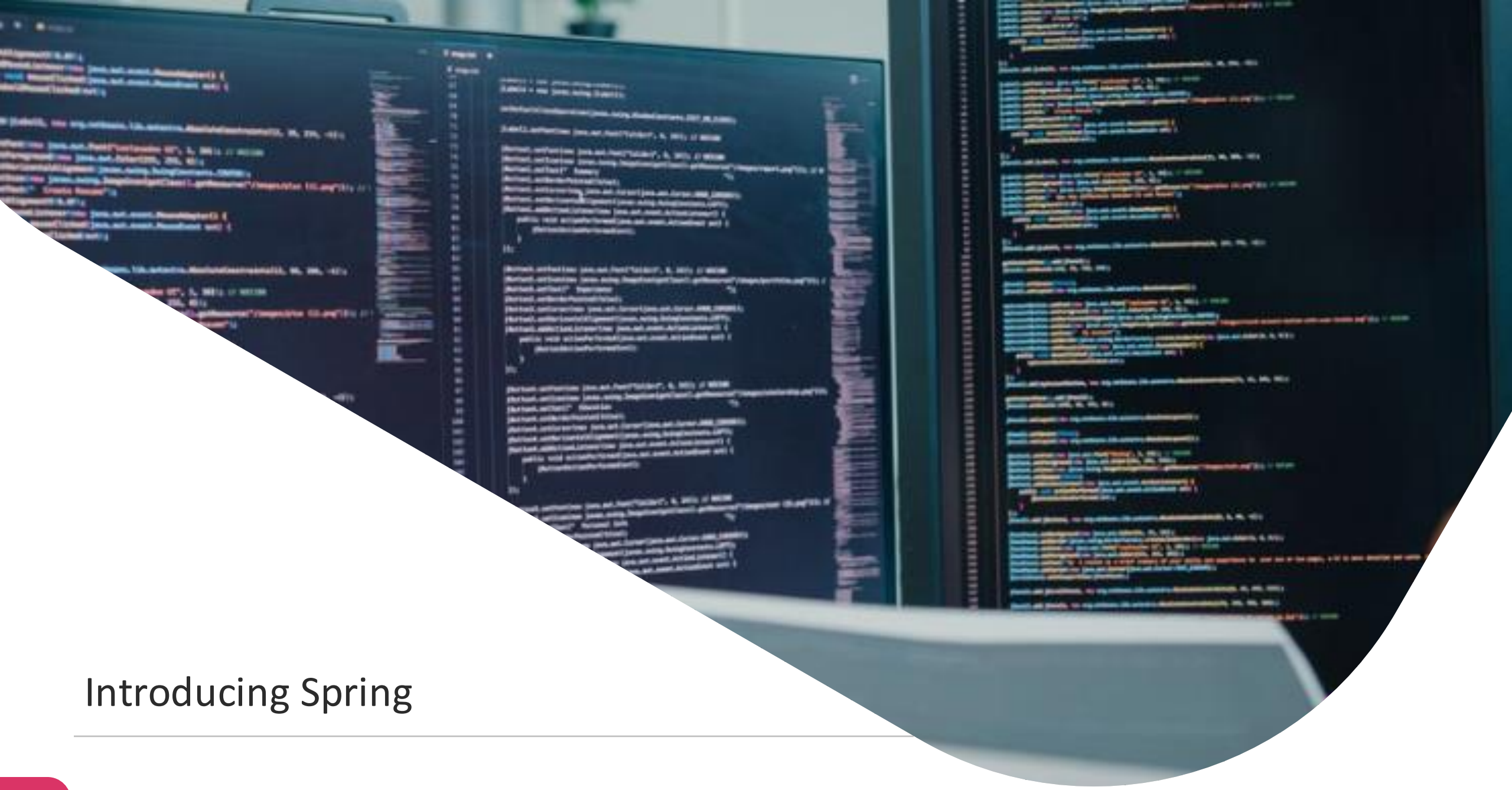


Introducing Spring

Introductions

Janani Ravi – co-founder of Loonycorn

20+ years in software development – worked at Google, Microsoft, Flipkart

Undergrad in Mumbai, grad school at Stanford

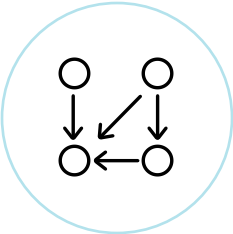
Worked with Skillsoft for 6+ years

Love dogs, have 4 dogs at home (oh and one human kid☺)

