Migration

JBoss to Tomcat Migration Plan

Current System Analysis

Application Metrics

- JSP Files: 165
- Custom Tag Libraries: 8
- Spring Components: 81
- Forms: 571
- Security Filters: 3

Technology Stack

- Spring MVC with JSP views
- Custom security filters (CSRF, Authorization)
- Custom TLD implementations
- Sybase Database

Technical Migration Plan & JIRA Tasks

EPIC-1: Security Framework Migration

Current State: Custom security filters and CSRF protection

Target State: Spring Security implementation

[SEC-1] Configure Spring Security Base Setup

```
@Configuration
@EnableWebSecurity
```

```
public class SecurityConfig extends WebSecurityConfigurerAdap
ter {
    @Override
    protected void configure(HttpSecurity http) {
        http
            .authorizeRequests()
                .antMatchers("/api/**").authenticated()
                .antMatchers("/context.root.token/logon.do").
permitAll()
                .anyRequest().authenticated()
            .and()
                .formLogin()
                .loginPage("/context.root.token/login.jsp");
    }
}
```

Acceptance Criteria:

- Spring Security successfully authenticates users
- Existing URLs are properly secured
- Login/logout flow works as expected

[SEC-2] CSRF Protection Migration

```
http.csrf()
    .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOn
lyFalse())
    .ignoringAntMatchers("/api/**");
```

Acceptance Criteria:

- CSRF protection active on all forms
- API endpoints properly configured for CSRF
- Token validation working correctly

EPIC-2: Custom Tag Library Migration

Current State: 8 TLD files with multiple custom tags

Target State: Spring-based tag implementation

[TAG-1] Authentication Tag Migration

```
<!-- Before -->
<app:checkLogon />

<!-- After -->
<sec:authorize access="isAuthenticated()">
    <!-- content -->
</sec:authorize>
```

[TAG-2] Custom Property Tags

```
<!-- Before -->
<app:write name="property" defaultValue="" />

<!-- After -->
<spring:eval expression="property" var="propValue" />
<c:out value="${propValue}" default="" />
```

EPIC-3: Spring MVC Enhancement

Current State: 81 Spring components with .do mappings

Target State: Modern Spring MVC architecture

[MVC-1] Controller Modernization

```
@Controller
@RequestMapping("/loadBalance")
public class LoadBalanceController {
```

```
    @PostMapping("/report")
    public String reportLoadBalances(@Valid @ModelAttribute L
oadBalanceForm form,
            BindingResult result) {
        if (result.hasErrors()) {
            return "reportForm";
        }
        // Processing logic
    }
}
```

[MVC-2] Exception Handling

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ModelAndView handleError(Exception ex) {
        ModelAndView mav = new ModelAndView("error");
        mav.addObject("exception", ex);
        return mav;
    }
}
```

EPIC-4: Tomcat Configuration

Current State: JBoss-specific configurations

Target State: Tomcat-optimized setup

[TOMCAT-1] Database Connection Pool

```
<Resource name="jdbc/myDataSource"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.sybase.jdbc4.jdbc.SybDriver"
```

```
    url="jdbc:sybase:Tds:host:port/database"
    maxTotal="20"
    maxIdle="10"
    maxWaitMillis="10000"/>
```

[TOMCAT-2] Session Management

```
<Context>
    <Manager className="org.apache.catalina.session.PersistentManager">
        <Store className="org.apache.catalina.session.FileStore"
            directory="${catalina.base}/sessions"/>
    </Manager>
</Context>
```

EPIC-5: JSP Modernization

Current State: 165 JSP files with 571 forms

Target State: Spring form tags with validation

[JSP-1] Form Handling Update

```
<form:form method="post" modelAttribute="loadBalanceForm">
    <form:errors path="*" cssClass="alert alert-danger"/>
    <div class="form-group">
        <form:label path="reportDate">Report Date</form:label>
        <form:input path="reportDate" class="form-control"/>
        <form:errors path="reportDate" cssClass="text-danger"/>
    </div>
</form:form>
```

[JSP-2] JSTL Updates

```
<spring:eval expression="T(com.fanniemae.ms3.util.Constants).
REPORTING_STATUS_UNREPORTED"
    var="unreportedStatus" />
<c:if test="${status eq unreportedStatus}">
    <!-- Unreported content -->
</c:if>
```

EPIC-6: API Modernization

Current State: .do pattern endpoints

Target State: RESTful API architecture

[API-1] REST Controller Implementation

```
@RestController
@RequestMapping("/api/v1/balances")
public class LoadBalanceRestController {
    @GetMapping
    public ResponseEntity<List<LoadBalance>> getBalances(
            @RequestParam(required = false) String status) {
        return ResponseEntity.ok(loadBalanceService.findBySta
tus(status));
    }

    @PostMapping("/report")
    public ResponseEntity<ReportResult> generateReport(
            @Valid @RequestBody ReportRequest request) {
        return ResponseEntity.ok(reportService.generateReport
(request));
    }
}
```

[API-2] Error Handling

```
@Data
public class ApiError {
    private HttpStatus status;
    private String message;
    private List<String> errors;
}

@ControllerAdvice
public class RestExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiError> handleNotFound(ResourceNo
tFoundException ex) {
        ApiError error = new ApiError(HttpStatus.NOT_FOUND);
        error.setMessage(ex.getMessage());
        return new ResponseEntity<>(error, HttpStatus.NOT_FOU
ND);
    }
}
```

EPIC-7: Performance Optimization

[PERF-1] Resource Handling

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry r
egistry) {
        registry.addResourceHandler("/resources/**")
                .addResourceLocations("/resources/")
                .setCachePeriod(3600)
                .resourceChain(true)
                .addResolver(new VersionResourceResolver()
```

```
                    .addContentVersionStrategy("/**"));
    }
}
```

[PERF-2] Caching Implementation

```
@Configuration
@EnableCaching
public class CacheConfig {
    @Bean
    public CacheManager cacheManager() {
        SimpleCacheManager cacheManager = new SimpleCacheMana
ger();
        cacheManager.setCaches(Arrays.asList(
            new ConcurrentMapCache("loadBalances"),
            new ConcurrentMapCache("reports")
        ));
        return cacheManager;
    }
}

@Service
public class LoadBalanceService {
    @Cacheable(value = "loadBalances", key = "#status")
    public List<LoadBalance> findByStatus(String status) {
        // Implementation
    }
}
```

Migration Dependencies Checklist

1. Spring Security (spring-security-web, spring-security-config)
2. Tomcat JDBC Connection Pool
3. Spring Cache Abstraction

4. Hibernate Validator
5. Jackson for REST APIs
6. Logback for logging
7. Spring Session for session management

Technical Considerations

1. Session state migration
2. Transaction management changeover
3. Security context propagation
4. Cache synchronization
5. Database connection pool optimization
6. Error handling standardization

Each JIRA story should include:

- Clear acceptance criteria
- Test cases (unit, integration)
- Rollback procedure
- Performance benchmarks
- Security compliance checks