PizzaWorld Dashboard

Technical Documentation

Complete Developer Guide & Code Documentation

Project:	PizzaWorld Business Intelligence Dashboard
Version:	1.0
Framework:	Spring Boot 3.4.6 + Angular 19
Database:	Supabase PostgreSQL
AI Integration:	Google Gemma AI
Repository:	github.com/luigids03/PizzaWorld

Contents

1 System Architecture

1.1 Overview

PizzaWorld is a full-stack business intelligence dashboard with microservices architecture:

- **Backend**: Spring Boot REST API with 90+ endpoints
- **Frontend**: Angular SPA with TypeScript
- **AI Layer**: Google Gemma integration with knowledge base
- **Security**: JWT authentication with role-based access
- **Database**: Supabase PostgreSQL with 15+ materialized views
- **Analytics**: Advanced store, customer, and product analytics
- **Export System**: Comprehensive CSV export for all data views
- **Email Support**: Background email processing system

1.2 Technology Stack

Layer	Technology	Version
Backend Framework	Spring Boot	3.4.6
Security	Spring Security	6.x
Database	PostgreSQL (Supabase)	Latest
ORM	Spring Data JPA	3.x
Frontend Framework	Angular	19
Language	TypeScript	5.7
UI Library	PrimeNG + Tailwind	19 + 3.4
Charts	ApexCharts	3.41
AI Integration	Google Gemma	Latest
Build Tool	Maven + Angular CLI	Latest

2 Backend Architecture

2.1 Package Structure

Listing 1: Backend Package Organization

```
src/main/java/pizzaworld/
          controller/          # REST API Controllers
                AuthController.java
                OptimizedPizzaController.java
                AIController.java
                SupportController.java
          service/             # Business Logic Layer
                OptimizedPizzaService.java
                AIService.java
                GemmaAIService.java
                StaticDocRetriever.java
                UserService.java
                EmailService.java
          repository/          # Data Access Layer
```

```
                OptimizedPizzaRepo.java
                UserRepo.java
        model/              # Entity Classes
                User.java
                ChatMessage.java
                AIInsight.java
                CustomUserDetails.java
        security/           # Security Components
                JwtAuthFilter.java
                SecurityConfig.java
        config/             # Configuration Classes
                CorsConfig.java
                EmailConfig.java
                PizzaConfig.java
        dto/                # Data Transfer Objects
                DashboardKpiDto.java
                ConsolidatedDto.java
                SalesKpiDto.java
        util/               # Utility Classes
            JwtUtil.java
            CsvExportUtil.java
            BcryptTool.java
```

2.2 Core Controllers

2.2.1 AuthController

Handles user authentication and authorization:

Listing 2: Authentication Endpoints

```
@RestController
@RequestMapping("/api")
public class AuthController {

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request)

    @GetMapping("/me")
    public ResponseEntity<?> getCurrentUser(@AuthenticationPrincipal
        CustomUserDetails user)

    @PostMapping("/logout")
    public ResponseEntity<?> logout()

    @PostMapping("/create-test-user")
    public ResponseEntity<?> createTestUser()
}
```

2.2.2 OptimizedPizzaController

Main business logic controller for KPIs and analytics:

