Workforce Forecasting System

Comprehensive Technical Documentation

A complete technical guide to the workforce forecasting platform

Workforce Forecasting System - Comprehensive Technical Documentation

Abstract

The Workforce Forecasting System is an enterprise-grade intelligent workforce management platform that leverages advanced machine learning algorithms to predict workforce demand, optimize shift scheduling, and provide real-time analytics for organizational planning. This system implements a microservices architecture integrating a modern reactive web application with state-of-the-art predictive models to enable data-driven decision-making in human resource management. The platform supports multiple machine learning models including Linear Regression, Random Forest, XGBoost, and LSTM networks, with automated model selection based on performance metrics.

_

Table of Contents

1. [Project Overview](#project-overview)

2. [System Architecture](#system-architecture)

3. [Technology Stack](#technology-stack)

4. [Project Structure](#project-structure)

5. [Database Design](#database-design)

6. [API Specifications](#api-specifications)

7. [Machine Learning Implementation](#machine-learning-implementation)

8. [Security Architecture](#security-architecture)

9. [Performance Optimization](#performance-optimization)

10. [Scalability & High Availability](#scalability--high-availability)

11. [Monitoring & Observability](#monitoring--observability)

12. [Testing Strategy](#testing-strategy)

13. [CI/CD Pipeline](#cicd-pipeline)

14. [Deployment Architecture](#deployment-architecture)

15. [Installation & Setup](#installation--setup)

16. [Configuration Management](#configuration-management)

17. [Troubleshooting Guide](#troubleshooting-guide)

18. [Future Roadmap](#future-roadmap)

_

Project Overview

Problem Statement

Organizations face significant challenges in workforce planning due to:

• Unpredictable demand fluctuations

• Inefficient shift scheduling leading to overstaffing/understaffing

• Lack of data-driven decision-making tools

• Manual processes prone to human error

• Inability to forecast workforce requirements accurately

Solution Overview

The Workforce Forecasting System addresses these challenges through:

• **Predictive Analytics**: ML models forecast workforce demand with high accuracy

• **Real-time Monitoring**: Live dashboards track workforce metrics

• **Automated Scheduling**: AI-powered shift optimization

• **Comparative Analysis**: Multiple model evaluation for best fit

• **Comprehensive Reporting**: PDF reports for stakeholder communication

Key Features

• **Real-time Analytics Dashboard**

• Live workforce metrics monitoring

• Attendance pattern analysis

• Capacity utilization tracking

• KPI cards with drill-down capabilities

• Interactive charts and visualizations

• **AI-Powered Forecasting**

• Multiple ML models (Linear Regression, Random Forest, XGBoost, LSTM)

• Automated model selection based on performance

• Hyperparameter optimization

• Time-series forecasting capabilities

• Confidence intervals for predictions

• **Smart Scheduling**

• Shift optimization interface

• Manual scheduling adjustments

• Shift pattern analysis

• **Model Comparison**

• Comparative evaluation framework

• Performance metrics visualization

• Model benchmarking

• Algorithm selection interface

• **Performance Monitoring**

• Real-time system health metrics

• Model performance tracking

• Microservice health monitoring

• KPI card tracking

• Performance metrics dashboard

• **Interactive Reports**

• PDF generation with iText 7

• Report generation interface

• Analytics reporting

• Data export capabilities

• **Role-Based Access Control**

• Admin: Full system access

• JWT-based authentication

• Secure API endpoints

• User management

• **Modern UI/UX**

• Responsive design with Vue.js 3

• PrimeVue UI components

• Interactive charts with Chart.js

• Clean, intuitive interface

• Real-time data updates

System Requirements

Functional Requirements

ID	Requirement	Priority
FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low

FR-1	User authentication with JWT	High
FR-10	Shift optimization	Low

FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium

FR-2	Role-based access control	High
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-10	Shift optimization	Low

Non-Functional Requirements

ID	Requirement	Target
NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-7	Backup	Daily automated backups

_

System Architecture

High-Level Architecture

The system implements a microservices architecture following the principles of Domain-Driven Design (DDD) and Event-Driven Architecture (EDA). The architecture is designed for scalability, maintainability, and fault isolation.

```
■■■
```

```
■■■
```

```
User Upload → Frontend → Backend → Python ML Service
     ■            ■          ■            ■
     ■            ■          ■            ■■ Data Preprocessing
     ■            ■          ■            ■■ Model Selection
     ■            ■          ■            ■■ Prediction Generation
     ■            ■          ■            ■■ Result Serialization
     ■            ■          ■
     ■            ■          ■■ Store in Database
     ■            ■          ■■ Return to Frontend
     ■            ■
     ■            ■■ Display Results
     ■■ View Dashboard
```

Authentication Flow

```
User Credentials → Frontend → Backend (Auth Controller)
     ■                  ■            ■
     ■                  ■            ■■ Validate Credentials
     ■                  ■            ■■ Generate JWT Token
     ■                  ■            ■■ Return Token + User Info
     ■                  ■
     ■                  ■■ Store Token in localStorage
     ■                  ■■ Redirect to Dashboard
     ■
     ■■ Subsequent Requests (JWT in Header)
```

Component Overview

1. Frontend Application (Vue.js 3)

Responsibilities:

• User interface rendering

• Client-side routing

• State management

• API communication

• Data visualization

Key Technologies:

• Vue.js 3 Composition API

• TypeScript for type safety

• PrimeVue for UI components

• Chart.js for data visualization

• Axios for HTTP requests

• Pinia for state management (future)

Design Patterns:

• Composition API for reusable logic

• Provider/Inject for dependency injection

• Suspense for async components

• Teleport for portal rendering

2. Backend Application (Spring Boot)

Responsibilities:

• REST API endpoints

• Business logic implementation

• Data persistence

• Authentication/authorization

• External service integration

• PDF generation

Key Technologies:

• Spring Boot 3.x

• Spring Security with JWT

• Spring Data JPA

• Hibernate ORM

• iText 7 for PDF generation

• Maven for dependency management

Design Patterns:

• Repository Pattern for data access

• Service Layer Pattern for business logic

• DTO Pattern for data transfer

• Builder Pattern for object construction

• Strategy Pattern for algorithm selection

3. Python ML Service (FastAPI)

Responsibilities:

• Model training

• Prediction generation

• Model evaluation

• Feature engineering

• Data preprocessing

Key Technologies:

• FastAPI for REST API

• scikit-learn for traditional ML

• TensorFlow/Keras for deep learning

• XGBoost for gradient boosting

• pandas for data manipulation

• numpy for numerical computing

Design Patterns:

• Factory Pattern for model creation

• Strategy Pattern for algorithm selection

• Template Method Pattern for training pipeline

• Observer Pattern for progress tracking

4. Database (MySQL)

Responsibilities:

• Persistent data storage

• Transaction management

• Data integrity

• Query optimization

• Backup and recovery

Key Features:

• InnoDB for transactional support

• Indexing for query performance

• Foreign key constraints

• Stored procedures (future)

• Database triggers (future)

Security Architecture

Authentication Layer

```
■■■
```

```
■■■
```

• **Framework**: Vue.js 3 with Composition API

• **Language**: JavaScript/TypeScript

• **UI Components**: PrimeVue

• **Charts**: Chart.js

• **HTTP Client**: Axios

• **State Management**: Vue Reactive API

• **Routing**: Vue Router

• **Build Tool**: Vite

• **Implemented Pages**: Dashboard, AI Models, Forecasting, Analytics, Benchmark, Capacity Planning, Employees, Monitor, Notifications, Reports, Settings, Shift Optimization

Backend

• **Framework**: Spring Boot 3.x

• **Language**: Java 17+

• **ORM**: Hibernate/JPA

• **Security**: Spring Security with JWT

• **Database**: MySQL

• **PDF Generation**: iText 7

• **Build Tool**: Gradle

• **Implemented Controllers**: Dashboard, Authentication, Training, Prediction, Analytics, Benchmark, Capacity, Employee, Forecasting, KPI Card, Microservice Health, Monitor, Notification, Optimization, Report, Settings, Shift Optimization

Python ML Service

• **Framework**: FastAPI

• **ML Libraries**: scikit-learn, TensorFlow/Keras, XGBoost

• **Data Processing**: pandas, numpy

- **Visualization**: matplotlib, seaborn
- **API**: Uvicorn server

_

Project Structure

Frontend Structure

```
Frontend/workforce-forecaasting-x/
■■■ src/
■   ■■■ auth/                          # Authentication
■   ■   ■■■ loginComponent.vue
■   ■   ■■■ authService.ts
■   ■   ■■■ authService.ts
■   ■■■ component/                     # Shared components
■   ■   ■■■ topMenuBarComponent.vue
■   ■■■ homeSetting/                   # Home settings page
■   ■   ■■■ homeSettingComponent.vue
■   ■   ■■■ homeSettingAPI.ts
■   ■■■ settings/                     # Main application pages
■   ■   ■■■ aiModels/                 # AI model management
■   ■   ■   ■■■ aiModelsComponent.vue
■   ■   ■   ■■■ aiModelAPI.ts
■   ■   ■   ■■■ aiModel.css
■   ■   ■■■ analyze/                  # Analytics
■   ■   ■   ■■■ analyticsComponent.vue
■   ■   ■   ■■■ analyticsAPI.ts
■   ■   ■■■ benchmark/                # Benchmarking
■   ■   ■   ■■■ benchmarkComponent.vue
■   ■   ■   ■■■ benchmarkAPI.ts
■   ■   ■■■ capacityPlanning/         # Capacity planning
■   ■   ■   ■■■ capacityPlanningComponent.vue
■   ■   ■   ■■■ capacityPlanningAPI.ts
■   ■   ■■■ dashboard/                # Main dashboard
■   ■   ■   ■■■ dashboardComponent.vue
■   ■   ■   ■■■ dashboardAPI.ts
■   ■   ■   ■■■ dashboardComponent.css
■   ■   ■■■ employees/                # Employee management
```

```
■   ■   ■   ■■■ employeesComponent.vue
■   ■   ■   ■■■ employeesAPI.ts
■   ■   ■■■ forecasting/              # Forecasting interface
■   ■   ■   ■■■ foreCastingComponent.vue
■   ■   ■   ■■■ foreCastingAPI.ts
■   ■   ■■■ monitor/                 # System monitoring
■   ■   ■   ■■■ monitorComponent.vue
■   ■   ■   ■■■ monitorAPI.ts
■   ■   ■■■ notification/             # Notifications
■   ■   ■   ■■■ notificationComponent.vue
■   ■   ■   ■■■ notificationAPI.ts
■   ■   ■■■ reports/                 # Reports generation
■   ■   ■   ■■■ reportComponent.vue
■   ■   ■   ■■■ reportAPI.ts
■   ■   ■■■ settings/                # Application settings
■   ■   ■   ■■■ setttingsComponent.vue
■   ■   ■   ■■■ settingsService.ts
■   ■   ■   ■■■ settingsComponent.css
■   ■   ■■■ shiftOptimization/        # Shift optimization
■   ■       ■■■ shiftOptimizationComponent.vue
■   ■       ■■■ shiftOptimizationAPI.ts
■   ■■■ services/                     # API services
■   ■   ■■■ apiClient.ts              # Axios configuration
■   ■■■ sideBar/                      # Navigation sidebar
■   ■   ■■■ sideBarComponent.vue
■   ■■■ router/                       # Route configuration
■   ■   ■■■ index.ts
■   ■   ■■■ routes.ts
■   ■■■ App.vue                      # Root component
■   ■■■ main.ts                      # Application entry
```

```
■■■ package.json
■■■ tsconfig.json
■■■ vite.config.ts
```

Backend Structure

```
Backend/workForceApplication/
■■■ src/main/java/com/boostphysioclinic/workforceapplication/
■   ■■■ client/                     # External API clients
■   ■   ■■■ RestClientConfig.java
■   ■■■ config/                     # Configuration
■   ■   ■■■ CorsConfig.java
■   ■   ■■■ SecurityConfig.java
■   ■■■ controller/                 # REST Controllers
■   ■   ■■■ AIModelController.java
■   ■   ■■■ DashboardController.java
■   ■   ■■■ PredictionController.java
■   ■   ■■■ SettingsController.java
■   ■   ■■■ ...
■   ■■■ dto/                        # Data Transfer Objects
■   ■   ■■■ entity/                  # Entity DTOs
■   ■   ■   ■■■ AIModel.java
■   ■   ■   ■■■ Employee.java
■   ■   ■   ■■■ Settings.java
■   ■   ■   ■■■ PredictionResult.java
■   ■   ■   ■■■ ...
■   ■   ■■■ PredictionResponse.java
■   ■   ■■■ PredictionResultDTO.java
■   ■   ■■■ PredictionListDeserializer.java
■   ■■■ Repository/                   # JPA Repositories
■   ■   ■■■ AIModelRepository.java
■   ■   ■■■ EmployeeRepository.java
■   ■   ■■■ SettingsRepository.java
■   ■   ■■■ PredictionResultRepository.java
■   ■   ■■■ ...
■   ■■■ security/                   # Security
```

```
■   ■   ■■■ JwtAuthenticationFilter.java
■   ■   ■■■ JwtService.java
■   ■■■ service/                      # Business Logic
■   ■   ■■■ AIModelService.java
■   ■   ■■■ CLPredictionService.java
■   ■   ■■■ PdfGenerationService.java
■   ■   ■■■ PredictionService.java
■   ■   ■■■ ...
■   ■■■ WorkForceApplication.java    # Main application
■■■ src/main/resources/
■   ■■■ application.properties       # Configuration
■   ■■■ schema.sql                   # Database schema
■■■ build.gradle                     # Gradle dependencies
```

Python ML Service Structure

```
workforce-forecasting-python/
■■■ dataset/                        # Data storage
■■■ evaluation/                     # Model evaluation
■   ■■■ evaluation.py
■■■ models/                         # ML Models
■   ■■■ linear_regression_model.py
■   ■■■ random_forest_model.py
■   ■■■ xgboost_model.py
■   ■■■ lstm_model.py
■■■ prediction/                     # Prediction logic
■   ■■■ __init__.py
■■■ preprocessing/                  # Data preprocessing
■   ■■■ preprocessing.py
■   ■■■ lstm_preprocessing.py
■■■ serviceFast/                    # FastAPI service
■   ■■■ __init__.py
■   ■■■ main.py
■■■ training/                       # Model training
■   ■■■ model_saver.py
■   ■■■ model_selector.py
■   ■■■ model_trainer.py
■   ■■■ prediction_result_saver.py
■   ■■■ training_service.py
■■■ visualization/                  # Visualization
■   ■■■ performance_comparison.py
■■■ saved_models/                   # Trained models storage
■■■ results/                        # Prediction results
■■■ uploads/                        # File uploads
■■■ main.py                         # Entry point
■■■ requirements.txt                 # Python dependencies
```

```
■■■ serviceFast/                     # FastAPI application
```

_

Database Design

Database Schema Overview

The database follows a normalized relational schema designed for data integrity, query performance, and scalability. The schema is organized into logical domains: authentication, workforce management, machine learning, and system configuration.

Entity-Relationship Diagram

```
■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■
■    users      ■       ■  ai_models        ■       ■prediction_runs  ■
■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■
■ id (PK)       ■       ■ id (PK)           ■       ■ id (PK)         ■
■ username      ■       ■ model_name        ■       ■ model_name      ■
■ password      ■       ■ algorithm         ■       ■ total_records   ■
■ role          ■       ■ version           ■       ■ avg_prediction  ■
■ email         ■       ■ rmse              ■       ■ max_prediction  ■
■ created_at    ■       ■ mae               ■       ■ min_prediction  ■
■■■■■■■■■■■■■■■■■■■       ■ mape              ■       ■ created_at      ■
                       ■ r_squared         ■       ■■■■■■■■■■■■■■■■■■■■■
                       ■ training_time     ■                ■
                       ■ status            ■                ■
                       ■ created_at        ■                ■
                       ■ trained_at        ■                ■
                       ■■■■■■■■■■■■■■■■■■■■■                    ■
                                                   ■
                                                   ■ 1
                                                   ■
                                                   ■ N
■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■■       ■■■■■■■■■■■■■■■■■■■■■■
■   settings    ■       ■ prediction_       ■■■■■■■■■■prediction_results■
■■■■■■■■■■■■■■■■■■■       ■ results           ■       ■■■■■■■■■■■■■■■■■■■■■
■ id (PK)       ■       ■■■■■■■■■■■■■■■■■■■■■■■       ■ id (PK)         ■
■ full_name     ■       ■ id (PK)           ■       ■ run_id (FK)     ■
■ email         ■       ■ run_id (FK)       ■       ■ attendance_date ■
■ role          ■       ■ attendance_date   ■       ■ department      ■
■ department    ■       ■ department        ■       ■ actual_demand   ■
■ dark_mode     ■       ■ actual_demand     ■       ■ predicted_demand■
■ compact_view  ■       ■ predicted_demand  ■       ■■■■■■■■■■■■■■■■■■■■
```

```
■ animations   ■       ■■■■■■■■■■■■■■■■■■■■■
■ critical_... ■
■ shift_rec... ■       ■■■■■■■■■■■■■■■■■■■■■■
■ system_mon.. ■       ■  employees       ■
■ email_digest ■       ■■■■■■■■■■■■■■■■■■■■■■
■ model_name   ■       ■ id (PK)          ■
■ refresh_int. ■       ■ employee_id      ■
■ api_url      ■       ■ name             ■
■ ml_url       ■       ■ department       ■
■ data_retent. ■       ■ role             ■
■ active_model ■       ■ email            ■
■ model_version■       ■ hire_date        ■
■ training_freq■       ■ status           ■
■ conf_thresh  ■       ■■■■■■■■■■■■■■■■■■■■■■
■ auto_retrain ■
■ monitoring   ■
■ feature_imp. ■
■ created_at   ■
■ updated_at   ■
■■■■■■■■■■■■■■■■■
```

Detailed Table Definitions

Users Table

```
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL COMMENT 'BCrypt hashed password',
    role VARCHAR(50) NOT NULL COMMENT 'ADMIN, MANAGER, VIEWER',
    email VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL,
    is_active BOOLEAN DEFAULT TRUE,
    INDEX idx_username (username),
    INDEX idx_role (role)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Indexes:

- `idx_username`: For login queries
- `idx_role`: For role-based filtering

Constraints:

- `username` must be unique
- `role` must be one of: ADMIN, MANAGER, VIEWER
- `password` stored as BCrypt hash

AI Models Table

```
CREATE TABLE ai_models (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    model_name VARCHAR(255) NOT NULL,
    algorithm VARCHAR(100) NOT NULL COMMENT 'LINEAR_REGRESSION, RANDOM_FOREST, XGBOOST, LSTM',
    version VARCHAR(50),
    rmse DOUBLE COMMENT 'Root Mean Square Error',
    mae DOUBLE COMMENT 'Mean Absolute Error',
    mape DOUBLE COMMENT 'Mean Absolute Percentage Error',
    r_squared DOUBLE COMMENT 'R-Squared Score',
    training_time BIGINT COMMENT 'Training time in milliseconds',
    status VARCHAR(50) DEFAULT 'PENDING' COMMENT 'PENDING, TRAINING, COMPLETED, FAILED',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    trained_at TIMESTAMP NULL,
    model_path VARCHAR(500) COMMENT 'Path to saved model file',
    hyperparameters JSON COMMENT 'Model hyperparameters as JSON',
    feature_importance JSON COMMENT 'Feature importance scores',
    INDEX idx_algorithm (algorithm),
    INDEX idx_status (status),
    INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Indexes:

- `idx_algorithm`: For filtering by algorithm type
- `idx_status`: For tracking training status
- `idx_created_at`: For chronological queries

JSON Fields:

- `hyperparameters`: Stores model-specific parameters
- `feature_importance`: Stores feature importance scores

Settings Table

```
CREATE TABLE settings (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,

    -- Profile Settings
    full_name VARCHAR(255),
    email VARCHAR(255),
    role VARCHAR(100),
    department VARCHAR(100),

    -- Appearance Settings
    dark_mode BOOLEAN DEFAULT FALSE,
    compact_view BOOLEAN DEFAULT FALSE,
    animations BOOLEAN DEFAULT TRUE,

    -- Notification Settings
    critical_alerts BOOLEAN DEFAULT TRUE,
    shift_recommendations BOOLEAN DEFAULT TRUE,
    system_monitoring BOOLEAN DEFAULT TRUE,
    email_digest BOOLEAN DEFAULT FALSE,

    -- Config Settings
    model_name VARCHAR(255),
    refresh_interval VARCHAR(50) DEFAULT '5m',
    api_url VARCHAR(500),
    ml_url VARCHAR(500),
    data_retention INT DEFAULT 90,

    -- AI Model Settings
    active_model VARCHAR(255),
    model_version VARCHAR(50),
```

```
    training_frequency VARCHAR(50) DEFAULT 'weekly',
    confidence_threshold INT DEFAULT 80,
    auto_retrain BOOLEAN DEFAULT FALSE,
    monitoring_enabled BOOLEAN DEFAULT TRUE,
    feature_importance BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_updated_at (updated_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Triggers:

- `updated_at` automatically updates on any row modification

Prediction Runs Table

```sql
CREATE TABLE prediction_runs (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    model_name VARCHAR(255) NOT NULL,
    model_id BIGINT,
    total_records INT,
    average_prediction DOUBLE,
    maximum_prediction DOUBLE,
    minimum_prediction DOUBLE,
    confidence_level DOUBLE DEFAULT 0.95,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP NULL,
    status VARCHAR(50) DEFAULT 'RUNNING' COMMENT 'RUNNING, COMPLETED, FAILED',
    error_message TEXT,
    metadata JSON COMMENT 'Additional run metadata',
    FOREIGN KEY (model_id) REFERENCES ai_models(id) ON DELETE SET NULL,
    INDEX idx_created_at (created_at),
    INDEX idx_status (status),
    INDEX idx_model_id (model_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Foreign Key:

• `model_id` references `ai_models(id)` with SET NULL on delete

Prediction Results Table```sql

CREATE TABLE prediction_results (

id BIGINT PRIMARY KEY AUTO_INCREMENT,

prediction_run_id BIGINT NOT NULL,

attendance_date DATE NOT NULL,

department VARCHAR(255),

actual_demand DOUBLE,

predicted_demand DOUBLE,

confidence_interval_lower DOUBLE,

confidence_interval_upper DOUBLE,

error_margin DOUBLE,

is_anomaly BOOLEAN DEFAULT FALSE,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (prediction_run_id) REFERENCES prediction_runs(id) ON DELETE CASCADE,

INDEX idx_run_id (prediction_run_id),

INDEX idx_attendance_date (attendance_date),

INDEX idx_department (department),

INDEX idx_is_anomaly (is_anomaly)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

```
**Foreign Key:**
- `prediction_run_id` references `prediction_runs(id)` with CASCADE delete

**Indexes:**
- `idx_run_id`: For fetching results by run
- `idx_attendance_date`: For time-series queries
- `idx_department`: For department-specific queries
- `idx_is_anomaly`: For anomaly detection queries

#### Employees Table
```

CREATE TABLE employees (

id BIGINT PRIMARY KEY AUTO_INCREMENT,

employee_id VARCHAR(50) UNIQUE NOT NULL,

name VARCHAR(255) NOT NULL,

department VARCHAR(100),

role VARCHAR(100),

email VARCHAR(255),

hire_date DATE,

status VARCHAR(50) DEFAULT 'ACTIVE' COMMENT 'ACTIVE, INACTIVE, ON_LEAVE',

skills JSON COMMENT 'Employee skills as JSON array',

availability JSON COMMENT 'Availability schedule',

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

INDEX idx_employee_id (employee_id),

INDEX idx_department (department),

INDEX idx_status (status)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

```
**JSON Fields:**
- `skills`: Array of employee skills
- `availability`: Weekly availability schedule

### Database Optimization Strategies

#### Indexing Strategy

1. **Primary Indexes**: All primary keys are auto-increment BIGINT
2. **Foreign Key Indexes**: All foreign keys have indexes for join performance
3. **Query Indexes**: Indexes on frequently queried columns
4. **Composite Indexes**: For multi-column queries (future)

#### Query Optimization

1. **Connection Pooling**: HikariCP with optimal pool size
2. **Query Caching**: Second-level Hibernate cache (future)
3. **Read Replicas**: For read-heavy workloads (future)
4. **Partitioning**: By date for large tables (future)

#### Data Retention Policy

- **Prediction Results**: Retained for 90 days (configurable)
- **Prediction Runs**: Retained for 1 year
- **AI Models**: All versions retained
- **Settings**: Single active record per user

### Backup Strategy
```

```
#### Backup Types

1. **Full Backups**: Daily at 2:00 AM
2. **Incremental Backups**: Every 4 hours
3. **Binary Logs**: Continuous for point-in-time recovery

#### Retention Policy

- Daily backups: 7 days
- Weekly backups: 4 weeks
- Monthly backups: 12 months

---

## API Specifications

### API Overview

The REST API follows RESTful principles with resource-based URLs, standard HTTP methods, and appropriate sta

### Base URL

- **Development**: `http://localhost:5233/api`
- **Production**: `https://api.workforce-system.com/api`

### Authentication

All endpoints except `/api/auth/login` require JWT authentication in the Authorization header:
```

Authorization: Bearer <jwt_token>

```
### Response Format

#### Success Response
```

{

"success": true,

"data": { ... },

"message": "Operation successful",

"timestamp": "2026-08-19T10:30:00Z"

}

```
#### Error Response
```

{

"success": false,

"error": {

"code": "ERROR_CODE",

"message": "Error description",

"details": { ... }

},

"timestamp": "2026-08-19T10:30:00Z"

}

```
### HTTP Status Codes

| Code | Description |
|------|-------------|
| 200 | OK - Request successful |
| 201 | Created - Resource created successfully |
| 204 | No Content - Successful with no return data |
| 400 | Bad Request - Invalid request parameters |
| 401 | Unauthorized - Authentication required/failed |
| 403 | Forbidden - Insufficient permissions |
| 404 | Not Found - Resource not found |
| 409 | Conflict - Resource conflict |
| 500 | Internal Server Error - Server error |
| 503 | Service Unavailable - Service temporarily down |

---

### Authentication Endpoints

#### POST /api/auth/login

Authenticates a user and returns a JWT token.

**Request Headers:**
```

Content-Type: application/json

```
**Request Body:**
```


{

"username": "admin",

"password": "admin123"

}


```
**Response (200 OK):**
```


{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"tokenType": "Bearer",

"expiresIn": 86400000,

"user": {

"id": 1,

"username": "admin",

"role": "ADMIN",

"email": "admin@workforce.com"

}

},

"message": "Login successful"

}


```
**Error Response (401 Unauthorized):**
```

{

"success": false,

"error": {

"code": "INVALID_CREDENTIALS",

"message": "Invalid username or password"

}

}

```
#### POST /api/auth/logout

Invalidates the current JWT token (Not Implemented).

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"message": "Logout successful"

}

```
#### POST /api/auth/refresh

Refreshes an expired JWT token (Not Implemented).

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresIn": 86400000

}

}

```
---

### Dashboard Endpoints

#### GET /api/dashboard/predictions

Fetches the latest prediction results for the dashboard.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| limit | integer | No | Maximum number of results (default: 50) |
| offset | integer | No | Offset for pagination (default: 0) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"model": "RandomForest_v1",

"total_records": 365,

"average_prediction": 45.5,

"maximum_prediction": 78.2,

"minimum_prediction": 12.3,

"results": [

{

"attendanceDate": "2026-08-19",

"department": "Operations",

"actualDemand": 42.0,

"predictedDemand": 44.5

}

]

}

}

```
#### GET /api/dashboard/metrics

Retrieves KPI metrics for the dashboard.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| period | string | No | Time period: today, week, month, year (default: today) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"totalEmployees": 245,

"activeShifts": 18,

"predictedDemand": 156,

"capacityUtilization": 78.5,

"attendanceRate": 92.3,

"productivityScore": 87.2

}

}

```
#### GET /api/dashboard/charts

Fetches chart data for dashboard visualizations.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| chartType | string | Yes | Type of chart: line, bar, pie, radar |
| period | string | No | Time period (default: week) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"labels": ["Mon", "Tue", "Wed", "Thu", "Fri"],

"datasets": [

{

"label": "Actual Demand",

"data": [45, 52, 48, 55, 50],

"backgroundColor": "rgba(59, 130, 246, 0.5)"

},

{

"label": "Predicted Demand",

"data": [47, 50, 50, 53, 48],

"backgroundColor": "rgba(139, 92, 246, 0.5)"

}

]

}

}

```
---

### AI Models Endpoints

#### GET /api/ai-models

Retrieves all AI models with their performance metrics.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| algorithm | string | No | Filter by algorithm type |
| status | string | No | Filter by status |
| sortBy | string | No | Sort field: created_at, rmse, mae (default: created_at) |
| sortOrder | string | No | Sort order: asc, desc (default: desc) |

**Response (200 OK):**
```

{

"success": true,

"data": [

{

"id": 1,

"modelName": "RandomForest_v1",

"algorithm": "RANDOM_FOREST",

"version": "1.0",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92,

"trainingTime": 12500,

"status": "COMPLETED",

"createdAt": "2026-08-15T10:00:00Z",

"trainedAt": "2026-08-15T10:03:25Z"

}

],

"pagination": {

"total": 10,

"page": 1,

"pageSize": 10

}

}

```
#### GET /api/ai-models/{id}

Retrieves details of a specific AI model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Model ID |

**Response (200 OK):**
```


{

"success": true,

"data": {

"id": 1,

"modelName": "RandomForest_v1",

"algorithm": "RANDOM_FOREST",

"version": "1.0",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92,

"trainingTime": 12500,

"status": "COMPLETED",

"hyperparameters": {

"n_estimators": 100,

"max_depth": 10,

"min_samples_split": 2

},

"featureImportance": {

"attendance_history": 0.35,

"seasonal_factors": 0.25,

"holidays": 0.15,

"weather": 0.10,

"other": 0.15

}

}

}

```
#### POST /api/ai-models/train

Trains a new ML model with the provided dataset.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: multipart/form-data

```
**Request Body (multipart/form-data):**
| Field | Type | Required | Description |
|-------|------|----------|-------------|
| file | file | Yes | Training dataset file (CSV/Excel) |
| algorithm | string | Yes | Algorithm: LINEAR_REGRESSION, RANDOM_FOREST, XGBOOST, LSTM |
| modelName | string | Yes | Name for the model |
| hyperparameters | string | No | JSON string of hyperparameters |

**Response (201 Created):**
```

{

"success": true,

"data": {

"modelId": 11,

"status": "TRAINING",

"message": "Model training started"

}

}

```
#### GET /api/ai-models/compare

Compares performance metrics of multiple models.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelIds | string | Yes | Comma-separated model IDs |

**Response (200 OK):**
```


{

"success": true,

"data": {

"comparison": [

{

"modelId": 1,

"modelName": "RandomForest_v1",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92

},

{

"modelId": 2,

"modelName": "XGBoost_v1",

"rmse": 3.12,

"mae": 2.45,

"mape": 4.89,

"rSquared": 0.94

}

],

"bestModel": {

"modelId": 2,

"modelName": "XGBoost_v1",

"reason": "Lowest RMSE and highest R-squared"

}

}

}

```
#### DELETE /api/ai-models/{id}

Deletes a specific AI model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Model ID |

**Response (200 OK):**
```

{

"success": true,

"message": "Model deleted successfully"

}

```
---

### Prediction Endpoints

#### POST /api/predict

Generates predictions using the specified model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: multipart/form-data

```
**Request Body (multipart/form-data):**
| Field | Type | Required | Description |
|-------|------|----------|-------------|
| file | file | Yes | Input data file (CSV/Excel) |
| modelId | long | No | Model ID to use (default: latest) |
| horizon | integer | No | Prediction horizon in days (default: 7) |

**Response (200 OK):**
```


{

"success": true,

"data": {

"predictionRunId": 25,

"model": "RandomForest_v1",

"totalRecords": 7,

"averagePrediction": 48.5,

"maximumPrediction": 65.2,

"minimumPrediction": 32.1,

"predictions": [

{

"attendanceDate": "2026-08-20",

"department": "Operations",

"predictedDemand": 52.3,

"confidenceInterval": {

"lower": 48.5,

"upper": 56.1

}

}

]

}

}

```
#### GET /api/predict/latest

Retrieves the latest prediction run results.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"id": 25,

"modelName": "RandomForest_v1",

"totalRecords": 7,

"averagePrediction": 48.5,

"maximumPrediction": 65.2,

"minimumPrediction": 32.1,

"createdAt": "2026-08-19T10:30:00Z",

"results": [

{

"attendanceDate": "2026-08-20",

"department": "Operations",

"actualDemand": null,

"predictedDemand": 52.3

}

]

}

}

```
#### GET /api/predict/{runId}

Retrieves a specific prediction run by ID.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| runId | long | Yes | Prediction run ID |

**Response (200 OK):**
```

{

"success": true,

"data": {

"id": 25,

"modelName": "RandomForest_v1",

"totalRecords": 7,

"results": [...]

}

}

```
---

### Settings Endpoints

#### GET /api/settings

Retrieves application settings.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"profile": {

"fullName": "John Doe",

"email": "john.doe@workforce.com",

"role": "ADMIN",

"department": "IT"

},

"appearance": {

"darkMode": false,

"compactView": false,

"animations": true

},

"notifications": {

"criticalAlerts": true,

"shiftRecommendations": true,

"systemMonitoring": true,

"emailDigest": false

},

"config": {

"model": "RandomForest_v1",

"refresh": "5m",

"apiUrl": "http://localhost:5233/api",

"mlUrl": "http://localhost:8000",

"dataRetention": 90

}

}

}

```
#### PUT /api/settings

Updates application settings.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Request Body:**
```


{

"profile": {

"fullName": "John Doe",

"email": "john.doe@workforce.com",

"role": "ADMIN",

"department": "IT"

},

"appearance": {

"darkMode": false,

"compactView": false,

"animations": true

},

"notifications": {

"criticalAlerts": true,

"shiftRecommendations": true,

"systemMonitoring": true,

"emailDigest": false

},

"config": {

"model": "RandomForest_v1",

"refresh": "5m",

"apiUrl": "http://localhost:5233/api",

"mlUrl": "http://localhost:8000",

"dataRetention": 90

}

}

```
**Response (200 OK):**
```

{

"success": true,

"message": "Settings saved successfully"

}

```
---

### Reports Endpoints

#### GET /api/reports/performance

Generates a performance report in PDF format.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelId | long | No | Model ID (default: latest) |
| startDate | string | No | Report start date (ISO format) |
| endDate | string | No | Report end date (ISO format) |

**Response (200 OK):**
- Content-Type: application/pdf
- Content-Disposition: attachment; filename="performance_report.pdf"

#### GET /api/reports/comparison

Generates a model comparison report in PDF format.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelIds | string | Yes | Comma-separated model IDs |

**Response (200 OK):**
- Content-Type: application/pdf
- Content-Disposition: attachment; filename="comparison_report.pdf"

#### GET /api/reports/forecast

Generates a forecast report with predictions.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| runId | long | Yes | Prediction run ID |
| format | string | No | Format: pdf, excel, csv (default: pdf) |

**Response (200 OK):**
- Content-Type: application/pdf (or application/vnd.ms-excel, text/csv)
- Content-Disposition: attachment; filename="forecast_report.pdf"

---

### Employee Endpoints

#### GET /api/employees

Retrieves all employees.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| department | string | No | Filter by department |
| status | string | No | Filter by status |
| page | integer | No | Page number (default: 1) |
| size | integer | No | Page size (default: 20) |

**Response (200 OK):**
```


{

"success": true,

"data": [

{

"id": 1,

"employeeId": "EMP001",

"name": "John Doe",

"department": "Operations",

"role": "Manager",

"email": "john.doe@workforce.com",

"hireDate": "2020-01-15",

"status": "ACTIVE"

}

],

"pagination": {

"total": 245,

"page": 1,

"pageSize": 20,

"totalPages": 13

}

}

```
#### POST /api/employees

Creates a new employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Request Body:**
```

{

"employeeId": "EMP246",

"name": "Jane Smith",

"department": "Operations",

"role": "Analyst",

"email": "jane.smith@workforce.com",

"hireDate": "2026-08-19",

"status": "ACTIVE"

}

```
**Response (201 Created):**
```

{

"success": true,

"data": {

"id": 246,

"employeeId": "EMP246",

"name": "Jane Smith",

"department": "Operations",

"role": "Analyst",

"email": "jane.smith@workforce.com",

"hireDate": "2026-08-19",

"status": "ACTIVE"

}

}

```
#### PUT /api/employees/{id}

Updates an existing employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Employee ID |

**Request Body:**
```

{

"name": "Jane Smith",

"department": "Analytics",

"role": "Senior Analyst",

"status": "ACTIVE"

}

```
**Response (200 OK):**
```


{

"success": true,

"data": {

"id": 246,

"name": "Jane Smith",

"department": "Analytics",

"role": "Senior Analyst",

"status": "ACTIVE"

}

}


```
#### DELETE /api/employees/{id}

Deletes an employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Employee ID |

**Response (200 OK):**
```


{

"success": true,

"message": "Employee deleted successfully"

}

```
---

### Monitoring Endpoints

#### GET /api/monitoring/health

Retrieves system health status.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"status": "HEALTHY",

"components": {

"database": "UP",

"pythonService": "UP",

"diskSpace": "OK"

},

"uptime": 86400,

"version": "1.0.0"

}

}

```
#### GET /api/monitoring/metrics

Retrieves system performance metrics.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| period | string | No | Time period: hour, day, week (default: hour) |

**Response (200 OK):**
```


{

"success": true,

"data": {

"cpuUsage": 45.2,

"memoryUsage": 68.5,

"diskUsage": 52.3,

"requestCount": 1250,

"averageResponseTime": 245,

"errorRate": 0.02

}

}

```
---

### Error Codes

| Code | Description |
|------|-------------|
| INVALID_CREDENTIALS | Invalid username or password |
| TOKEN_EXPIRED | JWT token has expired |
| TOKEN_INVALID | JWT token is invalid |
| INSUFFICIENT_PERMISSIONS | User lacks required permissions |
| RESOURCE_NOT_FOUND | Requested resource does not exist |
| VALIDATION_ERROR | Request validation failed |
| DUPLICATE_RESOURCE | Resource already exists |
| EXTERNAL_SERVICE_ERROR | External service error |
| DATABASE_ERROR | Database operation failed |
| INTERNAL_ERROR | Unexpected server error |

---

## Machine Learning Implementation

### Overview

The Machine Learning module implements a comprehensive forecasting system with multiple algorithms, automate

### Model Architecture
```

■■■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■ ■

■ ■ Preprocess- ■ ■ Feature ■ ■ Model ■ ■

■ ■ ing ■ ■ Selection ■ ■ Evaluation■ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■ ■

■ ■ Train-Test ■ ■ Hyperparam ■ ■ Model ■ ■

■ ■ Split ■ ■ Optimization■ ■ Selection■ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■ ■

■ ■

■■■■■■■■■■■■■■■■■■

from sklearn.pipeline import Pipeline

model = Pipeline([

('scaler', StandardScaler()),

('regressor', LinearRegression())

])

```
**Hyperparameters**:
- `fit_intercept`: Whether to calculate the intercept (default: True)
- `copy_X`: Whether to copy X before fitting (default: True)
- `n_jobs`: Number of jobs for parallel computation (default: None)

**Use Cases**:
- Baseline model for comparison
- Quick prototyping
- Interpretable results
- Small datasets with linear relationships

**Pros**:
- Fast training and prediction
- Highly interpretable
- Low computational cost
- Works well with linear relationships

**Cons**:
- Cannot capture non-linear patterns
- Sensitive to outliers
- Assumes independence of features
- Limited predictive power for complex patterns

---

#### 2. Random Forest

**Description**: An ensemble learning method that constructs multiple decision trees during training and outputs the mean pr
```

```
**Mathematical Formulation**:
```

■ = (1/N) Σ f■(x)

```
Where:
- ■ = predicted value
- N = number of trees
- f■(x) = prediction of i-th tree

**Implementation Details**:
```

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV

model = RandomForestRegressor(

n_estimators=100,

max_depth=10,

min_samples_split=2,

random_state=42

)
```

```
**Hyperparameters**:
- `n_estimators`: Number of trees (default: 100, range: 10-500)
- `max_depth`: Maximum tree depth (default: None, range: 5-30)
- `min_samples_split`: Minimum samples to split (default: 2, range: 2-20)
- `min_samples_leaf`: Minimum samples per leaf (default: 1, range: 1-10)
- `max_features`: Features considered for split (default: sqrt)
- `bootstrap`: Whether to use bootstrap samples (default: True)

**Feature Importance**:
Random Forest provides built-in feature importance scores:
```

```
importances = model.feature_importances_

feature_names = X.columns

for feature, importance in zip(feature_names, importances):

print(f"{feature}: {importance:.4f}")
```

```
**Use Cases**:
- Non-linear relationships
- Feature importance analysis
- Robust to overfitting
- Medium to large datasets

**Pros**:
- Handles non-linear relationships
- Robust to overfitting
- Provides feature importance
- Handles missing values well
- Parallelizable

**Cons**:
- Slower training than linear models
- Less interpretable than linear regression
- Large memory footprint
- Prediction latency higher than simple models

---

#### 3. XGBoost (Extreme Gradient Boosting)

**Description**: An optimized gradient boosting library that implements a gradient boosting framework. It's

**Mathematical Formulation**:
```

■■ = Σ f■(x■)

```
Where:
- ■■ = prediction for i-th instance
- f■ = k-th weak learner (decision tree)
- Objective function: L(ϕ) = Σ l(y■, ■■) + Σ Ω(f■)

**Implementation Details**:
```

import xgboost as xgb

from xgboost import XGBRegressor

model = XGBRegressor(

n_estimators=100,

max_depth=6,

learning_rate=0.1,

subsample=0.8,

colsample_bytree=0.8,

random_state=42

)

```
**Hyperparameters**:
- `n_estimators`: Number of boosting rounds (default: 100, range: 50-500)
- `max_depth`: Maximum tree depth (default: 6, range: 3-15)
- `learning_rate`: Shrinkage factor (default: 0.1, range: 0.01-0.3)
- `subsample`: Row sampling rate (default: 1.0, range: 0.5-1.0)
- `colsample_bytree`: Feature sampling rate (default: 1.0, range: 0.5-1.0)
- `gamma`: Minimum loss reduction (default: 0, range: 0-10)
- `reg_alpha`: L1 regularization (default: 0, range: 0-10)
- `reg_lambda`: L2 regularization (default: 1, range: 0-10)

**Early Stopping**:
```

model.fit(

X_train, y_train,

eval_set=[(X_val, y_val)],

early_stopping_rounds=10,

verbose=False

)

```
**Use Cases**:
- High-accuracy requirements
- Large datasets
- Complex non-linear patterns
- Competitions and production systems

**Pros**:
- State-of-the-art accuracy
- Handles missing values
- Regularization built-in
- Parallel and distributed computing
- Cross-validation support