Spring Framework

An open-source framework for building Java-based enterprise applications

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

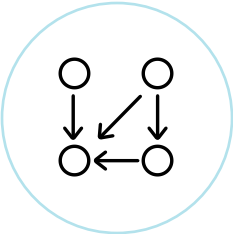
Spring MVC

Spring Security

Transaction management

Spring Cloud

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

Dependency Injection

A design pattern that allows a class to receive its dependencies from external sources rather than instantiating them

Concepts in Dependency Injection

Dependency: Object that another object relies on to function

Injection: The process of providing dependencies to a class

Repository Interface

```
public interface GreetingRepository {
        String getGreeting();
}
```

Repository Implementation

```
@Repository
public class GreetingRepositoryImpl implements GreetingRepository {

    @Override
    public String getGreeting() {
        return "Hello, Spring Boot!";
    }
}
```

Main Application Class

```
@SpringBootApplication
public class Application implements CommandLineRunner {

    private final GreetingService greetingService;

    @Autowired
    public Application(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(greetingService.greet());
    }
}
```

Main Application Class

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

    private final GreetingService greetingService;

    @Autowired
    public Application(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(greetingService.greet());
    }
}
```

Main Application Class

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

    private final GreetingService greetingService;

    @Autowired
    public Application(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(greetingService.greet());
    }
}
```

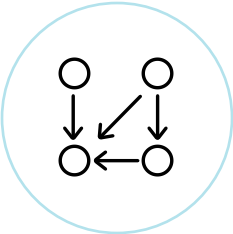
Main Application Class

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

    private final GreetingService greetingService;

    @Autowired
    public Application(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(greetingService.greet());
    }
}
```

Main Application Class

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

    private final GreetingService greetingService;

    @Autowired
    public Application(GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(greetingService.greet());
    }
}
```

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

Aspect-Oriented Programming

Aspect-oriented programming (AOP) in Spring allows for the separation of cross-cutting concerns, like logging, security, or transaction management, from the core business logic of an application

Repository Interface

```
public interface GreetingRepo {
        String getGreeting();
}
```

Repository Implementation with Regular Logging

```
@Repository
public class GreetingRepoImpl implements GreetingRepo {

    // Define a logger
    private static final Logger logger =
          LoggerFactory.getLogger(GreetingRepoImpl.class);

    @Override
    public String getGreeting() {
        // Log before the method execution
        logger.info("getGreeting method called.");

        String greeting = "Hello, Spring Boot!";

        // Log after the method execution
        logger.info("getGreeting method executed successfully”);

        return greeting;
    }
}
```

Repository Implementation with Regular Logging

```
@Repository
public class GreetingRepoImpl implements GreetingRepo {

    // Define a logger
    private static final Logger logger =
          LoggerFactory.getLogger(GreetingRepositoryImpl.class);

    @Override
    public String getGreeting() {
        // Log before the method execution
        logger.info("getGreeting method called.");

        String greeting = "Hello, Spring Boot!";

        // Log after the method execution
        logger.info("getGreeting method executed successfully”);

        return greeting;
    }
}
```

Repository Implementation with Regular Logging

```
@Repository
public class GreetingRepoImpl implements GreetingRepo {

    // Define a logger
    private static final Logger logger =
          LoggerFactory.getLogger(GreetingRepositoryImpl.class);

    @Override
    public String getGreeting() {
        // Log before the method execution
        logger.info("getGreeting method called.");

        String greeting = "Hello, Spring Boot!";

        // Log after the method execution
        logger.info("getGreeting method executed successfully”);

        return greeting;
    }
}
```

Repository Implementation with Logging Using AOP

```
@Repository
public class GreetingRepoImpl implements GreetingRepo {

    @Override
    public String getGreeting() {
        return "Hello, Spring Boot!";
    }
}
```