Listing 3: Business Endpoints

```
@RestController
@RequestMapping("/api/v2")
public class OptimizedPizzaController {

    @GetMapping("/dashboard/kpis")
    public ResponseEntity<DashboardKpiDto> getDashboardKPIs(
        @AuthenticationPrincipal CustomUserDetails user)
```

```
7
8       @GetMapping("/dashboard/kpis/export")
9       public void exportDashboardKPIs(@AuthenticationPrincipal CustomUserDetails
            user,
10                                      HttpServletResponse response)
11
12      @GetMapping("/orders/recent")
13      public ResponseEntity<List<Map<String, Object>>> getRecentOrders(
14          @RequestParam(defaultValue = "50") int limit,
15          @AuthenticationPrincipal CustomUserDetails user)
16
17      @GetMapping("/analytics/revenue/by-year")
18      public ResponseEntity<List<Map<String, Object>>> getRevenueByYear(
19          @AuthenticationPrincipal CustomUserDetails user)
20
21      @GetMapping("/analytics/customer-lifetime-value")
22      public ResponseEntity<List<Map<String, Object>>> getCustomerLifetimeValue(
23          @RequestParam(defaultValue = "100") Integer limit,
24          @RequestParam(required = false) List<String> states,
25          @RequestParam(required = false) List<String> storeIds,
26          @AuthenticationPrincipal CustomUserDetails user)
27
28      @GetMapping("/stores/{storeId}/analytics/overview")
29      public ResponseEntity<Map<String, Object>> getStoreAnalyticsOverview(
30          @PathVariable String storeId,
31          @RequestParam(required = false) String timePeriod,
32          @AuthenticationPrincipal CustomUserDetails user)
33
34      @GetMapping("/products/kpi")
35      public ResponseEntity<Map<String, Object>> getProductKPI(
36          @RequestParam String sku,
37          @RequestParam(defaultValue = "all-time") String timePeriod,
38          @AuthenticationPrincipal CustomUserDetails user)
39
40      @GetMapping("/kpis/global-store")
41      public ResponseEntity<List<Map<String, Object>>> getGlobalStoreKPIs(
42          @AuthenticationPrincipal CustomUserDetails user)
43
44      @GetMapping("/analytics/store-capacity-v3/summary")
45      public ResponseEntity<List<Map<String, Object>>> getStoreCapacityV3Summary(
46          @RequestParam(required = false) List<String> states,
47          @RequestParam(required = false) List<String> storeIds,
48          @AuthenticationPrincipal CustomUserDetails user)
49
50      @PostMapping("/stores/{storeId}/analytics/compare")
51      public ResponseEntity<List<Map<String, Object>>> getStoreComparePeriods(
52          @PathVariable String storeId,
53          @RequestBody Map<String, Object> requestBody,
54          @AuthenticationPrincipal CustomUserDetails user)
55  }
```

2.2.3 AIController

AI assistant integration with Google Gemma:

Listing 4: AI Endpoints

```
1   @RestController
2   @RequestMapping("/api/ai")
3   public class AIController {
4
5       @PostMapping("/chat")
6       public ResponseEntity<?> chat(@RequestBody ChatRequest request,
```

```
                                    @AuthenticationPrincipal CustomUserDetails user
                                        )

    @PostMapping(path = "/chat/stream", produces = MediaType.
        TEXT_EVENT_STREAM_VALUE)
    public Flux<ServerSentEvent<String>> chatStream(@RequestBody ChatRequest
        request,
                                                    @AuthenticationPrincipal
                                                        CustomUserDetails user)

    @GetMapping("/chat/history/{sessionId}")
    public ResponseEntity<?> getChatHistory(@PathVariable String sessionId,
                                            @AuthenticationPrincipal
                                                CustomUserDetails user)

    @PostMapping("/analyze")
    public ResponseEntity<?> analyzeQuery(@RequestBody AnalyzeRequest request,
                                          @AuthenticationPrincipal
                                              CustomUserDetails user)

    @GetMapping("/insights")
    public ResponseEntity<?> getInsights(@AuthenticationPrincipal
        CustomUserDetails user)

    @GetMapping("/health")
    public ResponseEntity<?> healthCheck()

    @GetMapping("/config")
    public ResponseEntity<?> getPublicConfig()

    @GetMapping("/status")
    public ResponseEntity<?> getAIStatus(@AuthenticationPrincipal
        CustomUserDetails user)

    @PostMapping("/test")
    public ResponseEntity<?> testGoogleAI(@AuthenticationPrincipal
        CustomUserDetails user)
}
```

2.2.4 SupportController

Email support system for customer service:

