NAGELLA PRAJITHRAJ

Day 4: Spring Core - System Configuration and User Management

Task 1: Configure Spring Beans for User Management and Session Handling

1. **Create the Spring Configuration File:**
   Define your Spring beans in a configuration file, either using XML or Java-based configuration. Here, we'll use Java-based configuration.

```
        @Configuration
       @ComponentScan(basePackages = "com.example.transitapp")
                public class AppConfig {
                 @Bean
       public UserService userService() {
       return new UserServiceImpl();
       }

       @Bean
   public SessionHandler sessionHandler() {
       return new SessionHandler();
   }
}
```

Define User and Session Handling Beans:

Implement the user service and session handler classes.

```
public interface UserService {
    void registerUser(User user);
    User getUser(String username);
}

@Service
public class UserServiceImpl implements UserService {

    private Map<String, User> userStore = new HashMap<>();

    @Override
    public void registerUser(User user) {
        userStore.put(user.getUsername(), user);
    }

    @Override
    public User getUser(String username) {
        return userStore.get(username);
    }
}

@Component
public class SessionHandler {

    private Map<String, HttpSession> sessions = new ConcurrentHashMap<>();

    public void addSession(String sessionId, HttpSession session) {
        sessions.put(sessionId, session);
    }

    public HttpSession getSession(String sessionId) {
        return sessions.get(sessionId);
    }
}
```

Task 2: Set Up Spring's Dependency Injection to Manage Services Related to Traffic Data

Define Traffic Data Service Beans:

Create the services and beans needed for traffic data management.

java

Copy code

```java
@Configuration
public class TrafficDataConfig {

    @Bean
    public TrafficDataService trafficDataService() {
        return new TrafficDataServiceImpl();
    }

    @Bean
    public TrafficDataFetcher trafficDataFetcher() {
        return new TrafficDataFetcher();
    }
}
```

1.

Implement the Traffic Data Service:

Write the service and data fetcher classes.

java

Copy code
```
public interface TrafficDataService {
    TrafficData getTrafficData();
}

@Service
public class TrafficDataServiceImpl implements TrafficDataService {

    @Autowired
    private TrafficDataFetcher trafficDataFetcher;

    @Override
    public TrafficData getTrafficData() {
        return trafficDataFetcher.fetchData();
    }
}

@Component
public class TrafficDataFetcher {

    public TrafficData fetchData() {
        // Fetch traffic data logic
        return new TrafficData();
    }
}
```

2.

Task 3: Establish a Secure Application Context for User Data Processing
Add Security Configuration:
Use Spring Security to secure your application context.
java
Copy code
```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
```

```
            .authorizeRequests()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .antMatchers("/user/**").hasRole("USER")
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
throws Exception {
        auth
            .inMemoryAuthentication()

.withUser("user").password("{noop}password").roles("USER")
                .and()

.withUser("admin").password("{noop}admin").roles("ADMIN");
    }
}
```

1.

Configure User Data Processing:
Securely process user data within the application.
java
Copy code

```
@Service
public class SecureUserService {

    @Autowired
    private UserService userService;

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public void deleteUser(String username) {
        userService.deleteUser(username);
    }

    @PreAuthorize("hasRole('ROLE_USER')")
    public User getUser(String username) {
        return userService.getUser(username);
    }
}
```

2.

Initialize Application Context:
Initialize the Spring application context and integrate all configurations.
java
Copy code

```
public class Application {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class,
TrafficDataConfig.class, SecurityConfig.class);

        UserService userService = context.getBean(UserService.class);
        userService.registerUser(new User("john_doe", "password"));

        TrafficDataService trafficDataService =
context.getBean(TrafficDataService.class);
        System.out.println(trafficDataService.getTrafficData());
    }
}
```

3.

This setup configures Spring Beans for user management and session handling, sets up dependency injection for traffic data services, and establishes a secure application context for processing user data. Adjust the configurations and implementations as needed for your specific application requirements.