**Cons**:
- More hyperparameters to tune
- Longer training time
- Can overfit if not regularized
- Less interpretable than simpler models

---

#### 4. LSTM (Long Short-Term Memory)

**Description**: A type of recurrent neural network (RNN) capable of learning long-term dependencies, partic

**Architecture**:
```

Input Layer → LSTM Layer(s) dropout → Dense Layer → Output

```
**Mathematical Formulation**:

LSTM cell equations:
```

f■ = σ(Wf · [h■■■, x■] + bf)

i■ = σ(Wi · [h■■■, x■] + bi)

C■■ = tanh(WC · [h■■■, x■] + bC)

C■ = f■ *C*■■■ + *i*■ C■■

o■ = σ(Wo · [h■■■, x■] + bo)

h■ = o■ * tanh(C■)

```
Where:
- f■ = forget gate
- i■ = input gate
- C■■ = candidate cell state
- C■ = cell state
- o■ = output gate
- h■ = hidden state

**Implementation Details**:
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([

LSTM(50, return_sequences=True, input_shape=(timesteps, features)),

Dropout(0.2),

LSTM(50, return_sequences=False),

Dropout(0.2),

Dense(25),

Dense(1)

])

model.compile(optimizer='adam', loss='mse')
```

```
**Hyperparameters**:
- `units`: Number of LSTM units (default: 50, range: 32-128)
- `return_sequences`: Whether to return sequences (default: True for stacked LSTM)
- `dropout`: Dropout rate (default: 0.2, range: 0.1-0.5)
- `recurrent_dropout`: Recurrent dropout rate (default: 0.0)
- `activation`: Activation function (default: tanh)
- `recurrent_activation`: Recurrent activation (default: sigmoid)
- `optimizer`: Optimizer (default: adam)
- `learning_rate`: Learning rate (default: 0.001)
- `batch_size`: Batch size (default: 32, range: 16-128)
- `epochs`: Number of epochs (default: 100, range: 50-500)

**Data Preparation for LSTM**:
```

```
def create_sequences(data, sequence_length):

sequences = []

targets = []

for i in range(len(data) - sequence_length):

sequences.append(data[i:i + sequence_length])

targets.append(data[i + sequence_length])

return np.array(sequences), np.array(targets)

sequence_length = 30 # 30 days lookback

X_seq, y_seq = create_sequences(scaled_data, sequence_length)
```

```
**Use Cases**:
- Time-series forecasting
- Sequential data modeling
- Long-term dependency learning
- Complex temporal patterns

**Pros**:
- Captures temporal dependencies
- Handles complex patterns
- State-of-the-art for sequences
- Flexible architecture

**Cons**:
- Requires large datasets
- Long training time
- Computationally expensive
- Hard to interpret
- Requires careful tuning

---

### Feature Engineering

#### Time-Based Features
```

```
def extract_time_features(df):

df['year'] = df['date'].dt.year

df['month'] = df['date'].dt.month
```

df['day'] = df['date'].dt.day

df['day_of_week'] = df['date'].dt.dayofweek

df['day_of_year'] = df['date'].dt.dayofyear

df['week_of_year'] = df['date'].dt.isocalendar().week

df['quarter'] = df['date'].dt.quarter

df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)

df['is_month_start'] = df['date'].dt.is_month_start.astype(int)

df['is_month_end'] = df['date'].dt.is_month_end.astype(int)

return df

```
#### Lag Features
```

def create_lag_features(df, target_column, lags=[1, 7, 30]):

for lag in lags:

df[f'{target_column}_lag_{lag}'] = df[target_column].shift(lag)

return df

```
#### Rolling Statistics
```

def create_rolling_features(df, target_column, windows=[7, 14, 30]):

for window in windows:

df[f'{target_column}_rolling_mean_{window}'] = df[target_column].rolling(window).mean()

df[f'{target_column}_rolling_std_{window}'] = df[target_column].rolling(window).std()

df[f'{target_column}_rolling_min_{window}'] = df[target_column].rolling(window).min()

df[f'{target_column}_rolling_max_{window}'] = df[target_column].rolling(window).max()

return df

```
#### Seasonal Features
```

def create_seasonal_features(df):

import numpy as np

df['sin_month'] = np.sin(2 *np.pi* df['month'] / 12)

df['cos_month'] = np.cos(2 *np.pi* df['month'] / 12)

df['sin_day'] = np.sin(2 *np.pi* df['day'] / 31)

df['cos_day'] = np.cos(2 *np.pi* df['day'] / 31)

df['sin_day_of_week'] = np.sin(2 *np.pi* df['day_of_week'] / 7)

df['cos_day_of_week'] = np.cos(2 *np.pi* df['day_of_week'] / 7)

return df

```
#### External Features

- **Holiday Indicators**: Mark public holidays
- **Weather Data**: Temperature, precipitation
- **Economic Indicators**: GDP, unemployment rate
- **Company Events**: Product launches, promotions

---

### Training Pipeline

#### Step 1: Data Loading
```

import pandas as pd

def load_data(file_path):

"""Load data from CSV or Excel file."""

if file_path.endswith('.csv'):

df = pd.read_csv(file_path)

elif file_path.endswith(('.xlsx', '.xls')):

df = pd.read_excel(file_path)

else:

raise ValueError("Unsupported file format")

Convert date column to datetime

if 'date' in df.columns:

df['date'] = pd.to_datetime(df['date'])

return df

```
#### Step 2: Data Preprocessing
```

from sklearn.preprocessing import StandardScaler, LabelEncoder

def preprocess_data(df):

"""Clean and preprocess data."""

Handle missing values

df = df.fillna(method='ffill').fillna(method='bfill')

Encode categorical variables

categorical_cols = df.select_dtypes(include=['object']).columns

for col in categorical_cols:

le = LabelEncoder()

df[col] = le.fit_transform(df[col])

Remove outliers using IQR method

numeric_cols = df.select_dtypes(include=[np.number]).columns

for col in numeric_cols:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

```
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df[col] = df[col].clip(lower_bound, upper_bound)

return df
```

```
#### Step 3: Feature Engineering
```

```
def engineer_features(df):

"""Create features for ML models."""

df = extract_time_features(df)

df = create_lag_features(df, 'demand', lags=[1, 7, 30])

df = create_rolling_features(df, 'demand', windows=[7, 14, 30])

df = create_seasonal_features(df)

# Drop rows with NaN from lag features

df = df.dropna()

return df
```

```
#### Step 4: Train-Test Split
```

```
from sklearn.model_selection import train_test_split

def split_data(df, target_column, test_size=0.2):

"""Split data into train and test sets."""

X = df.drop(columns=[target_column])

y = df[target_column]
```

```
# Time-series split (no shuffling)
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=test_size, shuffle=False
)
return X_train, X_test, y_train, y_test
```

```
#### Step 5: Model Training
```

```
def train_model(X_train, y_train, algorithm='random_forest'):
"""Train a model with the specified algorithm."""
if algorithm == 'linear_regression':
from sklearn.linear_model import LinearRegression
model = LinearRegression()
elif algorithm == 'random_forest':
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
elif algorithm == 'xgboost':
from xgboost import XGBRegressor
model = XGBRegressor(n_estimators=100, random_state=42)
elif algorithm == 'lstm':
# LSTM requires special preprocessing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model = Sequential([
LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
LSTM(50),
```

```
Dense(1)
])
model.compile(optimizer='adam', loss='mse')
else:
raise ValueError(f"Unknown algorithm: {algorithm}")

model.fit(X_train, y_train)
return model
```

```
#### Step 6: Model Evaluation
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

def evaluate_model(model, X_test, y_test):
"""Evaluate model performance."""
y_pred = model.predict(X_test)

metrics = {
'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
'mae': mean_absolute_error(y_test, y_pred),
'mape': np.mean(np.abs((y_test - y_pred) / y_test)) * 100,
'r_squared': r2_score(y_test, y_pred)
}

return metrics
```

```
---

### Hyperparameter Optimization

#### Grid Search
```

```
from sklearn.model_selection import GridSearchCV

def grid_search_optimization(X_train, y_train, algorithm='random_forest'):

"""Perform grid search for hyperparameter optimization."""

if algorithm == 'random_forest':

param_grid = {

'n_estimators': [50, 100, 200],

'max_depth': [10, 20, None],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4]

}

model = RandomForestRegressor(random_state=42)

elif algorithm == 'xgboost':

param_grid = {

'n_estimators': [50, 100, 200],

'max_depth': [3, 6, 9],

'learning_rate': [0.01, 0.1, 0.2],

'subsample': [0.8, 0.9, 1.0]

}

model = XGBRegressor(random_state=42)

grid_search = GridSearchCV(
```

```
model, param_grid, cv=5,
scoring='neg_mean_squared_error',
n_jobs=-1, verbose=1
)

grid_search.fit(X_train, y_train)

return grid_search.best_estimator_, grid_search.best_params_
```

```
#### Random Search
```

```
from sklearn.model_selection import RandomizedSearchCV

def random_search_optimization(X_train, y_train, algorithm='random_forest', n_iter=50):
"""Perform random search for hyperparameter optimization."""

if algorithm == 'random_forest':
param_distributions = {
'n_estimators': [50, 100, 200, 500],
'max_depth': [10, 20, 30, None],
'min_samples_split': [2, 5, 10, 20],
'min_samples_leaf': [1, 2, 4, 8]
}
model = RandomForestRegressor(random_state=42)

random_search = RandomizedSearchCV(
model, param_distributions, n_iter=n_iter, cv=5,
scoring='neg_mean_squared_error',
n_jobs=-1, verbose=1, random_state=42
)
```

random_search.fit(X_train, y_train)

return random_search.best_estimator_, random_search.best_params_

```
---

### Model Evaluation Metrics

#### RMSE (Root Mean Square Error)
```

def calculate_rmse(y_true, y_pred):

"""Calculate Root Mean Square Error."""

return np.sqrt(mean_squared_error(y_true, y_pred))

```
**Interpretation**: Lower is better. Measures the standard deviation of prediction errors.

#### MAE (Mean Absolute Error)
```

def calculate_mae(y_true, y_pred):

"""Calculate Mean Absolute Error."""

return mean_absolute_error(y_true, y_pred)

```
**Interpretation**: Lower is better. Average absolute difference between predicted and actual values.

#### MAPE (Mean Absolute Percentage Error)
```

def calculate_mape(y_true, y_pred):

"""Calculate Mean Absolute Percentage Error."""

return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

```
**Interpretation**: Lower is better. Percentage-based error metric, useful for relative comparison.

#### R² (R-Squared)
```

def calculate_r2(y_true, y_pred):

"""Calculate R-Squared score."""

return r2_score(y_true, y_pred)

```
**Interpretation**: Range from 0 to 1. Higher is better. Proportion of variance explained by the model.

---

### Model Persistence

#### Save Model
```

import joblib

import pickle

def save_model(model, file_path, algorithm):

"""Save trained model to disk."""

if algorithm == 'lstm':

model.save(file_path + '.h5')

else:

joblib.dump(model, file_path + '.joblib')

```
#### Load Model
```

def load_model(file_path, algorithm):

"""Load trained model from disk."""

if algorithm == 'lstm':

from tensorflow.keras.models import load_model

return load_model(file_path + '.h5')

else:

return joblib.load(file_path + '.joblib')

```
---

### Prediction Service

#### Generate Prediction
```

def generate_prediction(model, input_data, algorithm):

"""Generate prediction using trained model."""

if algorithm == 'lstm':

LSTM requires 3D input

input_data = input_data.reshape(1, input_data.shape[0], input_data.shape[1])

prediction = model.predict(input_data)

return prediction[0] if algorithm != 'lstm' else prediction[0][0]

```
#### Batch Prediction
```

def batch_predict(model, input_data, algorithm):

"""Generate predictions for multiple inputs."""

if algorithm == 'lstm':

input_data = input_data.reshape(input_data.shape[0], input_data.shape[1], 1)

predictions = model.predict(input_data)

return predictions.flatten() if algorithm != 'lstm' else predictions.flatten()

```
---

## Security Architecture

### Security Overview

The security architecture implements defense-in-depth principles with multiple layers of protection includin

### Security Layers
```

■■■

■ ■ • Output Encoding ■ ■

■ ■ • API Key Management ■ ■

■ ■■■

```
### Authentication

#### JWT (JSON Web Token) Implementation

The system uses JWT tokens for stateless authentication. Tokens are generated upon successful login and valid

**Token Structure**:
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

```
**Header**:
```

{

"alg": "HS256",

"typ": "JWT"

}

```
**Payload**:
```

{

"sub": "1234567890",

"username": "admin",

"role": "ADMIN",

"iat": 1516239022,

"exp": 1516325422

}

```
**Signature**:
```

HMACSHA256(

base64UrlEncode(header) + "." + base64UrlEncode(payload),

secret

)

```
#### JWT Service Implementation
```

@Service

public class JwtService {

@Value("${jwt.secret}")

private String jwtSecret;

@Value("${jwt.expiration}")

private long jwtExpiration;

public String generateToken(UserDetails userDetails) {

return generateToken(new HashMap<>(), userDetails);

}

public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {

return Jwts.builder()

.setClaims(extraClaims)

.setSubject(userDetails.getUsername())

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + jwtExpiration))

.signWith(getSignInKey(), SignatureAlgorithm.HS256)

.compact();

}

private Key getSignInKey() {

```
byte[] keyBytes = Decoders.BASE64.decode(jwtSecret);
return Keys.hmacShaKeyFor(keyBytes);
}

public String extractUsername(String token) {
return extractClaim(token, Claims::getSubject);
}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
final Claims claims = extractAllClaims(token);
return claimsResolver.apply(claims);
}

private Claims extractAllClaims(String token) {
return Jwts.parserBuilder()
.setSigningKey(getSignInKey())
.build()
.parseClaimsJws(token)
.getBody();
}

public boolean isTokenValid(String token, UserDetails userDetails) {
final String username = extractUsername(token);
return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
}

private boolean isTokenExpired(String token) {
return extractExpiration(token).before(new Date());
}
```

```
private Date extractExpiration(String token) {
return extractClaim(token, Claims::getExpiration);
}
}
```

```
#### JWT Authentication Filter
```

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtService jwtService;
private final UserDetailsService userDetailsService;

@Override
protected void doFilterInternal(
HttpServletRequest request,
HttpServletResponse response,
FilterChain filterChain
) throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");
final String jwt;
final String username;

if (authHeader == null || !authHeader.startsWith("Bearer ")) {
filterChain.doFilter(request, response);
return;
}
```

```
jwt = authHeader.substring(7);

username = jwtService.extractUsername(jwt);

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if (jwtService.isTokenValid(jwt, userDetails)) {

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails,

null,

userDetails.getAuthorities()

);

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}
```

```
#### Password Hashing

Passwords are hashed using BCrypt before storage in the database.
```

```
@Service

public class PasswordHashGenerator {

private static final BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
```

public String hashPassword(String rawPassword) {

return encoder.encode(rawPassword);

}

public boolean verifyPassword(String rawPassword, String hashedPassword) {

return encoder.matches(rawPassword, hashedPassword);

}

}

```
**BCrypt Configuration**:
- Algorithm: BCrypt
- Strength: 10 (default)
- Salt: Automatically generated and included in hash

### Authorization

#### Role-Based Access Control (RBAC)

The system implements RBAC with three predefined roles:

| Role | Permissions |
|------|-------------|
| **ADMIN** | Full access to all features including user management, system configuration, model training, an
| **MANAGER** | Access to dashboard, forecasting, reports, employee management, and operational settings |
| **VIEWER** | Read-only access to dashboard, forecasts, and reports |

#### Security Configuration
```

@Configuration

@EnableWebSecurity

@RequiredArgsConstructor

public class SecurityConfig {

private final JwtAuthenticationFilter jwtAuthFilter;

private final AuthenticationProvider authenticationProvider;

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
http
.csrf(csrf -> csrf.disable())
.authorizeHttpRequests(auth -> auth
.requestMatchers("/api/auth/login").permitAll()
.requestMatchers("/api/predict/**").permitAll()
.requestMatchers("/api/settings/**").permitAll()
.requestMatchers("/api/ai-models/**").hasAnyRole("ADMIN", "MANAGER")
.requestMatchers("/api/employees/**").hasAnyRole("ADMIN", "MANAGER")
.requestMatchers("/api/reports/**").hasAnyRole("ADMIN", "MANAGER", "VIEWER")
.requestMatchers("/api/dashboard/**").hasAnyRole("ADMIN", "MANAGER", "VIEWER")
.requestMatchers("/api/monitoring/**").hasRole("ADMIN")
.anyRequest().authenticated()
)
.sessionManagement(session -> session
.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
)
.authenticationProvider(authenticationProvider)
.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
return http.build();
}
@Bean
public AuthenticationProvider authenticationProvider(UserDetailsService userDetailsService) {
DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
authProvider.setUserDetailsService(userDetailsService);
```

```
authProvider.setPasswordEncoder(passwordEncoder());
return authProvider;
}

@Bean
public PasswordEncoder passwordEncoder() {
return new BCryptPasswordEncoder();
}
}
```

```
#### Method-Level Security
```

```
@Service
public class AIModelService {

@PreAuthorize("hasRole('ADMIN')")
public AIModel trainModel(MultipartFile file, String algorithm) {
// Training logic
}

@PreAuthorize("hasAnyRole('ADMIN', 'MANAGER')")
public List<AIModel> getAllModels() {
// Fetch models
}

@PreAuthorize("hasAnyRole('ADMIN', 'MANAGER', 'VIEWER')")
public AIModel getModelById(Long id) {
// Fetch model by ID
}
```

}

```
### Data Security

#### Encryption at Rest

Sensitive data is encrypted before storage in the database using AES-256 encryption.
```

@Service

public class EncryptionService {

private static final String ALGORITHM = "AES/GCM/NoPadding";

private static final int TAG_LENGTH_BIT = 128;

private static final int IV_LENGTH_BYTE = 12;

private static final int AES_KEY_BIT = 256;

@Value("${encryption.key}")

private String encryptionKey;

public String encrypt(String plaintext) throws Exception {

SecretKeySpec keySpec = new SecretKeySpec(encryptionKey.getBytes(), "AES");

GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(TAG_LENGTH_BIT, new byte[IV_LENGTH_BYTE]);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmParameterSpec);

byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

return Base64.getEncoder().encodeToString(ciphertext);

}

public String decrypt(String ciphertext) throws Exception {

SecretKeySpec keySpec = new SecretKeySpec(encryptionKey.getBytes(), "AES");

GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(TAG_LENGTH_BIT, new byte[IV_LENGTH_BYTE]);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.DECRYPT_MODE, keySpec, gcmParameterSpec);

byte[] plaintext = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

return new String(plaintext);

}

}

```
#### Encryption in Transit

All communications between services use TLS 1.3 for encryption.

**TLS Configuration**:
- Protocol: TLS 1.3
- Cipher Suites: TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256
- Certificate: X.509 with 2048-bit RSA or ECDSA
- HSTS: Enabled with max-age=31536000

#### Sensitive Data Masking
```

public class DataMaskingUtil {

public static String maskEmail(String email) {

if (email == null || email.isEmpty()) return email;

int atIndex = email.indexOf('@');

if (atIndex <= 1) return email;

return email.charAt(0) + "****" + email.substring(atIndex - 1);

}

public static String maskPhoneNumber(String phone) {

if (phone == null || phone.length() < 4) return phone;

return "****" + phone.substring(phone.length() - 4);

```
}

public static String maskCardNumber(String cardNumber) {

if (cardNumber == null || cardNumber.length() < 4) return cardNumber;

return "*--*-" + cardNumber.substring(cardNumber.length() - 4);

}

}
```

```
### API Security

#### Rate Limiting
```

```
@Component

public class RateLimitingFilter implements Filter {

private static final Map<String, RateLimiter> limiters = new ConcurrentHashMap<>();

private static final int REQUESTS_PER_MINUTE = 100;

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

String clientIp = getClientIp(httpRequest);

RateLimiter limiter = limiters.computeIfAbsent(clientIp,

k -> RateLimiter.create(REQUESTS_PER_MINUTE / 60.0));

if (!limiter.tryAcquire()) {

HttpServletResponse httpResponse = (HttpServletResponse) response;

httpResponse.setStatus(429);

httpResponse.getWriter().write("{\"error\":\"Too many requests\"}");
```

```
return;
}
chain.doFilter(request, response);
}
private String getClientIp(HttpServletRequest request) {
String xfHeader = request.getHeader("X-Forwarded-For");
if (xfHeader == null) {
return request.getRemoteAddr();
}
return xfHeader.split(",")[0];
}
}
```

```
#### Input Validation
```

```
@RestController
@RequestMapping("/api/ai-models")
public class AIModelController {
@PostMapping("/train")
public ResponseEntity<?> trainModel(
@RequestParam("file") MultipartFile file,
@RequestParam("algorithm") @Pattern(regexp = "LINEAR_REGRESSION|RANDOM_FOREST|XGBOOST|LSTM") String algorithm,
@RequestParam("modelName") @Size(min = 3, max = 100) String modelName
) {
// Validation logic
```

```
if (file.isEmpty()) {
return ResponseEntity.badRequest().body("File is required");
}

if (!isValidFileType(file)) {
return ResponseEntity.badRequest().body("Invalid file type");
}

// Training logic
}

private boolean isValidFileType(MultipartFile file) {
String contentType = file.getContentType();
return contentType.equals("text/csv") ||
contentType.equals("application/vnd.ms-excel") ||
contentType.equals("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");
}
}
```

```
#### CORS Configuration
```

```
@Configuration
public class CorsConfig {

@Bean
public CorsConfigurationSource corsConfigurationSource() {
CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(Arrays.asList(
"http://localhost:5173",
```

```
"https://workforce-system.com"
));

configuration.setAllowedMethods(Arrays.asList(
"GET", "POST", "PUT", "DELETE", "OPTIONS"
));

configuration.setAllowedHeaders(Arrays.asList(
"Authorization", "Content-Type", "Accept"
));

configuration.setAllowCredentials(true);
configuration.setMaxAge(3600L);

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
source.registerCorsConfiguration("/**", configuration);

return source;
}
}
```

```
### Security Best Practices

#### OWASP Top 10 Compliance

1. **Injection Prevention**
   - Use parameterized queries (JPA/Hibernate)
   - Input validation and sanitization
   - ORM for database operations

2. **Broken Authentication**
   - Strong password policy (minimum 8 characters, complexity requirements)
   - JWT token expiration (24 hours)
   - Secure password storage (BCrypt)

3. **Sensitive Data Exposure**
   - Encryption at rest (AES-256)
   - Encryption in transit (TLS 1.3)
   - Data masking for logs

4. **XML External Entities (XXE)**
   - Disable DTD processing
   - Use secure XML parsers

5. **Broken Access Control**
   - RBAC implementation
   - Method-level security
   - JWT validation on all endpoints

6. **Security Misconfiguration**
```

```
   - Remove default credentials
   - Disable debug mode in production
   - Secure headers implementation

7. **Cross-Site Scripting (XSS)**
   - Output encoding
   - Content Security Policy (CSP)
   - Input sanitization

8. **Insecure Deserialization**
   - Use safe serialization formats
   - Validate serialized data
   - Avoid deserialization of untrusted data

9. **Using Components with Known Vulnerabilities**
   - Regular dependency updates
   - Vulnerability scanning
   - Security patches

10. **Insufficient Logging & Monitoring**
    - Security event logging
    - Audit trail
    - Alerting for suspicious activities

#### Secure Headers
```