Listing 5: Support Endpoints

```
@RestController
@RequestMapping("/api")
public class SupportController {

    @Autowired
    private EmailService emailService;

    @PostMapping("/send-support-email")
    public ResponseEntity<?> sendSupportEmail(@RequestBody EmailRequest
        emailRequest) {
        // Background email processing for improved response times
        // Immediate success response - don't wait for email processing

        new Thread(() -> {
            try {
                emailService.sendSupportEmail(
                    emailRequest.from,
                    emailRequest.senderName,
```

```
                    emailRequest.subject,
                    emailRequest.message
                );
            } catch (Exception emailEx) {
                logger.warning("Background email failed: " + emailEx.getMessage
                    ());
            }
        }).start();

        return ResponseEntity.ok(Map.of(
            "success", true,
            "message", "Your message has been sent successfully!"
        ));
    }

    public static class EmailRequest {
        public String to;
        public String from;
        public String senderName;
        public String subject;
        public String message;
    }
}
```

2.3 Service Layer

2.3.1 AIService

Core AI orchestration service:

Listing 6: AI Service Implementation

```
@Service
public class AIService {

    @Autowired
    private GemmaAIService gemmaAIService;

    @Autowired
    private StaticDocRetriever docRetriever;

    // Non-persistent chat sessions (max 20 messages)
    private final Map<String, Deque<ChatMessage>> chatSessions = new
        ConcurrentHashMap<>();

    public ChatMessage processChatMessage(String sessionId, String message, User
         user) {
        // 1. Categorize message (support, analytics, general)
        // 2. Build conversation context
        // 3. Retrieve knowledge snippets
        // 4. Generate AI response with Gemma
        // 5. Apply business consistency validation
        // 6. Store in session (non-persistent)
    }

    public List<AIInsight> generateBusinessInsights(User user) {
        // Generate role-specific insights based on user permissions
    }
}
```

2.3.2 GemmaAIService

Google AI integration service:

Listing 7: Google AI Integration

```
@Service
public class GemmaAIService {

    @Value("${google.ai.api.key}")
    private String apiKey;

    @Value("${google.ai.model:gemma-3n-e2b-it}")
    private String model;

    private final WebClient webClient;

    public String generateResponse(String userMessage, User user,
                                   String category, Map<String, Object>
                                       businessContext) {
        // 1. Build business-specific prompt
        // 2. Call Google AI API
        // 3. Clean and validate response
        // 4. Return contextual answer
    }
}
```

2.4 Security Implementation

2.4.1 JWT Authentication

Listing 8: JWT Security Filter

```
@Component
public class JwtAuthFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) {
        // 1. Extract JWT token from Authorization header
        // 2. Validate token signature and expiration
        // 3. Load user details and permissions
        // 4. Set security context
    }
}
```

2.4.2 Security Configuration

Listing 9: Security Configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session -> session.sessionCreationPolicy(
                STATELESS))
```

```
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/login", "/api/health").permitAll()
                .requestMatchers("/api/admin/**").hasRole("HQ_ADMIN")
                .anyRequest().authenticated()
            )
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter
                .class)
            .build();
    }
}
```

2.5 Database Layer

2.5.1 Materialized Views

Performance-optimized views for analytics:

Listing 10: Key Materialized Views

```
-- KPI Views (Role-based)
kpis_global_hq          -- Company-wide KPIs
kpis_global_state       -- State-level KPIs
kpis_global_store       -- Store-level KPIs

-- Revenue Analytics
revenue_by_year_hq      -- Annual revenue trends
revenue_by_month_hq     -- Monthly analysis
revenue_by_week_hq      -- Weekly patterns
revenue_by_day_hq       -- Daily tracking
revenue_by_hour_hq      -- Hourly analysis

-- Performance Views
store_performance_hq    -- Store ranking
customer_lifetime_value -- CLV analysis
customer_retention_analysis -- Cohort analysis
top_products_hq         -- Product performance
```

2.5.2 Repository Layer

Listing 11: Optimized Repository