Logging Aspect

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* demo.GreetingRepoImpl.getGreeting(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
          throws Throwable {
        long startTime = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() -
                  startTime;

        System.out.println(”Cxecuted in " + executionTime + "ms");
        return proceed;
    }
}
```

Logging Aspect

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* demo.GreetingRepoImpl.getGreeting(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
          throws Throwable {
        long startTime = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() -
                  startTime;

        System.out.println(”Cxecuted in " + executionTime + "ms");
        return proceed;
    }
}
```

Logging Aspect

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* demo.GreetingRepoImpl.getGreeting(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
          throws Throwable {
        long startTime = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() -
                  startTime;

        System.out.println(”Cxecuted in " + executionTime + "ms");
        return proceed;
    }
}
```

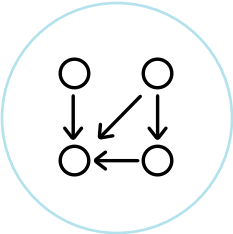
Logging Aspect

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* demo.GreetingRepoImpl.getGreeting(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
          throws Throwable {
        long startTime = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() -
                  startTime;

        System.out.println(”Cxecuted in " + executionTime + "ms");
        return proceed;
    }
}
```

Logging Aspect

```
@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* demo.GreetingRepoImpl.getGreeting(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint)
          throws Throwable {
        long startTime = System.currentTimeMillis();

        Object proceed = joinPoint.proceed();

        long executionTime = System.currentTimeMillis() -
                  startTime;

        System.out.println(”Cxecuted in " + executionTime + "ms");
        return proceed;
    }
}
```