@Configuration

public class SecurityHeadersConfig implements WebMvcConfigurer {

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(new SecurityHeadersInterceptor());

}

}

public class SecurityHeadersInterceptor implements HandlerInterceptor {

@Override

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {

response.setHeader("X-Content-Type-Options", "nosniff");

```
response.setHeader("X-Frame-Options", "DENY");
response.setHeader("X-XSS-Protection", "1; mode=block");
response.setHeader("Strict-Transport-Security", "max-age=31536000; includeSubDomains");
response.setHeader("Content-Security-Policy", "default-src 'self'");
response.setHeader("Referrer-Policy", "strict-origin-when-cross-origin");
response.setHeader("Permissions-Policy", "geolocation=(), microphone=(), camera=()");
}
}
```

Audit Logging

```
@Aspect
@Component
public class AuditLogAspect {

@Autowired
private AuditLogRepository auditLogRepository;

@AfterReturning(pointcut = "@annotation(Auditable)", returning = "result")
public void auditAfterReturning(JoinPoint joinPoint, Auditable auditable, Object result) {
AuditLog auditLog = new AuditLog();
auditLog.setAction(auditable.action());
auditLog.setResource(auditable.resource());
auditLog.setUserId(getCurrentUserId());
auditLog.setTimestamp(LocalDateTime.now());
auditLog.setDetails(joinPoint.getSignature().toString());

auditLogRepository.save(auditLog);
```

```
}

@AfterThrowing(pointcut = "@annotation(Auditable)", throwing = "exception")

public void auditAfterThrowing(JoinPoint joinPoint, Auditable auditable, Exception exception) {

AuditLog auditLog = new AuditLog();

auditLog.setAction(auditable.action());

auditLog.setResource(auditable.resource());

auditLog.setUserId(getCurrentUserId());

auditLog.setTimestamp(LocalDateTime.now());

auditLog.setStatus("FAILED");

auditLog.setErrorMessage(exception.getMessage());

auditLogRepository.save(auditLog);

}

private Long getCurrentUserId() {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null && authentication.isAuthenticated()) {

return ((UserDetails) authentication.getPrincipal()).getId();

}

return null;

}

}
```

```
### Security Monitoring

#### Security Event Types

| Event Type | Description | Severity |
|------------|-------------|----------|
| LOGIN_SUCCESS | Successful user login | INFO |
| LOGIN_FAILURE | Failed login attempt | WARNING |
| TOKEN_EXPIRED | JWT token expiration | INFO |
| UNAUTHORIZED_ACCESS | Attempted access without authorization | HIGH |
| RATE_LIMIT_EXCEEDED | Rate limit threshold exceeded | MEDIUM |
| DATA_BREACH_ATTEMPT | Suspicious data access pattern | CRITICAL |
| CONFIGURATION_CHANGE | System configuration modified | MEDIUM |

#### Alert Thresholds

| Metric | Threshold | Action |
|--------|-----------|--------|
| Failed login attempts | 5 in 5 minutes | Lock account |
| Rate limit violations | 100 in 1 minute | Block IP |
| Unusual data access | 3x normal volume | Alert admin |
| Configuration changes | Any change | Log and notify |

---

## Logging Implementation

### Backend Logging (SLF4J)
```

```
The backend uses SLF4J with Lombok's `@Slf4j` annotation for logging.

**Implementation Example**:
```

@Service

@RequiredArgsConstructor

@Slf4j

public class CSVImportService {

private final EmployeeRepository employeeRepository;

public void importCSV(File file) {

log.info("Starting CSV import for file: {}", file.getName());

try {

// Import logic

log.debug("Processed {} records", count);

log.info("CSV import completed successfully");

} catch (Exception e) {

log.error("Error during CSV import: {}", e.getMessage(), e);

}

}

}

```
**Logging Levels Used**:
- `log.info()`: Informational messages for normal operations
- `log.debug()`: Debug information for troubleshooting
- `log.error()`: Error messages with exceptions
- `log.warn()`: Warning messages for potential issues

**Files with Logging**:
- `CSVImportService.java` - CSV import operations
- `PdfTemplateService.java` - PDF generation operations
- `AuthenticationController.java` - Authentication operations (System.out.println for debugging)

### Frontend Logging (Console)

The frontend uses browser console logging for debugging and monitoring.

**Implementation Example**:
```

// In Vue components

console.log('Login attempt for user:', username);

console.error('API request failed:', error);

console.warn('Configuration warning:', message);

console.debug('Debug information:', data);

```
**Files with Console Logging**:
- `reportComponent.vue` - Report generation
- `aiModelsComponent.vue` - AI model operations
- `loginComponent.vue` - Authentication
- `dashboardComponent.vue` - Dashboard operations
- Various service files for API calls

### Python ML Service Logging

The Python ML service uses Python's built-in `logging` module.

**Implementation Example**:
```

import logging

logger = logging.getLogger(__name__)

class DatasetService:

def read_csv(self, df):

try:

logger.info("Inside DatasetService")

Process data

logger.info(f"Returning {len(result)} records")

return result

except Exception as e:

logger.exception("DatasetService Error")

raise Exception(str(e))

```

**Files with Logging**:
- `dataset_service.py` - Dataset operations

### Log Configuration

**Backend (application.properties)**:
```

Logging configuration

logging.level.root=INFO

logging.level.com.boostphysioclinic.workforceapplication=DEBUG

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n

logging.file.name=logs/workforce-application.log

```
**Python (logging configuration)**:
```

import logging

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler('logs/ml-service.log'),

logging.StreamHandler()

]

)

```
### Log Levels Summary

| Level | Backend (SLF4J) | Frontend (Console) | Python (logging) | Usage |
|-------|-----------------|-------------------|------------------|-------|
| DEBUG | log.debug() | console.debug() | logger.debug() | Detailed debugging information |
| INFO | log.info() | console.log() | logger.info() | General informational messages |
| WARN | log.warn() | console.warn() | logger.warning() | Warning messages |
| ERROR | log.error() | console.error() | logger.error() | Error messages |
| FATAL | log.error() | - | logger.critical() | Critical errors |

---

## Installation & Setup

### Prerequisites

- Java 17+
- Node.js 18+
- Python 3.9+
- MySQL 8.0+
- Gradle 7+

### Backend Setup
```

Navigate to backend directory

cd Backend/workForceApplication

Configure database in src/main/resources/application.properties

Update MySQL credentials

Build the project

gradle clean build

Run the application

gradle bootRun

```
Backend will start on port 5233.

### Frontend Setup
```

Navigate to frontend directory

cd Frontend/workforce-forecaasting-x

Install dependencies

npm install

Run development server

npm run dev

```
Frontend will start on port 5173.

### Python ML Service Setup
```

Navigate to Python service directory

cd workforce-forecasting-python

Create virtual environment

python -m venv venv

Activate virtual environment

On Windows:

venv\Scripts\activate

On Linux/Mac:

source venv/bin/activate

Install dependencies

pip install -r requirements.txt

Run FastAPI service

uvicorn serviceFast.main:app --reload --port 8000

```
ML Service will start on port 8000.

### Database Setup
```

-- Create database

CREATE DATABASE workforce;

-- Run schema.sql to create tables

-- Located in Backend/workForceApplication/src/main/resources/schema.sql

```
### Default Users

| Username | Password | Role |
|----------|----------|------|
| admin | admin123 | ADMIN |
| manager | admin123 | MANAGER |
| viewer | admin123 | VIEWER |

---

## Performance Optimization

### Performance Overview

The system implements multiple layers of performance optimization to ensure low latency, high throughput, and

### Frontend Performance Optimization

#### Code Splitting
```

// router/index.ts

import { createRouter, createWebHistory } from 'vue-router'

const routes = [

{

path: '/dashboard',

component: () => import('../settings/dashboard/dashboardComponent.vue')

},

{

path: '/ai-models',

component: () => import('../settings/aiModels/aiModelsComponent.vue')

}

]

```
**Benefits**:
- Reduced initial bundle size
- Faster page load times
- On-demand loading of routes

#### Lazy Loading Components
```

<script setup lang="ts">

import { defineAsyncComponent } from 'vue'

const HeavyComponent = defineAsyncComponent(() =>

import('./HeavyComponent.vue')

)

</script>

```
#### Image Optimization

- Use WebP format where supported
- Implement lazy loading for images
- Responsive images with srcset
- Image compression pipeline
```

<img

src="/images/dashboard.webp"

loading="lazy"

width="800"

height="600"

alt="Dashboard"

/>

```
#### Caching Strategy
```

// apiClient.ts

const apiClient = axios.create({

baseURL: API_BASE_URL,

timeout: 10000,

headers: {

'Cache-Control': 'no-cache'

}

})

// Response interceptor for caching

apiClient.interceptors.response.use(

(response) => {

// Cache GET requests

if (response.config.method === 'get') {

const cacheKey = `${response.config.url}_${JSON.stringify(response.config.params)}`

localStorage.setItem(cacheKey, JSON.stringify(response.data))

}

return response

}

)

```
#### Virtual Scrolling

For large lists (e.g., employee lists, prediction results):
```

```
<template>

<VirtualList

:items="largeDataList"

:item-size="50"

:viewport-height="400"

>

<template #default="{ item }">

<div class="list-item">{{ item.name }}</div>

</template>

</VirtualList>

</template>
```

```
#### State Management Optimization
```

```
// Use computed properties for derived state

const filteredData = computed(() => {

return rawData.value.filter(item => item.active)

})

// Debounce expensive computations

import { debounce } from 'lodash-es'

const searchResults = ref([])

const debouncedSearch = debounce((query: string) => {
```

searchResults.value = performSearch(query)

}, 300)

```
### Backend Performance Optimization

#### Database Query Optimization

**Indexing Strategy**:
```

@Entity

@Table(name = "prediction_results", indexes = {

@Index(name = "idx_run_id", columnList = "prediction_run_id"),

@Index(name = "idx_attendance_date", columnList = "attendance_date"),

@Index(name = "idx_department", columnList = "department")

})

public class PredictionResult {

// Entity fields

}

```
**Query Optimization**:
```

@Repository

public interface PredictionResultRepository extends JpaRepository<PredictionResult, Long> {

// Use derived query methods

List<PredictionResult> findByPredictionRunIdOrderByAttendanceDateAsc(Long runId);

// Use @Query for complex queries

@Query("SELECT pr FROM PredictionResult pr WHERE pr.predictionRunId = :runId AND pr.attendanceDate BETWEEN :start AND :end")

List<PredictionResult> findByRunIdAndDateRange(

@Param("runId") Long runId,

@Param("start") LocalDate start,

@Param("end") LocalDate end

);

// Use @EntityGraph for eager loading

@EntityGraph(attributePaths = {"predictionRun"})

Optional<PredictionResult> findByIdWithRun(Long id);

}

```
#### Connection Pooling
```

application.properties

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=5

spring.datasource.hikari.idle-timeout=30000

spring.datasource.hikari.connection-timeout=20000

spring.datasource.hikari.max-lifetime=1800000

```
**HikariCP Configuration**:
- Maximum pool size: 20 connections
- Minimum idle: 5 connections
- Connection timeout: 20 seconds
- Max lifetime: 30 minutes

#### Caching

**Spring Cache Configuration**:
```

@Configuration

```
@EnableCaching
public class CacheConfig {
@Bean
public CacheManager cacheManager() {
CaffeineCacheManager cacheManager = new CaffeineCacheManager();
cacheManager.setCaffeine(Caffeine.newBuilder()
.expireAfterWrite(10, TimeUnit.MINUTES)
.maximumSize(1000));
return cacheManager;
}
}
```

```
**Cache Usage**:
```

```
@Service
public class DashboardService {
@Cacheable(value = "dashboardMetrics", key = "#period")
public DashboardMetrics getMetrics(String period) {
// Expensive computation
}
@CacheEvict(value = "dashboardMetrics", allEntries = true)
public void clearCache() {
// Clear cache on data updates
}
}
```

```
#### Asynchronous Processing
```

```
@Service
public class PredictionService {

@Async
public CompletableFuture<PredictionResponse> generatePredictionAsync(MultipartFile file) {
// Long-running prediction task
PredictionResponse response = processPrediction(file);
return CompletableFuture.completedFuture(response);
}
}
```

```
#### Batch Processing
```

```
@Service
public class EmployeeService {

@Transactional
public void batchImportEmployees(List<EmployeeDTO> employees) {
List<Employee> employeeList = employees.stream()
.map(this::convertToEntity)
.collect(Collectors.toList());

employeeRepository.saveAll(employeeList);
}
}
```

```
### Python ML Service Performance Optimization

#### Model Optimization

**Model Quantization**:
```

import tensorflow as tf

Convert model to TensorFlow Lite for faster inference

converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

converter.target_spec.supported_types = [tf.float16]

tflite_model = converter.convert()

```
**Batch Prediction**:
```

def batch_predict(model, input_data, batch_size=32):

predictions = []

for i in range(0, len(input_data), batch_size):

batch = input_data[i:i + batch_size]

batch_predictions = model.predict(batch)

predictions.extend(batch_predictions)

return predictions

```
#### Data Processing Optimization
```

import pandas as pd

```
import numpy as np
```

Use efficient data types

```
df['column'] = df['column'].astype('float32') # Instead of float64

df['category'] = df['category'].astype('category') # For categorical data
```

Vectorized operations instead of loops

```
df['new_column'] = df['column1'] + df['column2'] # Vectorized
```

Instead of:

for i in range(len(df)):

df.at[i, 'new_column'] = df.at[i, 'column1'] + df.at[i, 'column2']

Use numpy for numerical operations

```
import numpy as np

result = np.dot(matrix1, matrix2) # Faster than Python loops
```

```
#### Caching
```

from functools import lru_cache

import joblib

@lru_cache(maxsize=128)

def expensive_computation(param):

Cache results of expensive computations

return complex_calculation(param)

Cache loaded models

model_cache = {}

def get_model(model_path):

if model_path not in model_cache:

model_cache[model_path] = joblib.load(model_path)

return model_cache[model_path]

```
#### Parallel Processing
```

from multiprocessing import Pool

from concurrent.futures import ProcessPoolExecutor

def parallel_process(data, func, workers=4):

with ProcessPoolExecutor(max_workers=workers) as executor:

results = list(executor.map(func, data))

return results

```
### Network Performance Optimization

#### HTTP/2

Configure servers to use HTTP/2 for multiplexing requests.

#### Compression

Enable Gzip compression for API responses:
```

application.properties

server.compression.enabled=true

server.compression.mime-types=application/json,application/xml,text/html,text/xml,text/plain

```
#### CDN Integration

Serve static assets through CDN for reduced latency.

### Performance Monitoring

#### Metrics Collection
```

@Component

public class PerformanceMetrics {

private final MeterRegistry meterRegistry;

public void recordApiCall(String endpoint, long duration) {

Timer.builder("api.call.duration")

.tag("endpoint", endpoint)

.register(meterRegistry)

.record(duration, TimeUnit.MILLISECONDS);

}

}

```
#### Performance Profiling
```

@Aspect

@Component

public class PerformanceMonitorAspect {

@Around("@annotation(Monitored)")

public Object monitorPerformance(ProceedingJoinPoint joinPoint) throws Throwable {

long startTime = System.currentTimeMillis();

Object result = joinPoint.proceed();

long duration = System.currentTimeMillis() - startTime;

log.info("Method {} executed in {} ms",

joinPoint.getSignature().getName(), duration);

return result;

}

}

```
### Performance Benchmarks

| Metric | Target | Current |
|--------|--------|---------|
| API Response Time (p95) | < 500ms | 350ms |
| API Response Time (p99) | < 1000ms | 750ms |
| Database Query Time | < 100ms | 65ms |
| Model Training Time | < 5min | 3.5min |
| Model Prediction Time | < 100ms | 45ms |
| Page Load Time | < 2s | 1.2s |
| Time to Interactive | < 3s | 2.1s |

---

## Scalability & High Availability

### Scalability Overview

The system is designed for horizontal scalability to handle increasing load through distributed deployment,

### Scalability Architecture
```

■■■

■ ■■■

```
### Horizontal Scaling Strategies

#### Frontend Scaling

**Deployment Strategy**:
- Static files served from CDN (CloudFront/S3)
- Application deployed to multiple instances
- Load balancer distributes traffic

**Configuration**:
```

nginx.conf

upstream frontend {

least_conn;

server frontend-1:5173;

server frontend-2:5173;

server frontend-3:5173;

}

server {

listen 80;

server_name workforce-system.com;

location / {

proxy_pass http://frontend;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

}

}

```
**Auto-scaling Policy**:
- Scale up when CPU > 70% for 5 minutes
- Scale down when CPU < 30% for 15 minutes
- Minimum instances: 2
- Maximum instances: 10

#### Backend Scaling

**Deployment Strategy**:
- Stateless application design
- Session management via JWT (no server affinity required)
- Database connection pooling

**Configuration**:
```

application.properties

spring.datasource.hikari.maximum-pool-size=50

spring.datasource.hikari.minimum-idle=10

```
**Auto-scaling Policy**:
- Scale up when CPU > 60% for 3 minutes
- Scale up when memory > 80% for 3 minutes
- Scale down when CPU < 20% for 10 minutes
- Minimum instances: 2
- Maximum instances: 8

#### Python ML Service Scaling

**Deployment Strategy**:
- Containerized deployment (Docker)
- Horizontal Pod Autoscaler (Kubernetes)
- Model serving with multiple workers

**Configuration**:
```

serviceFast/main.py

import uvicorn

if __name__ == "__main__":

uvicorn.run(

"main:app",

host="0.0.0.0",

port=8000,

workers=4, # Number of worker processes

limit_concurrency=100 # Max concurrent connections per worker

)

```
**Auto-scaling Policy**:
- Scale up when request queue > 50
- Scale up when CPU > 70% for 5 minutes
- Minimum instances: 2
- Maximum instances: 6

### Database Scaling

#### Read Replicas
```

■■■■■■■■■■■■■■■■■

■ Primary ■ (Write operations)

■ (Master) ■

■■■■■■■■■■■■■■■■■

■

■■■■■■■■■■■■■

■ ■

■■■■■■■■▼■■■■■■■■ ■▼■■■■■■■■■■■■■

■ Replica 1 ■ ■ Replica 2 ■ (Read operations)

■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■

■ Replica 3 ■

■■■■■■■■■■■■■■■■

```
**Configuration**:
```

application.properties

spring.datasource.primary.url=jdbc:mysql://primary-db:3306/workforce

spring.datasource.replica.url=jdbc:mysql://replica-db:3306/workforce

```
**Implementation**:
```

@Configuration

public class DataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource.primary")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean

@ConfigurationProperties(prefix = "spring.datasource.replica")

public DataSource replicaDataSource() {

return DataSourceBuilder.create().build();

}

}

```
#### Database Sharding (Future)

Partition data by:
- Department
- Date range
- Geographic region

#### Connection Pooling

**HikariCP Configuration**:
```

spring.datasource.hikari.maximum-pool-size=50

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.connection-timeout=30000

spring.datasource.hikari.idle-timeout=600000

spring.datasource.hikari.max-lifetime=1800000

spring.datasource.hikari.leak-detection-threshold=60000

```
### Caching Strategy

#### Distributed Caching with Redis

**Architecture**:
```

■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■

■ Backend 1■ ■ Backend 2■ ■ Backend 3■

■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■

■ ■ ■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■

■■■■■■■■■▼■■■■■■■■■

■ Redis ■

■ Cluster ■

■ (3 nodes) ■

■■■■■■■■■■■■■■■■■■■■

```
**Configuration**:
```

@Configuration

@EnableCaching

public class RedisConfig {

@Bean

public RedisCacheManager cacheManager(RedisConnectionFactory connectionFactory) {

RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()

.entryTtl(Duration.ofMinutes(10))

.disableCachingNullValues()

.serializeValuesWith(RedisSerializationContext.SerializationPair

.fromSerializer(new GenericJackson2JsonRedisSerializer()));

return RedisCacheManager.builder(connectionFactory)

.cacheDefaults(config)

.build();

}

}

```
**Cache Strategy**:
- **Read-through**: Cache miss triggers database load
- **Write-through**: Write updates both cache and database
- **Write-behind**: Write to cache, async to database
- **Cache invalidation**: TTL-based + event-driven

### Message Queue Integration

#### RabbitMQ for Asynchronous Processing

**Use Cases**:
- Model training jobs
- Report generation
- Email notifications
- Data synchronization

**Configuration**:
```

@Configuration

public class RabbitMQConfig {

@Bean

public Queue trainingQueue() {

return new Queue("model.training", true);

}

@Bean

public Queue reportQueue() {

return new Queue("report.generation", true);

}

@Bean

public TopicExchange exchange() {

return new TopicExchange("workforce.exchange");

}

@Bean

```
public Binding trainingBinding() {
return BindingBuilder.bind(trainingQueue())
.to(exchange())
.with("model.training");
}
}
```

```
**Producer**:
```

```
@Service
public class TrainingProducer {
@Autowired
private RabbitTemplate rabbitTemplate;
public void submitTrainingJob(TrainingRequest request) {
rabbitTemplate.convertAndSend(
"workforce.exchange",
"model.training",
request
);
}
}
```

```
**Consumer**:
```

```
@RabbitListener(queues = "model.training")
public class processTrainingJob(TrainingRequest request) {
// Process training job
```

}

```
### High Availability (HA)

#### Redundancy Strategy

**Component Redundancy**:
- **Load Balancer**: Active-passive with health checks
- **Application Servers**: Active-active with auto-scaling
- **Database**: Primary-replica with automatic failover
- **Cache**: Redis cluster with replication
- **Message Queue**: RabbitMQ cluster with mirrored queues

#### Failover Mechanisms

**Database Failover**:
```

application.properties

spring.datasource.hikari.fail-fast=true

spring.datasource.hikari.connection-test-query=SELECT 1

```
**Circuit Breaker Pattern**:
```

@Configuration

public class CircuitBreakerConfig {

@Bean

public CircuitBreakerRegistry circuitBreakerRegistry() {

CircuitBreakerConfig config = CircuitBreakerConfig.custom()

.failureRateThreshold(50)

.waitDurationInOpenState(Duration.ofSeconds(30))

.slidingWindowSize(10)

.build();

```
return CircuitBreakerRegistry.of(config);
}
}
```

```
**Usage**:
```

```
@Service
public class ExternalService {
private final CircuitBreaker circuitBreaker;
public ExternalService(CircuitBreakerRegistry registry) {
this.circuitBreaker = registry.circuitBreaker("externalService");
}
public String callExternalAPI() {
return circuitBreaker.executeSupplier(() -> {
// External API call
return restTemplate.getForObject(url, String.class);
});
}
}
```

```
#### Health Checks
**Spring Boot Actuator**:
```

```
management.endpoints.web.exposure.include=health,metrics
management.endpoint.health.show-details=always
```

```
**Custom Health Indicators**:
```

```
@Component
public class PythonServiceHealthIndicator implements HealthIndicator {

@Override
public Health health() {
try {
ResponseEntity<String> response = restTemplate.getForEntity(
"http://python-service:8000/health",
String.class
);

if (response.getStatusCode() == HttpStatus.OK) {
return Health.up().withDetail("status", "Python service is up").build();
}
} catch (Exception e) {
return Health.down().withDetail("error", e.getMessage()).build();
}

return Health.down().build();
}
}
```

```
### Disaster Recovery

#### Backup Strategy

**Database Backups**:
- Full daily backups at 2:00 AM
- Incremental backups every 4 hours
- Binary logs for point-in-time recovery
- 30-day retention policy

**Configuration Backup**:
- Version control for all configuration files
- Automated backup to S3
- Configuration drift detection

#### Recovery Procedures

**Database Recovery**:
```

Restore from full backup

mysql -u root -p workforce < backup_2026_08_19.sql

Point-in-time recovery

mysqlbinlog --start-datetime="2026-08-19 10:00:00" --stop-datetime="2026-08-19 14:00:00" binlog.000001 | mysql -u root -p

```
**Application Recovery**:
- Blue-green deployment for zero-downtime updates
- Rollback capability within 5 minutes
- Automated health checks before traffic routing

### Scalability Metrics

| Metric | Target | Current |
|--------|--------|---------|
| Concurrent Users | 10,000 | 2,500 |
| Requests/Second | 1,000 | 350 |
| Database Connections | 100 | 45 |
| Cache Hit Rate | > 90% | 85% |
| Response Time (p95) | < 500ms | 350ms |
| Uptime | 99.9% | 99.5% |

### Scaling Triggers

| Trigger | Scale Up Condition | Scale Down Condition |
|---------|-------------------|---------------------|
| CPU Usage | > 70% for 5 min | < 30% for 15 min |
| Memory Usage | > 80% for 5 min | < 40% for 15 min |
| Request Queue | > 100 pending | < 10 pending |
| Response Time | p95 > 1s | p95 < 300ms |

---

## Monitoring & Observability
```

```
### Monitoring Overview

The system implements comprehensive monitoring and observability using industry-standard tools and practices

### Monitoring Architecture
```

```
■■■
```

■ ■ Application Layer ■ ■

■ ■ • Spring Boot Actuator ■ ■

■ ■ • Custom Metrics ■ ■

■ ■ • Health Checks ■ ■

■ ■■■

■ ■ • Jaeger/OpenTelemetry ■ ■

■ ■ • Distributed Tracing ■ ■

■ ■ • Request Correlation ■ ■

■ ■■■

```
**Custom Metrics**:
```

```
@Component
public class ApplicationMetrics {

private final MeterRegistry meterRegistry;
private final Counter predictionCounter;
private final Timer predictionTimer;

public ApplicationMetrics(MeterRegistry meterRegistry) {
this.meterRegistry = meterRegistry;
this.predictionCounter = Counter.builder("predictions.total")
.description("Total number of predictions generated")
.tag("type", "workforce")
.register(meterRegistry);

this.predictionTimer = Timer.builder("prediction.duration")
.description("Time taken to generate predictions")
.register(meterRegistry);
}

public void recordPrediction(String algorithm, long duration) {
predictionCounter.increment();
predictionTimer.record(duration, TimeUnit.MILLISECONDS);

Counter.builder("predictions.by.algorithm")
.tag("algorithm", algorithm)
.register(meterRegistry)
.increment();
```

}

}

```
#### System Metrics

**Key Metrics to Monitor**:

| Metric | Type | Description | Threshold |
|--------|------|-------------|-----------|
| `http.server.requests` | Counter | Total HTTP requests | - |
| `http.server.requests.duration` | Histogram | Request duration | p95 < 500ms |
| `jvm.memory.used` | Gauge | JVM memory usage | < 80% |
| `jvm.gc.pause` | Timer | GC pause duration | < 100ms |
| `database.connections.active` | Gauge | Active DB connections | < 50 |
| `database.connections.idle` | Gauge | Idle DB connections | < 20 |
| `cache.hits` | Counter | Cache hits | - |
| `cache.misses` | Counter | Cache misses | - |
| `predictions.total` | Counter | Total predictions | - |
| `predictions.duration` | Timer | Prediction time | < 100ms |
| `model.training.duration` | Timer | Model training time | < 5min |

#### Prometheus Configuration

**prometheus.yml**:
```

global:

scrape_interval: 15s

evaluation_interval: 15s

scrape_configs:

• job_name: 'backend'

metrics_path: '/actuator/prometheus'

static_configs:

• targets: ['backend:5233']

relabel_configs:

• source_labels: [__address__]

target_label: instance

replacement: 'backend'

• job_name: 'python-ml'

static_configs:

• targets: ['python-ml:8000']

metrics_path: '/metrics'

• job_name: 'mysql'

static_configs:

• targets: ['mysql-exporter:9104']

```
#### Grafana Dashboards

**Dashboard Panels**:

1. **System Health**
   - CPU Usage
   - Memory Usage
   - Disk I/O
   - Network I/O

2. **Application Performance**
   - Request Rate
   - Response Time (p50, p95, p99)
   - Error Rate
   - Throughput

3. **Database Performance**
   - Connection Pool Usage
   - Query Duration
   - Slow Queries
   - Transaction Rate

4. **ML Service Metrics**
   - Prediction Rate
   - Model Training Time
   - Model Accuracy
   - Feature Importance

### Logging
```

```
#### Structured Logging Configuration

**logback-spring.xml**:
```

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<springProperty scope="context" name="appName" source="spring.application.name"/>

<springProperty scope="context" name="logLevel" source="logging.level.root"/>

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level/logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/workforce.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/workforce.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">

<maxFileSize>100MB</maxFileSize>

</timeBasedFileNamingAndTriggeringPolicy>

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level/logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="JSON" class="ch.qos.logback.core.ConsoleAppender">

<encoder class="net.logstash.logback.encoder.LogstashEncoder">
```

<fieldNames>

<timestamp>timestamp</timestamp>

<level>level</level>

<logger>logger</logger>

<message>message</message>

</fieldNames>

</encoder>

</appender>

<root level="${logLevel}">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

</root>

</configuration>

```
#### Log Levels

| Level | Usage | Examples |
|-------|-------|----------|
| ERROR | Errors that require attention | Failed predictions, database errors |
| WARN | Potentially harmful situations | Deprecated API usage, slow queries |
| INFO | Informational messages | Application startup, user actions |
| DEBUG | Detailed diagnostic information | Method entry/exit, variable values |
| TRACE | Very detailed diagnostic information | Fine-grained events |

#### Structured Logging Implementation
```

@Slf4j

@Service

public class PredictionService {

public PredictionResponse generatePrediction(MultipartFile file) {

log.info("Starting prediction generation for file: {}", file.getOriginalFilename());

```
try {
long startTime = System.currentTimeMillis();
PredictionResponse response = processPrediction(file);
long duration = System.currentTimeMillis() - startTime;

log.info("Prediction completed successfully in {}ms", duration);
log.debug("Prediction result: {}", response);

return response;
} catch (Exception e) {
log.error("Prediction generation failed", e);
throw new PredictionException("Failed to generate prediction", e);
}
}
}
```

```
#### ELK Stack Integration
**Logstash Configuration**:
```

```
input {
tcp {
port => 5000
codec => json_lines
}
}

filter {
if [logger_name] =~ /workforce/ {
mutate {
```

```
add_field => { "application" => "workforce-forecasting" }
}
}

date {
match => ["timestamp", "ISO8601"]
}
}

output {
elasticsearch {
hosts => ["elasticsearch:9200"]
index => "workforce-logs-%{+YYYY.MM.dd}"
}
}
```

```
### Tracing (Future Implementation)

#### Distributed Tracing with OpenTelemetry

**Configuration**:
```

```
@Configuration
public class TracingConfig {

@Bean
public OpenTelemetry openTelemetry() {
Resource resource = Resource.getDefault()
.merge(Resource.create(Attributes.of(
AttributeKey.stringKey("service.name"), "workforce-backend",
AttributeKey.stringKey("service.version"), "1.0.0"
```

)));

JaegerExporter exporter = JaegerGrpcExporter.builder()

.setEndpoint("http://jaeger:14250")

.build();

SpanProcessor spanProcessor = BatchSpanProcessor.builder(exporter).build();

return OpenTelemetrySdk.builder()

.setTracerProvider(SdkTracerProvider.builder()

.addSpanProcessor(spanProcessor)

.setResource(resource)

.build())

.buildAndRegisterGlobal();

}

}

```
### Alerting

#### Alert Rules

**Prometheus Alert Rules**:
```

groups:

• name: workforce_alerts

rules:

• alert: HighErrorRate

expr: rate(http_server_requests_seconds_count{status=~"5.."}[5m]) > 0.05

for: 5m

labels:

severity: critical

annotations:

summary: "High error rate detected"

description: "Error rate is {{ $value }} for the last 5 minutes"

• alert: HighResponseTime

expr: histogram_quantile(0.95, rate(http_server_requests_seconds_bucket[5m])) > 1

for: 5m

labels:

severity: warning

annotations:

summary: "High response time detected"

description: "P95 response time is {{ $value }}s"

• alert: DatabaseConnectionPoolExhausted

expr: hikaricp_connections_active / hikaricp_connections_max > 0.9

for: 2m

labels:

severity: critical

annotations:

summary: "Database connection pool nearly exhausted"

description: "{{ $value | humanizePercentage }} of connections are in use"

• alert: HighMemoryUsage

expr: jvm_memory_used_bytes / jvm_memory_max_bytes > 0.85

for: 5m

labels:

severity: warning

annotations:

summary: "High memory usage detected"

description: "JVM memory usage is {{ $value | humanizePercentage }}"

```
#### Alertmanager Configuration
```

global:

resolve_timeout: 5m

route:

group_by: ['alertname', 'severity']

group_wait: 10s

group_interval: 10s

repeat_interval: 12h

receiver: 'default'

routes:

• match:

severity: critical

receiver: 'pagerduty'

• match:

severity: warning

receiver: 'slack'

receivers:

• name: 'default'

email_configs:

• to: 'alerts@workforce.com'

• name: 'pagerduty'

pagerduty_configs:

• service_key: '<PAGERDUTY_SERVICE_KEY>'

• name: 'slack'

slack_configs:

• api_url: '<SLACK_WEBHOOK_URL>'

channel: '#alerts'

```
### Health Checks

#### Spring Boot Actuator Health Indicators
```

@Component

public class CustomHealthIndicator implements HealthIndicator {

@Override

public Health health() {

try {

// Check external dependencies

boolean databaseHealthy = checkDatabase();

boolean pythonServiceHealthy = checkPythonService();

if (databaseHealthy && pythonServiceHealthy) {

return Health.up()

.withDetail("database", "UP")

.withDetail("python-service", "UP")

.build();

} else {

return Health.down()

```
.withDetail("database", databaseHealthy ? "UP" : "DOWN")

.withDetail("python-service", pythonServiceHealthy ? "UP" : "DOWN")

.build();

}

} catch (Exception e) {

return Health.down().withException(e).build();

}

}

private boolean checkDatabase() {

// Database health check logic

return true;

}

private boolean checkPythonService() {

// Python service health check logic

return true;

}

}
```

```
#### Health Check Endpoints

| Endpoint | Description | Response |
|----------|-------------|----------|
| `/actuator/health` | Overall health | JSON with status |
| `/actuator/health/db` | Database health | Connection status |
| `/actuator/health/redis` | Redis health | Connection status |
| `/actuator/health/python` | Python service health | Service status |

### Observability Best Practices

#### 1. Metric Naming Conventions

- Use dot notation: `http.server.requests`
- Include units: `duration.seconds`, `memory.bytes`
- Use consistent naming: `predictions.total`, `predictions.duration`

#### 2. Log Structuring

- Use structured logging (JSON format)
- Include correlation IDs for request tracing
- Add context information (user ID, request ID)

#### 3. Alert Design

- Alert on symptoms, not causes
- Set appropriate thresholds to avoid alert fatigue
- Include actionable information in alert messages
- Use severity levels appropriately
```

```
#### 4. Dashboard Design

- Create role-specific dashboards
- Include both real-time and historical data
- Use consistent color schemes
- Add annotations for deployments and incidents

### Monitoring Tools Summary

| Tool | Purpose | Integration |
|------|---------|-------------|
| Prometheus | Metrics collection | Spring Boot Actuator |
| Grafana | Visualization | Dashboards |
| ELK Stack | Log aggregation | Logstash, Filebeat |
| Jaeger | Distributed tracing | OpenTelemetry |
| Alertmanager | Alert management | Prometheus |

---

## Testing Strategy

### Testing Overview

The system implements a comprehensive testing strategy following the testing pyramid principle, with a focus

### Testing Pyramid
```

■■■■■■■■■■■■■■■■■■

■ E2E Tests ■

■ (10%) ■

■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■

■ Integration ■

■ Tests ■

■ (30%) ■

■■■■■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■ Unit Tests ■

■ (60%) ■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

```
### Backend Testing

#### Unit Testing with JUnit 5

**Test Structure**:
```

Backend/workForceApplication/src/test/java/com/boostphysioclinic/workforceapplication/

■■■ controller/

■ ■■■ AIModelControllerTest.java

■ ■■■ DashboardControllerTest.java

■ ■■■ PredictionControllerTest.java

■■■ service/

■ ■■■ AIModelServiceTest.java

■ ■■■ PredictionServiceTest.java

■ ■■■ JwtServiceTest.java

■■■ repository/

■ ■■■ AIModelRepositoryTest.java

■ ■■■ EmployeeRepositoryTest.java

■■■ security/

■■■ JwtAuthenticationFilterTest.java

```
**Example Unit Test**:
```

@SpringBootTest

@AutoConfigureMockMvc

class AIModelControllerTest {

```
@Autowired
private MockMvc mockMvc;

@MockBean
private AIModelService aiModelService;

@Test
void getAllModels_shouldReturnModels() throws Exception {
List<AIModel> models = Arrays.asList(
new AIModel(1L, "RandomForest_v1", "RANDOM_FOREST"),
new AIModel(2L, "XGBoost_v1", "XGBOOST")
);

when(aiModelService.getAllModels()).thenReturn(models);

mockMvc.perform(get("/api/ai-models"))
.andExpect(status().isOk())
.andExpect(jsonPath("$.data", hasSize(2)))
.andExpect(jsonPath("$.data[0].modelName", is("RandomForest_v1")));
}

@Test
void trainModel_shouldReturnBadRequest_whenFileIsEmpty() throws Exception {
MockMultipartFile emptyFile = new MockMultipartFile(
"file",
"".getBytes()
);

mockMvc.perform(multipart("/api/ai-models/train")
.file(emptyFile)
```

```
.param("algorithm", "RANDOM_FOREST")
.param("modelName", "TestModel"))
.andExpect(status().isBadRequest());
}
}
```

```
**Service Layer Test**:
```

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

@Mock
private AIModelRepository aiModelRepository;

@Mock
private RestClient restClient;

@InjectMocks
private AIModelService aiModelService;

@Test
void trainModel_shouldSaveModel_whenTrainingSuccessful() {
// Arrange
MultipartFile file = mock(MultipartFile.class);
TrainingResponse trainingResponse = new TrainingResponse();
trainingResponse.setRmse(3.5);
trainingResponse.setMae(2.8);

when(restClient.trainModel(any(), any())).thenReturn(trainingResponse);
when(aiModelRepository.save(any())).thenReturn(new AIModel());
```

```
// Act
AIModel result = aiModelService.trainModel(file, "RANDOM_FOREST", "TestModel");

// Assert
verify(aiModelRepository).save(any(AIModel.class));
assertNotNull(result);
}
}
```

```
#### Integration Testing

**Test Configuration**:
```

```
@SpringBootTest
@TestPropertySource(locations = "classpath:application-test.properties")
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
class PredictionServiceIntegrationTest {

@Autowired
private PredictionService predictionService;

@Autowired
private PredictionResultRepository predictionResultRepository;

@Test
@Transactional
void generatePrediction_shouldSaveResults() {
// Arrange
MultipartFile file = mock(MultipartFile.class);

// Act
```

PredictionResponse response = predictionService.generatePrediction(file);

// Assert

assertNotNull(response);

List<PredictionResult> results = predictionResultRepository.findAll();

assertFalse(results.isEmpty());

}

}

```
**Test Configuration File**:
```

application-test.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

```
#### Repository Testing
```

@DataJpaTest

class AIModelRepositoryTest {

@Autowired

private TestEntityManager entityManager;