```
@Repository
public class OptimizedPizzaRepo {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    // Role-based data access with native SQL
    public DashboardKpiDto getDashboardKpis(User user) {
        String sql = switch(user.getRole()) {
            case "HQ_ADMIN" -> "SELECT * FROM kpis_global_hq";
            case "STATE_MANAGER" -> "SELECT * FROM kpis_global_state WHERE
                state_abbr = ?";
            case "STORE_MANAGER" -> "SELECT * FROM kpis_global_store WHERE
                storeid = ?";
            default -> throw new IllegalArgumentException("Invalid role: " +
                user.getRole());
        };

        return jdbcTemplate.queryForObject(sql,
            new BeanPropertyRowMapper<>(DashboardKpiDto.class),
```

```
            getParametersForRole(user));
    }

    public List<Map<String, Object>> getStoreAnalyticsOverview(User user, String storeId,
                                                                String timePeriod) {
        String sql = buildStoreAnalyticsQuery(user.getRole(), timePeriod);
        return jdbcTemplate.queryForList(sql, storeId);
    }

    public List<Map<String, Object>> getCustomerLifetimeValue(User user, Integer limit,
                                                               List<String> states) {
        String sql = buildCustomerCLVQuery(user.getRole());
        return jdbcTemplate.queryForList(sql, getRoleSpecificParameters(user));
    }

    public List<Map<String, Object>> getProductPerformanceAnalytics(User user,
                                                                     String category,
                                                                     Integer limit) {
        String sql = buildProductAnalyticsQuery(user.getRole(), category);
        return jdbcTemplate.queryForList(sql, limit);
    }

    private Object[] getParametersForRole(User user) {
        return switch(user.getRole()) {
            case "STATE_MANAGER" -> new Object[]{user.getStateAbbr()};
            case "STORE_MANAGER" -> new Object[]{user.getStoreId()};
            default -> new Object[]{};
        };
    }
}
```

3 Frontend Architecture

3.1 Angular Structure

Listing 12: Frontend Structure

```
src/app/
          pages/                    # Feature Modules
                dashboard/           # Main dashboard
                orders/             # Order management
                products/           # Product catalog
                stores/             # Store management
                customer-analytics/ # Customer insights
                delivery-metrics/   # Delivery tracking
                profile/            # User profile
                contact-support/    # Support system
                login/              # Authentication
          core/                   # Core Services
                auth.service.ts     # Authentication
                ai.service.ts       # AI integration
                kpi.service.ts      # KPI management
                auth.guard.ts       # Route protection
                cache.service.ts    # Client caching
                notification.service.ts # Notifications
```

```
                theme.service.ts     # Theme management
          shared/                  # Reusable Components
              ai-chatbot/          # AI chat interface
              loading-popup/       # Loading indicators
              notification/        # Alert system
              sidebar/             # Navigation
```

3.2 Core Services

3.2.1 AuthService

Listing 13: Authentication Service

```
@Injectable({providedIn: 'root'})
export class AuthService {
  private currentUserSubject = new BehaviorSubject<User | null>(null);
  public currentUser$ = this.currentUserSubject.asObservable();

  login(credentials: LoginRequest): Observable<LoginResponse> {
    return this.http.post<LoginResponse>('/api/login', credentials)
      .pipe(
        tap(response => {
          localStorage.setItem('token', response.token);
          this.currentUserSubject.next(response.user);
        })
      );
  }

  hasRole(role: string): boolean {
    const user = this.currentUserSubject.value;
    return user?.role === role;
  }
}
```

3.2.2 AIService

Listing 14: AI Service

```
@Injectable({providedIn: 'root'})
export class AIService {
  private chatHistorySubject = new BehaviorSubject<ChatMessage[]>([]);
  public chatHistory$ = this.chatHistorySubject.asObservable();

  sendMessage(message: string, context?: string): Observable<ChatMessage> {
    return this.http.post<ChatResponse>('/api/ai/chat', {
      sessionId: this.currentSessionId,
      message,
      context
    });
  }

  sendMessageStream(message: string, context?: string): Observable<string> {
    // Handle Server-Sent Events for real-time streaming
    return new Observable<string>((observer) => {
      // Fetch-based streaming implementation
    });
  }

  getChatHistory(): Observable<ChatMessage[]> {
    return this.http.get<any>('/api/ai/chat/history/' + this.currentSessionId);
  }
```