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

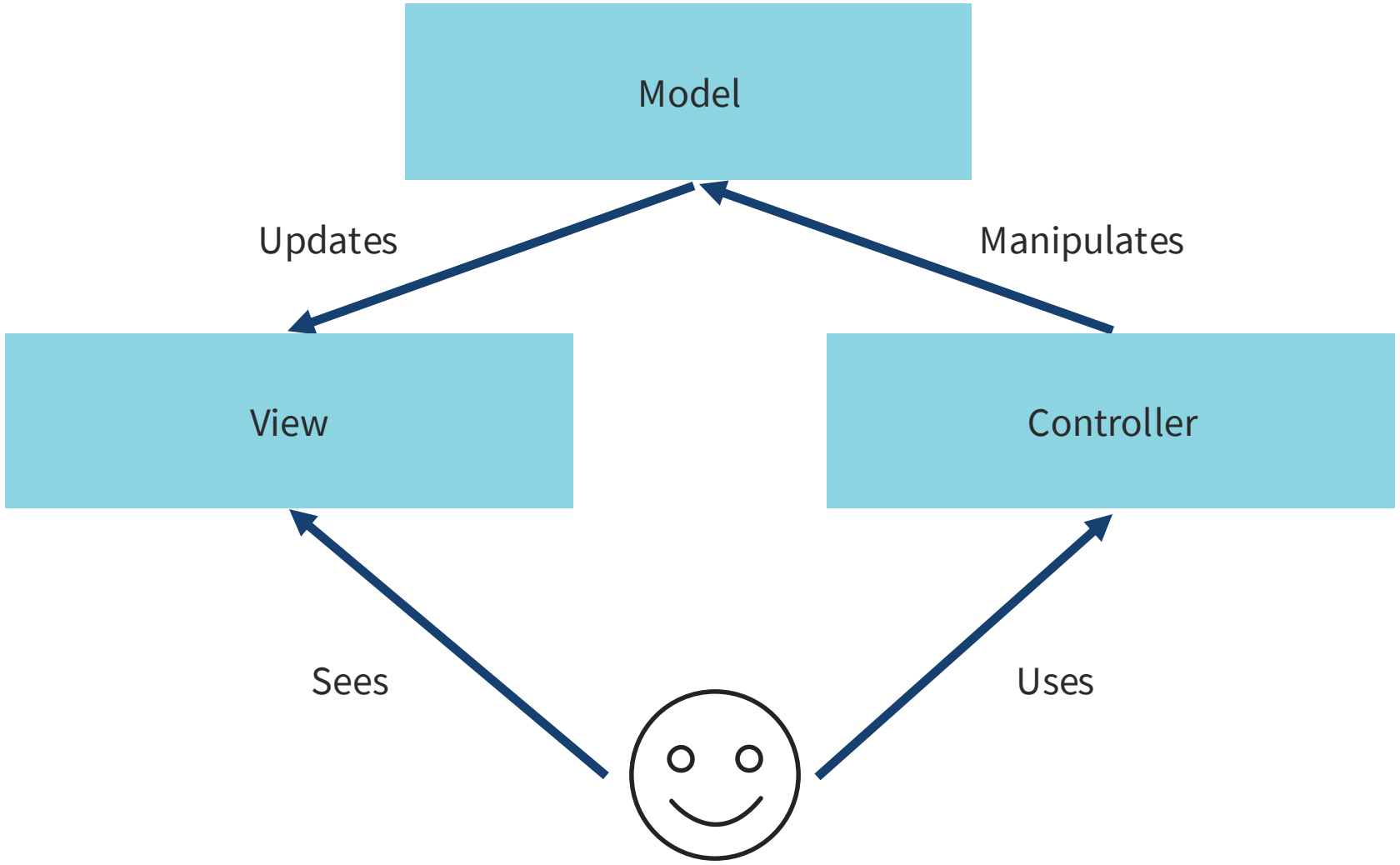
Spring MVC

Spring Security

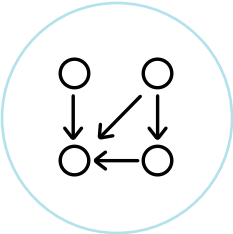
Transaction management

Spring Cloud

Model-View-Controller (MVC)



Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

Transaction Management

Declarative transaction management

Programmatic transaction management

Declarative Transaction Management

```
@Service
public class BankService {

    @Autowired
    private AccountRepository accountRepository;

    @Transactional
    public void transferMoney(
        Long fromAccountId, Long toAccountId, Double amount) {
       // Code to transfer money
    }
}
```

Programmatic Transaction Management

```
@Service
public class TransferService {

    @Autowired
    private PlatformTransactionManager transactionManager;

    @Autowired
    private AccountRepository accountRepository;

    public void transferMoney(
          Long fromAccountId, Long toAccountId, Double amount) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();

        TransactionStatus status = transactionManager.getTransaction(def);

        try {

            transactionManager.commit(status);
        } catch (Exception e) {
            transactionManager.rollback(status);
            throw e;
        }
    }
}
```

Programmatic Transaction Management

```
@Service
public class TransferService {

    @Autowired
    private PlatformTransactionManager transactionManager;

    @Autowired
    private AccountRepository accountRepository;

    public void transferMoney(
          Long fromAccountId, Long toAccountId, Double amount) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();

        TransactionStatus status = transactionManager.getTransaction(def);

        try {

            transactionManager.commit(status);
        } catch (Exception e) {
            transactionManager.rollback(status);
            throw e;
        }
    }
}
```

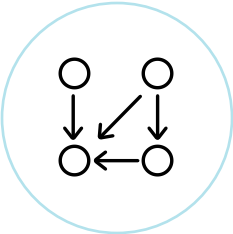
Programmatic Transaction Management

```
@Service
public class TransferService {

    @Autowired
    private PlatformTransactionManager transactionManager;

    @Autowired
    private AccountRepository accountRepository;

    public void transferMoney(
          Long fromAccountId, Long toAccountId, Double amount) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();

        TransactionStatus status = transactionManager.getTransaction(def);

        try {

            transactionManager.commit(status);
        } catch (Exception e) {
            transactionManager.rollback(status);
            throw e;
        }
    }
}
```

Programmatic Transaction Management

```
@Service
public class TransferService {

    @Autowired
    private PlatformTransactionManager transactionManager;

    @Autowired
    private AccountRepository accountRepository;

    public void transferMoney(
          Long fromAccountId, Long toAccountId, Double amount) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();

        TransactionStatus status = transactionManager.getTransaction(def);

        try {

            transactionManager.commit(status);
        } catch (Exception e) {
            transactionManager.rollback(status);
            throw e;
        }
    }
}
```