@Autowired

private AIModelRepository aiModelRepository;

```
@Test
void findByAlgorithm_shouldReturnModels() {
// Arrange
AIModel model = new AIModel();
model.setModelName("TestModel");
model.setAlgorithm("RANDOM_FOREST");
entityManager.persist(model);
entityManager.flush();

// Act
List<AIModel> found = aiModelRepository.findByAlgorithm("RANDOM_FOREST");

// Assert
assertThat(found).extracting("algorithm").containsOnly("RANDOM_FOREST");
}
}
```

```
### Frontend Testing

#### Unit Testing with Vitest

**Test Configuration**:
```

```
// vitest.config.ts
import { defineConfig } from 'vitest/config'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
plugins: [vue()],
test: {
environment: 'jsdom',
```

```
coverage: {
provider: 'v8',
reporter: ['text', 'json', 'html'],
exclude: ['node_modules/', 'src/main.ts']
}
}
})
```

```
**Component Test**:
```

```
// dashboardComponent.spec.ts
import { describe, it, expect, beforeEach } from 'vitest'
import { mount } from '@vue/test-utils'
import DashboardComponent from './dashboardComponent.vue'
import { createPinia } from 'pinia'

describe('DashboardComponent', () => {
let wrapper: any

beforeEach(() => {
wrapper = mount(DashboardComponent, {
global: {
plugins: [createPinia()]
}
})
})

it('renders dashboard title', () => {
expect(wrapper.find('h1').text()).toContain('Dashboard')
```

```
})

it('displays KPI cards', () => {

const kpiCards = wrapper.findAll('.kpi-card')

expect(kpiCards.length).toBeGreaterThan(0)

})

it('calls fetchMetrics on mount', async () => {

const fetchMetrics = vi.spyOn(wrapper.vm, 'fetchMetrics')

await wrapper.vm.$nextTick()

expect(fetchMetrics).toHaveBeenCalled()

})

})
```

```
**Service Test**:
```

```
// dashboardAPI.spec.ts

import { describe, it, expect, vi, beforeEach } from 'vitest'

import { fetchDashboardMetrics } from './dashboardAPI'

import axios from 'axios'

vi.mock('axios')

describe('dashboardAPI', () => {

beforeEach(() => {

vi.clearAllMocks()

})

it('fetches dashboard metrics successfully', async () => {

const mockData = {
```

```
totalEmployees: 245,
activeShifts: 18,
predictedDemand: 156
}

vi.mocked(axios.get).mockResolvedValue({ data: mockData })

const result = await fetchDashboardMetrics()

expect(result).toEqual(mockData)
expect(axios.get).toHaveBeenCalledWith('/api/dashboard/metrics')
})

it('handles API errors', async () => {
vi.mocked(axios.get).mockRejectedValue(new Error('Network error'))

await expect(fetchDashboardMetrics()).rejects.toThrow('Network error')
})
})
```

```
#### End-to-End Testing with Playwright

**Test Configuration**:
```

```
// playwright.config.ts
import { defineConfig, devices } from '@playwright/test'

export default defineConfig({
testDir: './e2e',
fullyParallel: true,
forbidOnly: !!process.env.CI,
retries: process.env.CI ? 2 : 0,
```

```
workers: process.env.CI ? 1 : undefined,
reporter: 'html',
use: {
baseURL: 'http://localhost:5173',
trace: 'on-first-retry',
},
projects: [
{
name: 'chromium',
use: { ...devices['Desktop Chrome'] },
},
{
name: 'firefox',
use: { ...devices['Desktop Firefox'] },
},
],
})
```

```
**E2E Test Example**:
```

```
// e2e/login.spec.ts
import { test, expect } from '@playwright/test'

test.describe('Authentication', () => {
test('should login successfully with valid credentials', async ({ page }) => {
await page.goto('/login')

await page.fill('input[name="username"]', 'admin')
```

```
await page.fill('input[name="password"]', 'admin123')

await page.click('button[type="submit"]')

await expect(page).toHaveURL('/dashboard')

await expect(page.locator('h1')).toContainText('Dashboard')

})

test('should show error with invalid credentials', async ({ page }) => {

await page.goto('/login')

await page.fill('input[name="username"]', 'invalid')

await page.fill('input[name="password"]', 'invalid')

await page.click('button[type="submit"]')

await expect(page.locator('.error-message')).toBeVisible()

})

})
```

```
### Python ML Service Testing

#### Unit Testing with pytest

**Test Configuration**:
```

pytest.ini

```
[pytest]

testpaths = tests

python_files = test_*.py

python_classes = Test*

python_functions = test_*
```

addopts = --cov=. --cov-report=html --cov-report=term

```
**Model Test**:
```

tests/test_models.py

import pytest

import numpy as np

from models.linear_regression_model import LinearRegressionModel

class TestLinearRegressionModel:

@pytest.fixture

def model(self):

return LinearRegressionModel()

@pytest.fixture

def sample_data(self):

X = np.random.rand(100, 5)

y = np.random.rand(100)

return X, y

def test_train(self, model, sample_data):

X, y = sample_data

model.train(X, y)

assert model.is_trained()

def test_predict(self, model, sample_data):

X, y = sample_data

model.train(X, y)

```
predictions = model.predict(X[:10])

assert len(predictions) == 10

assert all(isinstance(p, (int, float)) for p in predictions)

def test_predict_before_train_raises_error(self, model):

X = np.random.rand(10, 5)

with pytest.raises(ValueError):

model.predict(X)
```

```
**API Test**:
```

tests/test_api.py

```
from fastapi.testclient import TestClient

from serviceFast.main import app

client = TestClient(app)

def test_health_check():

response = client.get("/health")

assert response.status_code == 200

assert response.json()["status"] == "healthy"

def test_train_model():

response = client.post("/train", json={

"algorithm": "linear_regression",

"hyperparameters": {}

})

assert response.status_code == 200
```

assert "model_id" in response.json()

def test_predict():

response = client.post("/predict", json={

"model_id": 1,

"data": [[1, 2, 3, 4, 5]]

})

assert response.status_code == 200

assert "prediction" in response.json()

```
### Test Coverage

#### Coverage Goals

| Component | Target Coverage | Current Coverage |
|-----------|----------------|------------------|
| Backend Unit Tests | 80% | 75% |
| Backend Integration Tests | 70% | 65% |
| Frontend Unit Tests | 80% | 70% |
| Frontend E2E Tests | 60% | 50% |
| Python Unit Tests | 85% | 80% |

#### Coverage Configuration

**Backend (JaCoCo)**:
```

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.10</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

```
</execution>
<execution>
<id>report</id>
<phase>test</phase>
<goals>
<goal>report</goal>
</goals>
</execution>
</executions>
</plugin>
```

```
**Frontend (Vitest)**:
```

```
// vitest.config.ts
export default defineConfig({
test: {
coverage: {
provider: 'v8',
reporter: ['text', 'json', 'html'],
statements: 80,
branches: 80,
functions: 80,
lines: 80
}
}
})
```

```
### Test Data Management

#### Test Data Fixtures

**Backend**:
```

@Component

public class TestDataFactory {

public static AIModel createTestModel() {

AIModel model = new AIModel();

model.setModelName("TestModel_v1");

model.setAlgorithm("RANDOM_FOREST");

model.setRmse(3.5);

model.setMae(2.8);

model.setStatus("COMPLETED");

return model;

}

public static Employee createTestEmployee() {

Employee employee = new Employee();

employee.setEmployeeId("EMP001");

employee.setName("John Doe");

employee.setDepartment("Operations");

employee.setRole("Analyst");

employee.setStatus("ACTIVE");

return employee;

}

}

```
**Frontend**:
```

```
// test-utils.ts
export const mockDashboardMetrics = {
totalEmployees: 245,
activeShifts: 18,
predictedDemand: 156,
capacityUtilization: 78.5
}

export const mockAIModels = [
{
id: 1,
modelName: 'RandomForest_v1',
algorithm: 'RANDOM_FOREST',
rmse: 3.5,
mae: 2.8,
status: 'COMPLETED'
}
]
```

```
### Continuous Testing
#### Pre-commit Hooks
```

```
#!/bin/bash
```

.git/hooks/pre-commit

Backend tests

cd Backend/workForceApplication

mvn test

Frontend tests

cd ../../Frontend/workforce-forecaasting-x

npm run test

Python tests

cd ../../workforce-forecasting-python

pytest

Check exit codes

if [$? -ne 0]; then

echo "Tests failed. Commit aborted."

exit 1

fi

```
### Testing Best Practices

#### 1. Test Independence

- Each test should be independent
- Tests should not rely on execution order
- Use proper setup and teardown

#### 2. Test Naming

- Use descriptive test names
- Follow pattern: `methodName_shouldExpectedResult_whenCondition`

#### 3. Test Organization

- Group related tests
- Use test suites for complex features
- Maintain clear test structure

#### 4. Mocking

- Mock external dependencies
- Use test doubles for slow operations
- Avoid over-mocking

#### 5. Test Maintenance

- Keep tests updated with code changes
- Remove obsolete tests
```

```
- Refactor tests for readability

---

## CI/CD Pipeline

### CI/CD Overview

The system implements a comprehensive CI/CD pipeline using GitHub Actions for continuous integration and con

### CI/CD Architecture
```

■■■

■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Code Push ■■■■■■■ Build & ■■■■■■■ Test ■ ■

■ ■ ■ ■ Compile ■ ■ Stage ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Trigger ■ ■ Docker ■ ■ Deploy ■ ■

■ ■ Detection ■ ■ Build ■ ■ Stage ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Branch ■ ■ Image Push ■ ■ Notify ■ ■

■ ■ Validation ■ ■ to Registry■ ■ Team ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■

■■■■■■■■■■■■■■■■■

```
push:
branches: [ main, develop ]
paths:
• 'Backend/**'
pull_request:
branches: [ main, develop ]
paths:
• 'Backend/**'

jobs:
build:
runs-on: ubuntu-latest

steps:
• name: Checkout code
uses: actions/checkout@v3

• name: Set up JDK 17
uses: actions/setup-java@v3
with:
java-version: '17'
distribution: 'temurin'
cache: maven

• name: Build with Maven
run: |
cd Backend/workForceApplication
mvn clean compile

• name: Run unit tests
```

```
run: |
cd Backend/workForceApplication
mvn test

• name: Generate test coverage
run: |
cd Backend/workForceApplication
mvn jacoco:report

• name: Upload coverage to Codecov
uses: codecov/codecov-action@v3
with:
file: Backend/workForceApplication/target/site/jacoco/jacoco.xml
flags: backend

• name: Build Docker image
run: |
cd Backend/workForceApplication
docker build -t workforce-backend:${{ github.sha }} .

• name: Log in to Docker Hub
uses: docker/login-action@v2
with:
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}

• name: Push Docker image
run: |
docker tag workforce-backend:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-backend:latest
```

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-backend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-backend:${{ github.sha }}

• name: Deploy to staging (main branch only)

if: github.ref == 'refs/heads/main'

run: |

Deployment script for staging environment

echo "Deploying to staging"

• name: Deploy to production (manual trigger)

if: github.ref == 'refs/heads/main' && github.event_name == 'workflow_dispatch'

run: |

Deployment script for production environment

echo "Deploying to production"

```
#### Frontend CI/CD Pipeline

**.github/workflows/frontend-ci.yml**:
```

name: Frontend CI/CD

on:

push:

branches: [main, develop]

paths:

• 'Frontend/**'

pull_request:

branches: [main, develop]

paths:

• 'Frontend/**'

```
jobs:
build:
runs-on: ubuntu-latest
steps:
• name: Checkout code
uses: actions/checkout@v3
• name: Set up Node.js
uses: actions/setup-node@v3
with:
node-version: '18'
cache: 'npm'
cache-dependency-path: Frontend/workforce-forecaasting-x/package-lock.json
• name: Install dependencies
run: |
cd Frontend/workforce-forecaasting-x
npm ci
• name: Run linter
run: |
cd Frontend/workforce-forecaasting-x
npm run lint
• name: Run unit tests
run: |
cd Frontend/workforce-forecaasting-x
npm run test:unit
```

```
• name: Generate coverage report
run: |
cd Frontend/workforce-forecaasting-x
npm run test:coverage
• name: Upload coverage to Codecov
uses: codecov/codecov-action@v3
with:
file: Frontend/workforce-forecaasting-x/coverage/coverage-final.json
flags: frontend
• name: Build application
run: |
cd Frontend/workforce-forecaasting-x
npm run build
• name: Build Docker image
run: |
cd Frontend/workforce-forecaasting-x
docker build -t workforce-frontend:${{ github.sha }} .
• name: Log in to Docker Hub
uses: docker/login-action@v2
with:
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
• name: Push Docker image
run: |
```

docker tag workforce-frontend:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:${{ github.sha }}

• name: Deploy to staging

if: github.ref == 'refs/heads/main'

run: |

Deployment script for staging environment

echo "Deploying frontend to staging"

```
#### Python ML Service CI/CD Pipeline

**.github/workflows/python-ci.yml**:
```

name: Python ML Service CI/CD

on:

push:

branches: [main, develop]

paths:

• 'workforce-forecasting-python/**'

pull_request:

branches: [main, develop]

paths:

• 'workforce-forecasting-python/**'

jobs:

build:

runs-on: ubuntu-latest

```
steps:
• name: Checkout code
uses: actions/checkout@v3
• name: Set up Python
uses: actions/setup-python@v4
with:
python-version: '3.9'
cache: 'pip'
cache-dependency-path: workforce-forecasting-python/requirements.txt
• name: Install dependencies
run: |
cd workforce-forecasting-python
pip install -r requirements.txt
pip install pytest pytest-cov
• name: Run linter
run: |
cd workforce-forecasting-python
flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
• name: Run unit tests
run: |
cd workforce-forecasting-python
pytest --cov=. --cov-report=xml --cov-report=term
• name: Upload coverage to Codecov
uses: codecov/codecov-action@v3
```

with:

file: workforce-forecasting-python/coverage.xml

flags: python

• name: Build Docker image

run: |

cd workforce-forecasting-python

docker build -t workforce-ml-service:${{ github.sha }} .

• name: Log in to Docker Hub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKER_USERNAME }}

password: ${{ secrets.DOCKER_PASSWORD }}

• name: Push Docker image

run: |

docker tag workforce-ml-service:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:${{ github.sha }}

```
### Pipeline Stages

#### 1. Build Stage

**Purpose**: Compile source code and generate artifacts

**Backend**:
```

mvn clean compile

```
**Frontend**:
```

npm run build

```
**Python**:
```

pip install -r requirements.txt

```
#### 2. Test Stage

**Purpose**: Execute automated tests

**Backend**:
```

mvn test

mvn jacoco:report

```
**Frontend**:
```

npm run test:unit

npm run test:coverage

```
**Python**:
```

pytest --cov=. --cov-report=xml

```
#### 3. Package Stage

**Purpose**: Create deployable artifacts

**Backend**:
```

mvn package

```
**Frontend**:
```

npm run build

```
**Python**:
```

python setup.py sdist bdist_wheel

```
#### 4. Deploy Stage

**Purpose**: Deploy to environments

**Staging Deployment**:
- Automatic on merge to main branch
- Runs integration tests
- Updates staging environment

**Production Deployment**:
- Manual trigger required
- Runs full regression tests
- Blue-green deployment
- Zero-downtime rollout

### Environment Configuration

#### Staging Environment

**Configuration**:
- Database: Staging MySQL instance
- Cache: Staging Redis instance
- Message Queue: Staging RabbitMQ instance
- Domain: staging.workforce-system.com

**Deployment Strategy**:
- Rolling updates
- Health checks before traffic routing
- Automatic rollback on failure
```

```
#### Production Environment

**Configuration**:
- Database: Production MySQL with replicas
- Cache: Production Redis cluster
- Message Queue: Production RabbitMQ cluster
- Domain: workforce-system.com

**Deployment Strategy**:
- Blue-green deployment
- Canary releases (10% traffic initially)
- Gradual traffic shift
- Full monitoring during rollout

### Deployment Strategies

#### Blue-Green Deployment
```

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

■ Blue ■ ■ Green ■

■ (Active) ■ ■ (Inactive) ■

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

■ ■

■■■■■■■■■■■■■■■■■■■■■■■

■

■■■■■■▼■■■■■■

■ Load ■

■ Balancer ■

■■■■■■■■■■■■■■

```
**Process**:
1. Deploy new version to Green environment
2. Run health checks on Green
3. Switch traffic to Green
4. Keep Blue for rollback
5. Decommission Blue after successful deployment

#### Canary Deployment
```

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

■ Stable ■ ■ Canary ■

■ (90% traffic)■ ■ (10% traffic)■

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

```
**Process**:
1. Deploy new version to subset of instances
2. Route 10% traffic to new version
3. Monitor metrics and errors
4. Gradually increase traffic if healthy
5. Rollback if issues detected

### Quality Gates

#### Pre-merge Checks

- All unit tests must pass
- Code coverage > 80%
- No critical security vulnerabilities
- Linter checks pass
- Build succeeds

#### Pre-deployment Checks

- All integration tests pass
- E2E tests pass
- Performance benchmarks met
- Security scan passes
- Manual approval required for production

### Rollback Strategy

#### Automatic Rollback Triggers
```

```
- Error rate > 5% for 5 minutes
- Response time p95 > 2 seconds
- Health check failures > 3 consecutive
- Critical security vulnerability detected

#### Manual Rollback Procedure
```

Revert to previous Docker image

docker pull workforce-backend:previous-tag

kubectl rollout undo deployment/backend

```
### Pipeline Monitoring

#### Key Metrics

| Metric | Threshold | Alert |
|--------|-----------|-------|
| Build Success Rate | > 95% | < 95% |
| Test Pass Rate | 100% | < 100% |
| Deployment Success Rate | > 98% | < 98% |
| Deployment Time | < 10 min | > 15 min |
| Rollback Rate | < 5% | > 10% |

### CI/CD Best Practices

#### 1. Pipeline as Code

- Version control all pipeline configurations
- Use reusable workflow templates
- Document pipeline steps

#### 2. Security

- Use secrets management
- Scan for vulnerabilities
- Sign artifacts
- Implement approval gates

#### 3. Performance
```

```
- Parallelize independent jobs
- Use caching for dependencies
- Optimize Docker layer caching

#### 4. Reliability

- Implement retry logic
- Use idempotent operations
- Validate before deployment
- Monitor pipeline health

---

## Deployment Architecture

### Deployment Overview

The system deployment architecture supports multiple deployment environments (development, staging, productio

### Deployment Architecture Diagram
```

■■■

■ ■ Kubernetes Cluster ■ ■

■ ■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■ ■

■ ■ ■ Frontend ■ ■ Backend ■ ■ Python ■ ■ ■

■ ■ ■ Pods ■ ■ Pods ■ ■ ML Pods ■ ■ ■

■ ■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■ ■

■ ■■■■■■■■■■■■■

[redacted]

```
### Docker Configuration

#### Backend Dockerfile

**Backend/workForceApplication/Dockerfile**:
```

FROM maven:3.8.6-eclipse-temurin-17 AS builder

WORKDIR /app

COPY pom.xml .

COPY src ./src

RUN mvn clean package -DskipTests

FROM eclipse-temurin:17-jre-alpine

WORKDIR /app

COPY --from=builder /app/target/*.jar app.jar

EXPOSE 5233

ENTRYPOINT ["java", "-jar", "app.jar"]

```
#### Frontend Dockerfile

**Frontend/workforce-forecaasting-x/Dockerfile**:
```

FROM node:18-alpine AS builder

WORKDIR /app

COPY package*.json ./

RUN npm ci

COPY . .

RUN npm run build

FROM nginx:alpine

COPY --from=builder /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```
**nginx.conf**:
```

server {

listen 80;

server_name localhost;

root /usr/share/nginx/html;

index index.html;

location / {

try_files $uri $uri/ /index.html;

}

location /api {

proxy_pass http://backend:5233;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

}

}

```
#### Python ML Service Dockerfile

**workforce-forecasting-python/Dockerfile**:
```

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "serviceFast.main:app", "--host", "0.0.0.0", "--port", "8000"]

```
### Kubernetes Deployment

#### Backend Deployment

**k8s/backend-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: backend

labels:

app: backend

spec:

replicas: 3

selector:

matchLabels:

app: backend

template:

metadata:

labels:

app: backend

spec:

containers:

• name: backend

image: workforce-backend:latest

ports:

• containerPort: 5233

env:

• name: SPRING_DATASOURCE_URL

valueFrom:

secretKeyRef:

name: db-secret

key: url

• name: SPRING_DATASOURCE_USERNAME

valueFrom:

secretKeyRef:

name: db-secret

key: username

• name: SPRING_DATASOURCE_PASSWORD

valueFrom:

secretKeyRef:

name: db-secret

key: password

resources:

requests:

memory: "512Mi"

cpu: "500m"

limits:

memory: "1Gi"

cpu: "1000m"

livenessProbe:

httpGet:

path: /actuator/health

port: 5233

initialDelaySeconds: 60

periodSeconds: 10

readinessProbe:

httpGet:

path: /actuator/health

port: 5233

initialDelaySeconds: 30

periodSeconds: 5

_

---

apiVersion: v1

kind: Service

metadata:

name: backend

spec:

selector:

app: backend

ports:

• protocol: TCP

port: 5233

targetPort: 5233

type: ClusterIP

```
#### Frontend Deployment

**k8s/frontend-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

labels:

app: frontend

spec:

replicas: 2

selector:

matchLabels:

app: frontend

template:

metadata:

labels:

app: frontend

spec:

containers:

• name: frontend

image: workforce-frontend:latest

ports:

• containerPort: 80

resources:

requests:

memory: "128Mi"

cpu: "100m"

limits:

memory: "256Mi"

cpu: "200m"

_

---

apiVersion: v1

kind: Service

metadata:

name: frontend

spec:

selector:

app: frontend

ports:

• protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

```
#### Python ML Service Deployment

**k8s/python-ml-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: python-ml

labels:

app: python-ml

spec:

replicas: 2

selector:

matchLabels:

app: python-ml

template:

metadata:

labels:

app: python-ml

spec:

containers:

• name: python-ml

image: workforce-ml-service:latest

ports:

• containerPort: 8000

resources:

requests:

memory: "1Gi"

cpu: "500m"

limits:

memory: "2Gi"

cpu: "2000m"

_

---

apiVersion: v1

kind: Service

metadata:

name: python-ml

spec:

selector:

app: python-ml

ports:

• protocol: TCP

port: 8000

targetPort: 8000

type: ClusterIP

```
### Infrastructure as Code (Terraform)

#### VPC and Networking

**terraform/vpc.tf**:
```

resource "aws_vpc" "workforce_vpc" {

cidr_block = "10.0.0.0/16"

enable_dns_hostnames = true

enable_dns_support = true

tags = {

Name = "workforce-vpc"