```

  getInsights(): Observable<AIInsight[]> {
    return this.http.get<InsightsResponse>('/api/ai/insights');
  }

  analyzeQuery(query: string, context?: string, type?: string): Observable<
      AnalysisResponse> {
    return this.http.post<AnalysisResponse>('/api/ai/analyze', { query, context,
        type });
  }

  healthCheck(): Observable<boolean> {
    return this.http.get<any>('/api/ai/health');
  }

  testGoogleAI(): Observable<any> {
    return this.http.post<any>('/api/ai/test', {});
  }

  getAIStatus(): Observable<any> {
    return this.http.get<any>('/api/ai/status');
  }

  clearChatSession(): void {
    this.currentSessionId = this.generateSessionId();
    this.chatHistorySubject.next([]);
  }
}
```

3.3 Component Architecture

3.3.1 Dashboard Component

Listing 15: Dashboard Implementation

```
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class DashboardComponent implements OnInit {

  kpis$ = this.kpiService.getDashboardKpis();
  recentOrders$ = this.kpiService.getRecentOrders();
  revenueChart$ = this.kpiService.getRevenueTrend();

  constructor(
    private kpiService: KpiService,
    private authService: AuthService,
    private cdr: ChangeDetectorRef
  ) {}

  ngOnInit() {
    // Auto-refresh every 30 seconds
    interval(30000).pipe(
      startWith(0),
      switchMap(() => this.loadDashboardData())
    ).subscribe();
  }

  private loadDashboardData() {
    return combineLatest([
```

```
      this.kpis$,
      this.recentOrders$,
      this.revenueChart$
    ]);
  }
}
```

3.3.2 AI Chatbot Component

Listing 16: AI Chatbot

```
@Component({
  selector: 'app-ai-chatbot',
  templateUrl: './ai-chatbot.component.html'
})
export class AIChatbotComponent {

  messages$ = this.aiService.chatHistory$;
  isStreaming = false;

  sendMessage(message: string) {
    this.isStreaming = true;

    this.aiService.streamChat(message).subscribe({
      next: (token) => this.appendStreamingToken(token),
      complete: () => this.isStreaming = false,
      error: (error) => this.handleStreamError(error)
    });
  }

  private appendStreamingToken(token: string) {
    // Real-time token streaming for smooth UX
  }
}
```

4 AI Integration

4.1 Google Gemma Integration

4.1.1 AI Configuration

Listing 17: AI Configuration

```
# Google AI Settings
google.ai.api.key=${GOOGLE_AI_API_KEY}
google.ai.model=${GOOGLE_AI_MODEL:gemma-3n-e2b-it}
google.ai.enabled=${GOOGLE_AI_ENABLED:true}
```

4.1.2 Knowledge Base System

Listing 18: Document Retrieval

```
@Service
public class StaticDocRetriever {

    private List<KnowledgeChunk> knowledgeBase;

    @PostConstruct
```

```
    public void loadKnowledgeBase() {
        // Load from resources/knowledge/
        // - business-operations.md
        // - technical-guide.md
        // - faq.md
    }

    public Optional<String> findMatch(String query) {
        return knowledgeBase.stream()
            .filter(chunk -> matchesKeywords(chunk, query))
            .findFirst()
            .map(KnowledgeChunk::getContent);
    }
}
```

4.2 Chat System Features

- **Non-persistent Sessions**: Last 20 messages per session
- **Real-time Streaming**: Server-Sent Events for token-by-token delivery
- **Business Context**: AI responses include live business data
- **Role-based Responses**: Tailored to user permissions
- **Intelligent Fallbacks**: Rule-based responses when AI unavailable
- **Knowledge Integration**: Contextual responses using knowledge base

5 Database Schema

5.1 Core Tables

Listing 19: Core Database Tables

