactions.

Client-side vs. server-side rendering 3

Client-side pros:

1. Rich site interactions
2. Fast website rendering after the initial load.
3. Great for web applications.
4. Robust selection of JavaScript libraries.

Client-side vs. server-side rendering 4

Client-side cons:

1. Low SEO if not implemented correctly.
2. Initial load might require more time.
3. In most cases, requires an external library.

Next Goals

1. Permissions mgmt
2. Multi threading