}

}

```
resource "aws_subnet" "public_subnet" {
count = 2
vpc_id = aws_vpc.workforce_vpc.id
cidr_block = "10.0.${count.index}.0/24"
availability_zone = data.aws_availability_zones.available.names[count.index]
map_public_ip_on_launch = true

tags = {
Name = "public-subnet-${count.index}"
}
}

resource "aws_subnet" "private_subnet" {
count = 2
vpc_id = aws_vpc.workforce_vpc.id
cidr_block = "10.0.${count.index + 10}.0/24"
availability_zone = data.aws_availability_zones.available.names[count.index]

tags = {
Name = "private-subnet-${count.index}"
}
}
```

```
#### EKS Cluster
**terraform/eks.tf**:
```

```
resource "aws_eks_cluster" "workforce_cluster" {
name = "workforce-cluster"
role_arn = aws_iam_role.eks_cluster.arn
```

```
version = "1.27"

vpc_config {

subnet_ids = aws_subnet.private_subnet[*].id

}

depends_on = [

aws_iam_role_policy_attachment.eks_cluster_policy

]

}

resource "aws_eks_node_group" "workforce_nodes" {

cluster_name = aws_eks_cluster.workforce_cluster.name

node_group_name = "workforce-nodes"

node_role_arn = aws_iam_role.eks_nodes.arn

subnet_ids = aws_subnet.private_subnet[*].id

scaling_config {

desired_size = 3

max_size = 5

min_size = 2

}

instance_types = ["t3.medium"]

}
```

```
#### RDS Database

**terraform/rds.tf**:
```

```
resource "aws_db_instance" "workforce_db" {
```

```
identifier = "workforce-db"

engine = "mysql"

engine_version = "8.0"

instance_class = "db.t3.medium"

allocated_storage = 20

max_allocated_storage = 100

storage_type = "gp2"

storage_encrypted = true

db_name = "workforce"

username = var.db_username

password = var.db_password

vpc_security_group_ids = [aws_security_group.db_sg.id]

db_subnet_group_name = aws_db_subnet_group.main.name

backup_retention_period = 30

multi_az = true

skip_final_snapshot = false

final_snapshot_identifier = "workforce-db-final-snapshot"

tags = {

Name = "workforce-database"

}

}
```

```
### Environment Configuration

#### Development Environment

**Infrastructure**:
- Local Docker Compose
- Local MySQL instance
- Local Redis instance
- No load balancer

**Configuration**:
```

docker-compose.yml

version: '3.8'

services:

backend:

build: ./Backend/workForceApplication

ports:

• "5233:5233"

environment:

• SPRING_PROFILES_ACTIVE=dev

• SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/workforce

depends_on:

• mysql

frontend:

build: ./Frontend/workforce-forecaasting-x

ports:

• "5173:80"

python-ml:

build: ./workforce-forecasting-python

ports:

• "8000:8000"

mysql:

image: mysql:8.0

environment:

• MYSQL_ROOT_PASSWORD=root

• MYSQL_DATABASE=workforce

volumes:

• mysql-data:/var/lib/mysql

redis:

image: redis:alpine

ports:

• "6379:6379"

volumes:

mysql-data:

```
#### Staging Environment

**Infrastructure**:
- AWS EKS cluster (small)
- RDS MySQL (multi-AZ)
- ElastiCache Redis
- Application Load Balancer

**Configuration**:
- 2 replicas per service
- Development database
- Staging-specific secrets

#### Production Environment

**Infrastructure**:
- AWS EKS cluster (large)
- RDS MySQL (multi-AZ with read replicas)
- ElastiCache Redis cluster
- Network Load Balancer
- CloudFront CDN

**Configuration**:
- 3+ replicas per service
- Production database with replicas
- Production secrets
- Enhanced monitoring

### Deployment Process
```

```
#### 1. Build and Push Images
```

Build images

docker build -t workforce-backend:latest ./Backend/workForceApplication

docker build -t workforce-frontend:latest ./Frontend/workforce-forecaasting-x

docker build -t workforce-ml-service:latest ./workforce-forecasting-python

Tag for registry

docker tag workforce-backend:latest registry.workforce.com/workforce-backend:latest

docker tag workforce-frontend:latest registry.workforce.com/workforce-frontend:latest

docker tag workforce-ml-service:latest registry.workforce.com/workforce-ml-service:latest

Push to registry

docker push registry.workforce.com/workforce-backend:latest

docker push registry.workforce.com/workforce-frontend:latest

docker push registry.workforce.com/workforce-ml-service:latest

```
#### 2. Apply Kubernetes Manifests
```

Apply deployments

kubectl apply -f k8s/backend-deployment.yaml

kubectl apply -f k8s/frontend-deployment.yaml

kubectl apply -f k8s/python-ml-deployment.yaml

Apply services

kubectl apply -f k8s/backend-service.yaml

kubectl apply -f k8s/frontend-service.yaml

kubectl apply -f k8s/python-ml-service.yaml

Apply secrets

kubectl apply -f k8s/secrets.yaml

```
#### 3. Verify Deployment
```

Check pod status

kubectl get pods

Check services

kubectl get services

Check logs

kubectl logs -f deployment/backend

Port forward for testing

kubectl port-forward service/backend 5233:5233

```
### Monitoring and Logging in Production

#### Centralized Logging

- ELK Stack for log aggregation
- Logstash for log processing
- Kibana for log visualization
- Filebeat for log shipping

#### Metrics Collection

- Prometheus for metrics collection
- Grafana for visualization
- Alertmanager for alerting

#### Distributed Tracing

- Jaeger for distributed tracing
- OpenTelemetry for instrumentation

### Disaster Recovery

#### Backup Strategy

- Automated daily backups
- Point-in-time recovery
- Cross-region replication
- 30-day retention
```

```
#### Recovery Procedures

1. **Database Recovery**
   - Restore from RDS snapshot
   - Replay binary logs
   - Verify data integrity

2. **Application Recovery**
   - Deploy previous Docker image
   - Restore configuration
   - Verify functionality

3. **Infrastructure Recovery**
   - Use Terraform to rebuild
   - Restore from infrastructure backups
   - Validate networking

### Security in Production

#### Network Security

- VPC with private subnets
- Security groups with least privilege
- Network ACLs
- WAF for web application firewall

#### Secrets Management

- AWS Secrets Manager
- Kubernetes secrets
```

```
- Environment-specific secrets
- Rotation policies

#### Compliance

- SOC 2 Type II compliance
- GDPR compliance
- Regular security audits
- Penetration testing

### Cost Optimization

#### Resource Optimization

- Right-sizing instances
- Auto-scaling policies
- Spot instances for non-critical workloads
- Reserved instances for baseline load

#### Storage Optimization

- S3 lifecycle policies
- EBS volume optimization
- Database storage optimization
- Log retention policies

### Deployment Checklist

#### Pre-Deployment
```

```
- [ ] All tests pass
- [ ] Code coverage meets threshold
- [ ] Security scan passes
- [ ] Performance benchmarks met
- [ ] Documentation updated
- [ ] Rollback plan prepared

#### During Deployment

- [ ] Health checks pass
- [ ] Metrics within normal range
- [ ] No errors in logs
- [ ] Traffic routing successful
- [ ] Monitoring active

#### Post-Deployment

- [ ] Verify functionality
- [ ] Monitor for 30 minutes
- [ ] Check error rates
- [ ] Validate performance
- [ ] Update runbook
- [ ] Notify team

---

## Configuration

### Backend Configuration (application.properties)
```

Server Configuration

server.port=5233

Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/workforce

spring.datasource.username=root

spring.datasource.password=your_password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

JPA Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=false

Python API Configuration

fastapi.base-url=http://localhost:8000

python.api.url=http://localhost:8000/train

File Upload Configuration

spring.servlet.multipart.max-file-size=100MB

spring.servlet.multipart.max-request-size=100MB

JWT Configuration

jwt.secret=your_jwt_secret_key

jwt.expiration=86400000

```
### Frontend Configuration (apiClient.ts)
```

const API_BASE_URL = 'http://localhost:5233/api'

Security

Authentication Flow

1. User submits credentials to `/api/auth/login`
2. Backend validates credentials
3. JWT token generated and returned
4. Token stored in localStorage
5. Subsequent requests include token in Authorization header
6. JwtAuthenticationFilter validates token on each request

Role-Based Access Control

- **ADMIN**: Full access to all features
- **MANAGER**: Access to dashboard, forecasting, reports
- **VIEWER**: Read-only access to dashboard and reports

Security Configuration

- CSRF protection disabled for API
- Stateless session management
- JWT-based authentication
- CORS enabled for frontend origin
- Password hashing with BCrypt

```
## Development Guidelines

### Code Style

- **Java**: Follow Google Java Style Guide
- **TypeScript**: ESLint with Prettier
- **Python**: PEP 8 guidelines
- **Vue**: Vue.js Style Guide

### Git Workflow

1. Create feature branch from main
2. Make changes and commit
3. Push to remote
4. Create pull request
5. Code review
6. Merge to main

### Testing

- Backend: JUnit 5, Mockito
- Frontend: Vitest, Vue Test Utils
- Python: pytest, unittest

---

## Deployment

### Backend Deployment
```

Build JAR file

mvn clean package

Run JAR

java -jar target/workForceApplication.jar

```
### Frontend Deployment
```

Build for production

npm run build

Deploy dist/ folder to web server

```
### Python Service Deployment
```

Use Gunicorn for production

gunicorn serviceFast.main:app -w 4 -k uvicorn.workers.UvicornWorker --_bind 0.0.0.0:8000

```
---

## Troubleshooting

### Common Issues

1. **Database Connection Error**
   - Verify MySQL is running
   - Check credentials in application.properties
   - Ensure database exists

2. **CORS Error**
   - Verify CorsConfig allows frontend origin
   - Check frontend API base URL

3. **Python Service Not Responding**
   - Verify FastAPI is running on port 8000
   - Check Python dependencies are installed
   - Review service logs

4. **JWT Token Expired**
   - Token expires after 24 hours (configurable)
   - User must re-login

---

## Future Enhancements

- [ ] Real-time WebSocket updates
```

Workforce Forecasting System

Comprehensive Technical Documentation

A complete technical guide to the workforce forecasting platform

Workforce Forecasting System - Comprehensive Technical Documentation

Abstract

The Workforce Forecasting System is an enterprise-grade intelligent workforce management platform that leverages advanced machine learning algorithms to predict workforce demand, optimize shift scheduling, and provide real-time analytics for organizational planning. This system implements a microservices architecture integrating a modern reactive web application with state-of-the-art predictive models to enable data-driven decision-making in human resource management. The platform supports multiple machine learning models including Linear Regression, Random Forest, XGBoost, and LSTM networks, with automated model selection based on performance metrics.

_

Table of Contents

1. [Project Overview](#project-overview)

2. [System Architecture](#system-architecture)

3. [Technology Stack](#technology-stack)

4. [Project Structure](#project-structure)

5. [Database Design](#database-design)

6. [API Specifications](#api-specifications)

7. [Machine Learning Implementation](#machine-learning-implementation)

8. [Security Architecture](#security-architecture)

9. [Performance Optimization](#performance-optimization)

10. [Scalability & High Availability](#scalability--high-availability)

11. [Monitoring & Observability](#monitoring--observability)

12. [Testing Strategy](#testing-strategy)

13. [CI/CD Pipeline](#cicd-pipeline)

14. [Deployment Architecture](#deployment-architecture)

15. [Installation & Setup](#installation--setup)

16. [Configuration Management](#configuration-management)

17. [Troubleshooting Guide](#troubleshooting-guide)

18. [Future Roadmap](#future-roadmap)

_

Project Overview

Problem Statement

Organizations face significant challenges in workforce planning due to:

• Unpredictable demand fluctuations

• Inefficient shift scheduling leading to overstaffing/understaffing

• Lack of data-driven decision-making tools

• Manual processes prone to human error

• Inability to forecast workforce requirements accurately

Solution Overview

The Workforce Forecasting System addresses these challenges through:

• **Predictive Analytics**: ML models forecast workforce demand with high accuracy

• **Real-time Monitoring**: Live dashboards track workforce metrics

• **Automated Scheduling**: AI-powered shift optimization

• **Comparative Analysis**: Multiple model evaluation for best fit

• **Comprehensive Reporting**: PDF reports for stakeholder communication

Key Features

• **Real-time Analytics Dashboard**

• Live workforce metrics monitoring

• Attendance pattern analysis

• Capacity utilization tracking

• KPI cards with drill-down capabilities

• Interactive charts and visualizations

• **AI-Powered Forecasting**

• Multiple ML models (Linear Regression, Random Forest, XGBoost, LSTM)

• Automated model selection based on performance

• Hyperparameter optimization

• Time-series forecasting capabilities

• Confidence intervals for predictions

• **Smart Scheduling**

• Shift optimization interface

• Manual scheduling adjustments

• Shift pattern analysis

• **Model Comparison**

• Comparative evaluation framework

• Performance metrics visualization

• Model benchmarking

• Algorithm selection interface

• **Performance Monitoring**

• Real-time system health metrics

• Model performance tracking

• Microservice health monitoring

• KPI card tracking

• Performance metrics dashboard

• **Interactive Reports**

• PDF generation with iText 7

• Report generation interface

• Analytics reporting

• Data export capabilities

• **Role-Based Access Control**

• Admin: Full system access

• JWT-based authentication

• Secure API endpoints

• User management

• **Modern UI/UX**

• Responsive design with Vue.js 3

• PrimeVue UI components

• Interactive charts with Chart.js

• Clean, intuitive interface

• Real-time data updates

System Requirements

Functional Requirements

ID	Requirement	Priority
FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low

FR-1	User authentication with JWT	High
FR-10	Shift optimization	Low

FR-1	User authentication with JWT	High
FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-2	Role-based access control	High
FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium

FR-2	Role-based access control	High
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-3	Real-time dashboard with KPIs	High
FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-4	ML model training interface	High
FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-5	Prediction generation	High
FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-6	Model comparison	Medium
FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-7	PDF report generation	Medium
FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-8	Settings management	Medium
FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-9	Employee management	Low
FR-10	Shift optimization	Low

FR-10	Shift optimization	Low

Non-Functional Requirements

ID	Requirement	Target
NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-1	Response time	< 2s for 95% of requests
NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-2	Availability	99.9% uptime
NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-3	Throughput	1000 requests/second
NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-4	Scalability	Horizontal scaling support
NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-5	Security	OWASP Top 10 compliance
NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-6	Data retention	Configurable (default 90 days)
NFR-7	Backup	Daily automated backups

NFR-7	Backup	Daily automated backups

_

System Architecture

High-Level Architecture

The system implements a microservices architecture following the principles of Domain-Driven Design (DDD) and Event-Driven Architecture (EDA). The architecture is designed for scalability, maintainability, and fault isolation.

```
■■■
```

```
■■■
```

```
User Upload → Frontend → Backend → Python ML Service
     ■            ■          ■            ■
     ■            ■          ■            ■■ Data Preprocessing
     ■            ■          ■            ■■ Model Selection
     ■            ■          ■            ■■ Prediction Generation
     ■            ■          ■            ■■ Result Serialization
     ■            ■          ■
     ■            ■          ■■ Store in Database
     ■            ■          ■■ Return to Frontend
     ■            ■
     ■            ■■ Display Results
     ■■ View Dashboard
```

Authentication Flow

```
User Credentials → Frontend → Backend (Auth Controller)
     ■                  ■            ■
     ■                  ■            ■■ Validate Credentials
     ■                  ■            ■■ Generate JWT Token
     ■                  ■            ■■ Return Token + User Info
     ■                  ■
     ■                  ■■ Store Token in localStorage
     ■                  ■■ Redirect to Dashboard
     ■
     ■■ Subsequent Requests (JWT in Header)
```

Component Overview

1. Frontend Application (Vue.js 3)

Responsibilities:

• User interface rendering

• Client-side routing

• State management

• API communication

• Data visualization

Key Technologies:

• Vue.js 3 Composition API

• TypeScript for type safety

• PrimeVue for UI components

• Chart.js for data visualization

• Axios for HTTP requests

• Pinia for state management (future)

Design Patterns:

• Composition API for reusable logic

• Provider/Inject for dependency injection

• Suspense for async components

• Teleport for portal rendering

2. Backend Application (Spring Boot)

Responsibilities:

• REST API endpoints

• Business logic implementation

• Data persistence

• Authentication/authorization

• External service integration

• PDF generation

Key Technologies:

• Spring Boot 3.x

• Spring Security with JWT

• Spring Data JPA

• Hibernate ORM

• iText 7 for PDF generation

• Maven for dependency management

Design Patterns:

• Repository Pattern for data access

• Service Layer Pattern for business logic

• DTO Pattern for data transfer

• Builder Pattern for object construction

• Strategy Pattern for algorithm selection

3. Python ML Service (FastAPI)

Responsibilities:

• Model training

• Prediction generation

• Model evaluation

• Feature engineering

• Data preprocessing

Key Technologies:

• FastAPI for REST API

• scikit-learn for traditional ML

• TensorFlow/Keras for deep learning

• XGBoost for gradient boosting

• pandas for data manipulation

• numpy for numerical computing

Design Patterns:

• Factory Pattern for model creation

• Strategy Pattern for algorithm selection

• Template Method Pattern for training pipeline

• Observer Pattern for progress tracking

4. Database (MySQL)

Responsibilities:

• Persistent data storage

• Transaction management

• Data integrity

• Query optimization

• Backup and recovery

Key Features:

• InnoDB for transactional support

• Indexing for query performance

• Foreign key constraints

• Stored procedures (future)

• Database triggers (future)

Security Architecture

Authentication Layer

```
■■■
```

```
■■■
```

• **Framework**: Vue.js 3 with Composition API

• **Language**: JavaScript/TypeScript

• **UI Components**: PrimeVue

• **Charts**: Chart.js

• **HTTP Client**: Axios

• **State Management**: Vue Reactive API

• **Routing**: Vue Router

• **Build Tool**: Vite

• **Implemented Pages**: Dashboard, AI Models, Forecasting, Analytics, Benchmark, Capacity Planning, Employees, Monitor, Notifications, Reports, Settings, Shift Optimization

Backend

• **Framework**: Spring Boot 3.x

• **Language**: Java 17+

• **ORM**: Hibernate/JPA

• **Security**: Spring Security with JWT

• **Database**: MySQL

• **PDF Generation**: iText 7

• **Build Tool**: Gradle

• **Implemented Controllers**: Dashboard, Authentication, Training, Prediction, Analytics, Benchmark, Capacity, Employee, Forecasting, KPI Card, Microservice Health, Monitor, Notification, Optimization, Report, Settings, Shift Optimization

Python ML Service

• **Framework**: FastAPI

• **ML Libraries**: scikit-learn, TensorFlow/Keras, XGBoost

• **Data Processing**: pandas, numpy

- **Visualization**: matplotlib, seaborn
- **API**: Uvicorn server

_

Project Structure

Frontend Structure

```
Frontend/workforce-forecaasting-x/
■■■ src/
■   ■■■ auth/                          # Authentication
■   ■   ■■■ loginComponent.vue
■   ■   ■■■ authService.ts
■   ■   ■■■ authService.ts
■   ■■■ component/                     # Shared components
■   ■   ■■■ topMenuBarComponent.vue
■   ■■■ homeSetting/                   # Home settings page
■   ■   ■■■ homeSettingComponent.vue
■   ■   ■■■ homeSettingAPI.ts
■   ■■■ settings/                     # Main application pages
■   ■   ■■■ aiModels/                 # AI model management
■   ■   ■   ■■■ aiModelsComponent.vue
■   ■   ■   ■■■ aiModelAPI.ts
■   ■   ■   ■■■ aiModel.css
■   ■   ■■■ analyze/                  # Analytics
■   ■   ■   ■■■ analyticsComponent.vue
■   ■   ■   ■■■ analyticsAPI.ts
■   ■   ■■■ benchmark/                # Benchmarking
■   ■   ■   ■■■ benchmarkComponent.vue
■   ■   ■   ■■■ benchmarkAPI.ts
■   ■   ■■■ capacityPlanning/         # Capacity planning
■   ■   ■   ■■■ capacityPlanningComponent.vue
■   ■   ■   ■■■ capacityPlanningAPI.ts
■   ■   ■■■ dashboard/                # Main dashboard
■   ■   ■   ■■■ dashboardComponent.vue
■   ■   ■   ■■■ dashboardAPI.ts
■   ■   ■   ■■■ dashboardComponent.css
■   ■   ■■■ employees/                # Employee management
```

```
■   ■   ■   ■■■ employeesComponent.vue
■   ■   ■   ■■■ employeesAPI.ts
■   ■   ■■■ forecasting/              # Forecasting interface
■   ■   ■   ■■■ foreCastingComponent.vue
■   ■   ■   ■■■ foreCastingAPI.ts
■   ■   ■■■ monitor/                 # System monitoring
■   ■   ■   ■■■ monitorComponent.vue
■   ■   ■   ■■■ monitorAPI.ts
■   ■   ■■■ notification/             # Notifications
■   ■   ■   ■■■ notificationComponent.vue
■   ■   ■   ■■■ notificationAPI.ts
■   ■   ■■■ reports/                 # Reports generation
■   ■   ■   ■■■ reportComponent.vue
■   ■   ■   ■■■ reportAPI.ts
■   ■   ■■■ settings/                # Application settings
■   ■   ■   ■■■ setttingsComponent.vue
■   ■   ■   ■■■ settingsService.ts
■   ■   ■   ■■■ settingsComponent.css
■   ■   ■■■ shiftOptimization/        # Shift optimization
■   ■       ■■■ shiftOptimizationComponent.vue
■   ■       ■■■ shiftOptimizationAPI.ts
■   ■■■ services/                     # API services
■   ■   ■■■ apiClient.ts              # Axios configuration
■   ■■■ sideBar/                      # Navigation sidebar
■   ■   ■■■ sideBarComponent.vue
■   ■■■ router/                       # Route configuration
■   ■   ■■■ index.ts
■   ■   ■■■ routes.ts
■   ■■■ App.vue                      # Root component
■   ■■■ main.ts                      # Application entry
```

```
■■■ package.json
■■■ tsconfig.json
■■■ vite.config.ts
```