```
-- User Management
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL,
    state_abbr VARCHAR(10),
    store_id INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Orders and Items
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    customer_id INTEGER,
    store_id INTEGER,
    order_date TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(50)
);

CREATE TABLE order_items (
    id SERIAL PRIMARY KEY,
    order_id INTEGER REFERENCES orders(id),
    product_sku VARCHAR(100),
```

```
    quantity INTEGER,
    unit_price DECIMAL(10,2)
);

-- Products and Stores
CREATE TABLE products (
    sku VARCHAR(100) PRIMARY KEY,
    product_name VARCHAR(255),
    category VARCHAR(100),
    size VARCHAR(50),
    price DECIMAL(10,2),
    launch_date DATE
);

CREATE TABLE stores (
    id SERIAL PRIMARY KEY,
    store_name VARCHAR(255),
    city VARCHAR(100),
    state_name VARCHAR(100),
    state_abbr VARCHAR(10),
    zip_code VARCHAR(20)
);
```

5.2 Role-Based Data Access

Listing 20: Role-Based Views

```
-- HQ Admin: All data
CREATE MATERIALIZED VIEW kpis_global_hq AS
SELECT
    SUM(total_amount) as total_revenue,
    COUNT(*) as total_orders,
    AVG(total_amount) as avg_order_value,
    COUNT(DISTINCT customer_id) as total_customers
FROM orders;

-- State Manager: State-filtered data
CREATE MATERIALIZED VIEW kpis_global_state AS
SELECT
    s.state_abbr,
    SUM(o.total_amount) as total_revenue,
    COUNT(o.*) as total_orders,
    AVG(o.total_amount) as avg_order_value
FROM orders o
JOIN stores s ON o.store_id = s.id
GROUP BY s.state_abbr;

-- Store Manager: Store-specific data
CREATE MATERIALIZED VIEW kpis_global_store AS
SELECT
    store_id,
    SUM(total_amount) as total_revenue,
    COUNT(*) as total_orders,
    AVG(total_amount) as avg_order_value
FROM orders
GROUP BY store_id;
```

6 Configuration Management

6.1 Environment Variables

Listing 21: Application Configuration

```
# Database Configuration
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}

# Security
jwt.secret=${JWT_SECRET}

# Email Configuration
spring.mail.username=pizzaworldplus@gmail.com
spring.mail.password=${GMAIL_APP_PASSWORD}

# AI Configuration
google.ai.api.key=${GOOGLE_AI_API_KEY}
google.ai.model=${GOOGLE_AI_MODEL:gemma-3n-e2b-it}

# Performance Settings
spring.datasource.hikari.maximum-pool-size=30
spring.datasource.hikari.minimum-idle=10
```

6.2 CORS Configuration

Listing 22: CORS Setup

```
@Configuration
public class CorsConfig {

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(Arrays.asList(
            "http://localhost:*",
            "https://*.onrender.com",
            "https://pizzaworldplus.tech",
            "https://*.pizzaworldplus.tech"
        ));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "
            DELETE"));
        configuration.setAllowCredentials(true);
        return source;
    }
}
```

7 Deployment DevOps

7.1 Docker Configuration

Listing 23: Production Dockerfile

```
FROM eclipse-temurin:17-jdk-jammy
WORKDIR /app

# Copy and build application
```

```
COPY . .
RUN ./mvnw clean package -DskipTests

# Runtime configuration
EXPOSE 8080
CMD ["java", "-Xmx1400m", "-Xms512m", "-XX:+UseG1GC", "-jar", "target/*.jar"]
```

7.2 Cloud Deployment

Listing 24: Render.com Configuration

```
services:
  - type: web
    name: pizzaworld-backend
    env: docker
    dockerfilePath: ./Dockerfile
    envVars:
      - key: SPRING_PROFILES_ACTIVE
        value: production
      - key: DB_URL
        fromDatabase:
          name: pizzaworld-db
          property: connectionString