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

Data Access Framework

- Allow developers to interact with databases in a consistent and simplified way
- Reduce boilerplate code for opening/closing connections, handling exceptions

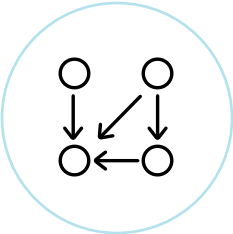
Components of Spring's Data Access Framework

Spring JDBC: Template-based approach to handle database access using raw queries

Spring Data JPA: Simplifies data access and Object Relational Mapping (ORM)

Integrates with the transaction management system to ensure ACID properties for transactions

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

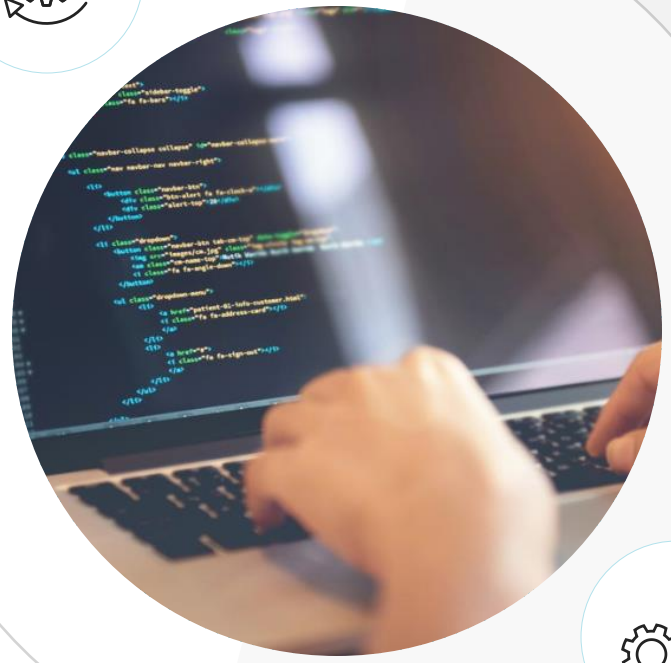
Spring Boot

Spring Boot is a framework built on top of the Spring Framework that simplifies the development of standalone, production-ready Spring applications

Spring Boot Features

Auto-configuration

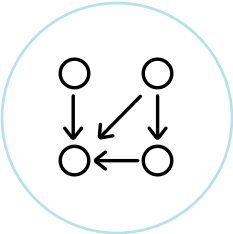
Embedded servers



Spring Boot Starters

Actuators

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

Spring Security

Transaction management

Spring Cloud

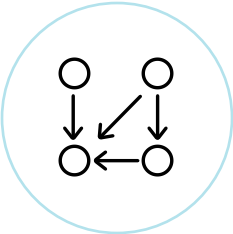
Spring Security

Authentication and authorization in Java applications

Out-of-the-box security features (e.g., form-based login, OAuth 2, method-level security)

Integrates seamlessly with other Spring components

Key Components of Spring



Dependency injection

Data access framework

Aspect-oriented programming

Spring Boot

Spring MVC

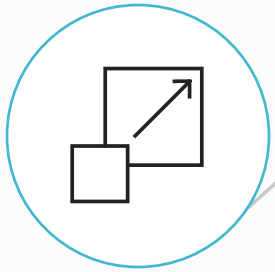
Spring Security

Transaction management

Spring Cloud

Spring Cloud

Tools and frameworks for managing microservices architectures



Cloud-native features – scaling, centralized configurations, service monitoring

Simplifies implementing distributed patterns

Bootcamp Topics

- Introducing Spring Boot
- Setting up our first project
- Spring Initialzr
- Simple REST API
- Path variables, query parameters

Bootcamp Topics

- Dependency injection
- Spring JDBC
- Spring Data JPA
  - Annotations
  - Custom operations
  - Relationships
- REST API with JPA

Bootcamp Topics

- Introducing Thymeleaf
- Simple UI and Forms with Thymeleaf
- Introducing Spring AOP
- Spring actuators