  - type: web
    name: pizzaworld-frontend
    runtime: static
    buildCommand: npm install && npm run build:prod
    staticPublishPath: ./dist/frontend
```

8 Testing Strategy

8.1 Backend Testing

Listing 25: Unit Test Example

```
@ExtendWith(MockitoExtension.class)
class AIServiceTest {

    @Mock
    private GemmaAIService gemmaAIService;

    @Mock
    private StaticDocRetriever docRetriever;

    @InjectMocks
    private AIService aiService;

    @Test
    void testChatMessageProcessing() {
        // Given
        User user = new User("testuser", "HQ_ADMIN");
        String message = "What is our revenue?";

        // When
        ChatMessage response = aiService.processChatMessage("session1", message,
            user);

        // Then
        assertThat(response.getMessage()).isNotNull();
```

```
        assertThat(response.getCategory()).isEqualTo("analytics");
    }
}
```

8.2 Frontend Testing

Listing 26: Component Test

```
describe('DashboardComponent', () => {
  let component: DashboardComponent;
  let fixture: ComponentFixture<DashboardComponent>;
  let kpiService: jasmine.SpyObj<KpiService>;

  beforeEach(() => {
    const spy = jasmine.createSpyObj('KpiService', ['getDashboardKpis']);

    TestBed.configureTestingModule({
      declarations: [DashboardComponent],
      providers: [
        { provide: KpiService, useValue: spy }
      ]
    });

    fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    kpiService = TestBed.inject(KpiService) as jasmine.SpyObj<KpiService>;
  });

  it('should load dashboard data on init', () => {
    kpiService.getDashboardKpis.and.returnValue(of(mockKpis));

    component.ngOnInit();

    expect(kpiService.getDashboardKpis).toHaveBeenCalled();
  });
});
```

9 Performance Optimization

9.1 Backend Optimizations

- **Materialized Views**: Pre-computed analytics for instant response
- **Connection Pooling**: HikariCP with 30 max connections
- **JVM Tuning**: G1GC with 2GB heap, string deduplication
- **Query Optimization**: Native SQL with proper indexing
- **Caching**: Application-level caching for business context

9.2 Frontend Optimizations

- **Lazy Loading**: Route-based code splitting
- **OnPush Strategy**: Optimized change detection
- **Virtual Scrolling**: For large data sets

- **HTTP Caching**: Response caching with interceptors
- **Bundle Optimization**: Tree shaking and minification

10 Security Implementation

10.1 Authentication Flow

1. User submits credentials to `/api/login`
2. Server validates credentials against BCrypt hash
3. JWT token generated with user role and permissions
4. Token returned to client with 24-hour expiration
5. Client includes token in Authorization header
6. `JwtAuthFilter` validates token on each request
7. Security context set with user details and roles

10.2 Authorization Matrix

Legend: F=Full, L=Filtered, R=Read-only, X=Denied

Endpoint	HQ	ST	SH
dashboard/kpis	F	L	L
orders	F	L	L
products	F	L	R
stores	F	L	L
stores/analytics/**	F	L	L
analytics/customer-**	F	L	L
analytics/capacity-v3/**	F	L	L
analytics/peak-hours	F	F	X
kpis/global-store	F	L	L
products/kpi	F	L	L
products/trend	F	L	L
chart/time-periods/**	F	F	F
**/compare	F	L	L
**/export	F	L	L
ai/**	F	L	L
send-support-email	F	F	F
create-test-user	F	X	X

Roles: HQ=HQ_ADMIN, ST=STATE_MANAGER, SH=STORE_MANAGER
Access Types:

- **F (Full)**: Complete access to all data across system
- **L (Filtered)**: Role-based data filtering (state/store scope)
- **R (Read-only)**: View access without modification rights
- **X (Denied)**: Access blocked (403 Forbidden)

Key Points:

- All endpoints use `/api/v2/` prefix (abbreviated for space)
- Filtering automatically applied based on user's assigned state/store
- Export functions inherit same permissions as data endpoints
- Peak hours restricted to management levels only
- Support email and time utilities available to all roles