Backend Structure

```
Backend/workForceApplication/
■■■ src/main/java/com/boostphysioclinic/workforceapplication/
■   ■■■ client/                     # External API clients
■   ■   ■■■ RestClientConfig.java
■   ■■■ config/                     # Configuration
■   ■   ■■■ CorsConfig.java
■   ■   ■■■ SecurityConfig.java
■   ■■■ controller/                 # REST Controllers
■   ■   ■■■ AIModelController.java
■   ■   ■■■ DashboardController.java
■   ■   ■■■ PredictionController.java
■   ■   ■■■ SettingsController.java
■   ■   ■■■ ...
■   ■■■ dto/                        # Data Transfer Objects
■   ■   ■■■ entity/                  # Entity DTOs
■   ■   ■   ■■■ AIModel.java
■   ■   ■   ■■■ Employee.java
■   ■   ■   ■■■ Settings.java
■   ■   ■   ■■■ PredictionResult.java
■   ■   ■   ■■■ ...
■   ■   ■■■ PredictionResponse.java
■   ■   ■■■ PredictionResultDTO.java
■   ■   ■■■ PredictionListDeserializer.java
■   ■■■ Repository/                   # JPA Repositories
■   ■   ■■■ AIModelRepository.java
■   ■   ■■■ EmployeeRepository.java
■   ■   ■■■ SettingsRepository.java
■   ■   ■■■ PredictionResultRepository.java
■   ■   ■■■ ...
■   ■■■ security/                   # Security
```

```
■   ■   ■■■ JwtAuthenticationFilter.java
■   ■   ■■■ JwtService.java
■   ■■■ service/                      # Business Logic
■   ■   ■■■ AIModelService.java
■   ■   ■■■ CLPredictionService.java
■   ■   ■■■ PdfGenerationService.java
■   ■   ■■■ PredictionService.java
■   ■   ■■■ ...
■   ■■■ WorkForceApplication.java    # Main application
■■■ src/main/resources/
■   ■■■ application.properties       # Configuration
■   ■■■ schema.sql                   # Database schema
■■■ build.gradle                     # Gradle dependencies
```

Python ML Service Structure

```
workforce-forecasting-python/
■■■ dataset/                        # Data storage
■■■ evaluation/                     # Model evaluation
■   ■■■ evaluation.py
■■■ models/                         # ML Models
■   ■■■ linear_regression_model.py
■   ■■■ random_forest_model.py
■   ■■■ xgboost_model.py
■   ■■■ lstm_model.py
■■■ prediction/                     # Prediction logic
■   ■■■ __init__.py
■■■ preprocessing/                  # Data preprocessing
■   ■■■ preprocessing.py
■   ■■■ lstm_preprocessing.py
■■■ serviceFast/                    # FastAPI service
■   ■■■ __init__.py
■   ■■■ main.py
■■■ training/                       # Model training
■   ■■■ model_saver.py
■   ■■■ model_selector.py
■   ■■■ model_trainer.py
■   ■■■ prediction_result_saver.py
■   ■■■ training_service.py
■■■ visualization/                  # Visualization
■   ■■■ performance_comparison.py
■■■ saved_models/                   # Trained models storage
■■■ results/                        # Prediction results
■■■ uploads/                        # File uploads
■■■ main.py                         # Entry point
■■■ requirements.txt                 # Python dependencies
```

```
■■■ serviceFast/                     # FastAPI application
```

_

Database Design

Database Schema Overview

The database follows a normalized relational schema designed for data integrity, query performance, and scalability. The schema is organized into logical domains: authentication, workforce management, machine learning, and system configuration.

Entity-Relationship Diagram

```
■■■
```

```
■ animations   ■       ■■■■■■■■■■■■■■■■■■■■■
■ critical_... ■
■ shift_rec... ■       ■■■■■■■■■■■■■■■■■■■■■
■ system_mon.. ■       ■  employees       ■
■ email_digest ■       ■■■■■■■■■■■■■■■■■■■■■
■ model_name   ■       ■ id (PK)          ■
■ refresh_int. ■       ■ employee_id      ■
■ api_url      ■       ■ name             ■
■ ml_url       ■       ■ department       ■
■ data_retent. ■       ■ role             ■
■ active_model ■       ■ email            ■
■ model_version■       ■ hire_date        ■
■ training_freq■       ■ status           ■
■ conf_thresh  ■       ■■■■■■■■■■■■■■■■■■■■■
■ auto_retrain ■
■ monitoring   ■
■ feature_imp. ■
■ created_at   ■
■ updated_at   ■
■■■■■■■■■■■■■■■■■
```

Detailed Table Definitions

Users Table

```
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL COMMENT 'BCrypt hashed password',
    role VARCHAR(50) NOT NULL COMMENT 'ADMIN, MANAGER, VIEWER',
    email VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP NULL,
    is_active BOOLEAN DEFAULT TRUE,
    INDEX idx_username (username),
    INDEX idx_role (role)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Indexes:

- `idx_username`: For login queries
- `idx_role`: For role-based filtering

Constraints:

- `username` must be unique
- `role` must be one of: ADMIN, MANAGER, VIEWER
- `password` stored as BCrypt hash

AI Models Table

```
CREATE TABLE ai_models (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    model_name VARCHAR(255) NOT NULL,
    algorithm VARCHAR(100) NOT NULL COMMENT 'LINEAR_REGRESSION, RANDOM_FOREST, XGBOOST, LSTM',
    version VARCHAR(50),
    rmse DOUBLE COMMENT 'Root Mean Square Error',
    mae DOUBLE COMMENT 'Mean Absolute Error',
    mape DOUBLE COMMENT 'Mean Absolute Percentage Error',
    r_squared DOUBLE COMMENT 'R-Squared Score',
    training_time BIGINT COMMENT 'Training time in milliseconds',
    status VARCHAR(50) DEFAULT 'PENDING' COMMENT 'PENDING, TRAINING, COMPLETED, FAILED',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    trained_at TIMESTAMP NULL,
    model_path VARCHAR(500) COMMENT 'Path to saved model file',
    hyperparameters JSON COMMENT 'Model hyperparameters as JSON',
    feature_importance JSON COMMENT 'Feature importance scores',
    INDEX idx_algorithm (algorithm),
    INDEX idx_status (status),
    INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Indexes:

- `idx_algorithm`: For filtering by algorithm type
- `idx_status`: For tracking training status
- `idx_created_at`: For chronological queries

JSON Fields:

- `hyperparameters`: Stores model-specific parameters
- `feature_importance`: Stores feature importance scores

Settings Table

```sql
CREATE TABLE settings (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,

    -- Profile Settings
    full_name VARCHAR(255),
    email VARCHAR(255),
    role VARCHAR(100),
    department VARCHAR(100),

    -- Appearance Settings
    dark_mode BOOLEAN DEFAULT FALSE,
    compact_view BOOLEAN DEFAULT FALSE,
    animations BOOLEAN DEFAULT TRUE,

    -- Notification Settings
    critical_alerts BOOLEAN DEFAULT TRUE,
    shift_recommendations BOOLEAN DEFAULT TRUE,
    system_monitoring BOOLEAN DEFAULT TRUE,
    email_digest BOOLEAN DEFAULT FALSE,

    -- Config Settings
    model_name VARCHAR(255),
    refresh_interval VARCHAR(50) DEFAULT '5m',
    api_url VARCHAR(500),
    ml_url VARCHAR(500),
    data_retention INT DEFAULT 90,

    -- AI Model Settings
    active_model VARCHAR(255),
    model_version VARCHAR(50),
```

```sql
    training_frequency VARCHAR(50) DEFAULT 'weekly',
    confidence_threshold INT DEFAULT 80,
    auto_retrain BOOLEAN DEFAULT FALSE,
    monitoring_enabled BOOLEAN DEFAULT TRUE,
    feature_importance BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_updated_at (updated_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Triggers:

- `updated_at` automatically updates on any row modification

Prediction Runs Table

```sql
CREATE TABLE prediction_runs (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    model_name VARCHAR(255) NOT NULL,
    model_id BIGINT,
    total_records INT,
    average_prediction DOUBLE,
    maximum_prediction DOUBLE,
    minimum_prediction DOUBLE,
    confidence_level DOUBLE DEFAULT 0.95,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP NULL,
    status VARCHAR(50) DEFAULT 'RUNNING' COMMENT 'RUNNING, COMPLETED, FAILED',
    error_message TEXT,
    metadata JSON COMMENT 'Additional run metadata',
    FOREIGN KEY (model_id) REFERENCES ai_models(id) ON DELETE SET NULL,
    INDEX idx_created_at (created_at),
    INDEX idx_status (status),
    INDEX idx_model_id (model_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Foreign Key:

- `model_id` references `ai_models(id)` with SET NULL on delete

Prediction Results Table```sql

CREATE TABLE prediction_results (

id BIGINT PRIMARY KEY AUTO_INCREMENT,

prediction_run_id BIGINT NOT NULL,

attendance_date DATE NOT NULL,

department VARCHAR(255),

actual_demand DOUBLE,

predicted_demand DOUBLE,

confidence_interval_lower DOUBLE,

confidence_interval_upper DOUBLE,

error_margin DOUBLE,

is_anomaly BOOLEAN DEFAULT FALSE,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (prediction_run_id) REFERENCES prediction_runs(id) ON DELETE CASCADE,

INDEX idx_run_id (prediction_run_id),

INDEX idx_attendance_date (attendance_date),

INDEX idx_department (department),

INDEX idx_is_anomaly (is_anomaly)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

```
**Foreign Key:**
- `prediction_run_id` references `prediction_runs(id)` with CASCADE delete

**Indexes:**
- `idx_run_id`: For fetching results by run
- `idx_attendance_date`: For time-series queries
- `idx_department`: For department-specific queries
- `idx_is_anomaly`: For anomaly detection queries

#### Employees Table
```

CREATE TABLE employees (

id BIGINT PRIMARY KEY AUTO_INCREMENT,

employee_id VARCHAR(50) UNIQUE NOT NULL,

name VARCHAR(255) NOT NULL,

department VARCHAR(100),

role VARCHAR(100),

email VARCHAR(255),

hire_date DATE,

status VARCHAR(50) DEFAULT 'ACTIVE' COMMENT 'ACTIVE, INACTIVE, ON_LEAVE',

skills JSON COMMENT 'Employee skills as JSON array',

availability JSON COMMENT 'Availability schedule',

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

INDEX idx_employee_id (employee_id),

INDEX idx_department (department),

INDEX idx_status (status)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

```
**JSON Fields:**
- `skills`: Array of employee skills
- `availability`: Weekly availability schedule

### Database Optimization Strategies

#### Indexing Strategy

1. **Primary Indexes**: All primary keys are auto-increment BIGINT
2. **Foreign Key Indexes**: All foreign keys have indexes for join performance
3. **Query Indexes**: Indexes on frequently queried columns
4. **Composite Indexes**: For multi-column queries (future)

#### Query Optimization

1. **Connection Pooling**: HikariCP with optimal pool size
2. **Query Caching**: Second-level Hibernate cache (future)
3. **Read Replicas**: For read-heavy workloads (future)
4. **Partitioning**: By date for large tables (future)

#### Data Retention Policy

- **Prediction Results**: Retained for 90 days (configurable)
- **Prediction Runs**: Retained for 1 year
- **AI Models**: All versions retained
- **Settings**: Single active record per user

### Backup Strategy
```

```
#### Backup Types

1. **Full Backups**: Daily at 2:00 AM
2. **Incremental Backups**: Every 4 hours
3. **Binary Logs**: Continuous for point-in-time recovery

#### Retention Policy

- Daily backups: 7 days
- Weekly backups: 4 weeks
- Monthly backups: 12 months

---

## API Specifications

### API Overview

The REST API follows RESTful principles with resource-based URLs, standard HTTP methods, and appropriate sta

### Base URL

- **Development**: `http://localhost:5233/api`
- **Production**: `https://api.workforce-system.com/api`

### Authentication

All endpoints except `/api/auth/login` require JWT authentication in the Authorization header:
```

Authorization: Bearer <jwt_token>

```
### Response Format

#### Success Response
```

{

"success": true,

"data": { ... },

"message": "Operation successful",

"timestamp": "2026-08-19T10:30:00Z"

}

```
#### Error Response
```

{

"success": false,

"error": {

"code": "ERROR_CODE",

"message": "Error description",

"details": { ... }

},

"timestamp": "2026-08-19T10:30:00Z"

}

```
### HTTP Status Codes

| Code | Description |
|------|-------------|
| 200 | OK - Request successful |
| 201 | Created - Resource created successfully |
| 204 | No Content - Successful with no return data |
| 400 | Bad Request - Invalid request parameters |
| 401 | Unauthorized - Authentication required/failed |
| 403 | Forbidden - Insufficient permissions |
| 404 | Not Found - Resource not found |
| 409 | Conflict - Resource conflict |
| 500 | Internal Server Error - Server error |
| 503 | Service Unavailable - Service temporarily down |

---

### Authentication Endpoints

#### POST /api/auth/login

Authenticates a user and returns a JWT token.

**Request Headers:**
```

Content-Type: application/json

```
**Request Body:**
```


{

"username": "admin",

"password": "admin123"

}


```
**Response (200 OK):**
```


{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"tokenType": "Bearer",

"expiresIn": 86400000,

"user": {

"id": 1,

"username": "admin",

"role": "ADMIN",

"email": "admin@workforce.com"

}

},

"message": "Login successful"

}


```
**Error Response (401 Unauthorized):**
```

{

"success": false,

"error": {

"code": "INVALID_CREDENTIALS",

"message": "Invalid username or password"

}

}

```
#### POST /api/auth/logout

Invalidates the current JWT token (Not Implemented).

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"message": "Logout successful"

}

```
#### POST /api/auth/refresh

Refreshes an expired JWT token (Not Implemented).

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresIn": 86400000

}

}

```
---

### Dashboard Endpoints

#### GET /api/dashboard/predictions

Fetches the latest prediction results for the dashboard.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| limit | integer | No | Maximum number of results (default: 50) |
| offset | integer | No | Offset for pagination (default: 0) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"model": "RandomForest_v1",

"total_records": 365,

"average_prediction": 45.5,

"maximum_prediction": 78.2,

"minimum_prediction": 12.3,

"results": [

{

"attendanceDate": "2026-08-19",

"department": "Operations",

"actualDemand": 42.0,

"predictedDemand": 44.5

}

]

}

}

```
#### GET /api/dashboard/metrics

Retrieves KPI metrics for the dashboard.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| period | string | No | Time period: today, week, month, year (default: today) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"totalEmployees": 245,

"activeShifts": 18,

"predictedDemand": 156,

"capacityUtilization": 78.5,

"attendanceRate": 92.3,

"productivityScore": 87.2

}

}

```
#### GET /api/dashboard/charts

Fetches chart data for dashboard visualizations.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| chartType | string | Yes | Type of chart: line, bar, pie, radar |
| period | string | No | Time period (default: week) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"labels": ["Mon", "Tue", "Wed", "Thu", "Fri"],

"datasets": [

{

"label": "Actual Demand",

"data": [45, 52, 48, 55, 50],

"backgroundColor": "rgba(59, 130, 246, 0.5)"

},

{

"label": "Predicted Demand",

"data": [47, 50, 50, 53, 48],

"backgroundColor": "rgba(139, 92, 246, 0.5)"

}

]

}

}

```
---

### AI Models Endpoints

#### GET /api/ai-models

Retrieves all AI models with their performance metrics.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| algorithm | string | No | Filter by algorithm type |
| status | string | No | Filter by status |
| sortBy | string | No | Sort field: created_at, rmse, mae (default: created_at) |
| sortOrder | string | No | Sort order: asc, desc (default: desc) |

**Response (200 OK):**
```

{

"success": true,

"data": [

{

"id": 1,

"modelName": "RandomForest_v1",

"algorithm": "RANDOM_FOREST",

"version": "1.0",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92,

"trainingTime": 12500,

"status": "COMPLETED",

"createdAt": "2026-08-15T10:00:00Z",

"trainedAt": "2026-08-15T10:03:25Z"

}

],

"pagination": {

"total": 10,

"page": 1,

"pageSize": 10

}

}

```
#### GET /api/ai-models/{id}

Retrieves details of a specific AI model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Model ID |

**Response (200 OK):**
```


{

"success": true,

"data": {

"id": 1,

"modelName": "RandomForest_v1",

"algorithm": "RANDOM_FOREST",

"version": "1.0",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92,

"trainingTime": 12500,

"status": "COMPLETED",

"hyperparameters": {

"n_estimators": 100,

"max_depth": 10,

"min_samples_split": 2

},

"featureImportance": {

"attendance_history": 0.35,

"seasonal_factors": 0.25,

"holidays": 0.15,

"weather": 0.10,

"other": 0.15

}

}

}

```
#### POST /api/ai-models/train

Trains a new ML model with the provided dataset.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: multipart/form-data

```
**Request Body (multipart/form-data):**
| Field | Type | Required | Description |
|-------|------|----------|-------------|
| file | file | Yes | Training dataset file (CSV/Excel) |
| algorithm | string | Yes | Algorithm: LINEAR_REGRESSION, RANDOM_FOREST, XGBOOST, LSTM |
| modelName | string | Yes | Name for the model |
| hyperparameters | string | No | JSON string of hyperparameters |

**Response (201 Created):**
```

{

"success": true,

"data": {

"modelId": 11,

"status": "TRAINING",

"message": "Model training started"

}

}

```
#### GET /api/ai-models/compare

Compares performance metrics of multiple models.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelIds | string | Yes | Comma-separated model IDs |

**Response (200 OK):**
```


{

"success": true,

"data": {

"comparison": [

{

"modelId": 1,

"modelName": "RandomForest_v1",

"rmse": 3.45,

"mae": 2.78,

"mape": 5.23,

"rSquared": 0.92

},

{

"modelId": 2,

"modelName": "XGBoost_v1",

"rmse": 3.12,

"mae": 2.45,

"mape": 4.89,

"rSquared": 0.94

}

],

"bestModel": {

"modelId": 2,

"modelName": "XGBoost_v1",

"reason": "Lowest RMSE and highest R-squared"

}

}

}

```
#### DELETE /api/ai-models/{id}

Deletes a specific AI model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Model ID |

**Response (200 OK):**
```

{

"success": true,

"message": "Model deleted successfully"

}

```
---

### Prediction Endpoints

#### POST /api/predict

Generates predictions using the specified model.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: multipart/form-data

```
**Request Body (multipart/form-data):**
| Field | Type | Required | Description |
|-------|------|----------|-------------|
| file | file | Yes | Input data file (CSV/Excel) |
| modelId | long | No | Model ID to use (default: latest) |
| horizon | integer | No | Prediction horizon in days (default: 7) |

**Response (200 OK):**
```

{

"success": true,

"data": {

"predictionRunId": 25,

"model": "RandomForest_v1",

"totalRecords": 7,

"averagePrediction": 48.5,

"maximumPrediction": 65.2,

"minimumPrediction": 32.1,

"predictions": [

{

"attendanceDate": "2026-08-20",

"department": "Operations",

"predictedDemand": 52.3,

"confidenceInterval": {

"lower": 48.5,

"upper": 56.1

}

}

]

}

}

```
#### GET /api/predict/latest

Retrieves the latest prediction run results.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"id": 25,

"modelName": "RandomForest_v1",

"totalRecords": 7,

"averagePrediction": 48.5,

"maximumPrediction": 65.2,

"minimumPrediction": 32.1,

"createdAt": "2026-08-19T10:30:00Z",

"results": [

{

"attendanceDate": "2026-08-20",

"department": "Operations",

"actualDemand": null,

"predictedDemand": 52.3

}

]

}

}

```
#### GET /api/predict/{runId}

Retrieves a specific prediction run by ID.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| runId | long | Yes | Prediction run ID |

**Response (200 OK):**
```

{

"success": true,

"data": {

"id": 25,

"modelName": "RandomForest_v1",

"totalRecords": 7,

"results": [...]

}

}

```
---

### Settings Endpoints

#### GET /api/settings

Retrieves application settings.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"profile": {

"fullName": "John Doe",

"email": "john.doe@workforce.com",

"role": "ADMIN",

"department": "IT"

},

"appearance": {

"darkMode": false,

"compactView": false,

"animations": true

},

"notifications": {

"criticalAlerts": true,

"shiftRecommendations": true,

"systemMonitoring": true,

"emailDigest": false

},

"config": {

"model": "RandomForest_v1",

"refresh": "5m",

"apiUrl": "http://localhost:5233/api",

"mlUrl": "http://localhost:8000",

"dataRetention": 90

}

}

}

```
#### PUT /api/settings

Updates application settings.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Request Body:**
```


{

"profile": {

"fullName": "John Doe",

"email": "john.doe@workforce.com",

"role": "ADMIN",

"department": "IT"

},

"appearance": {

"darkMode": false,

"compactView": false,

"animations": true

},

"notifications": {

"criticalAlerts": true,

"shiftRecommendations": true,

"systemMonitoring": true,

"emailDigest": false

},

"config": {

"model": "RandomForest_v1",

"refresh": "5m",

"apiUrl": "http://localhost:5233/api",

"mlUrl": "http://localhost:8000",

"dataRetention": 90

}

}

```
**Response (200 OK):**
```

{

"success": true,

"message": "Settings saved successfully"

}

```
---

### Reports Endpoints

#### GET /api/reports/performance

Generates a performance report in PDF format.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelId | long | No | Model ID (default: latest) |
| startDate | string | No | Report start date (ISO format) |
| endDate | string | No | Report end date (ISO format) |

**Response (200 OK):**
- Content-Type: application/pdf
- Content-Disposition: attachment; filename="performance_report.pdf"

#### GET /api/reports/comparison

Generates a model comparison report in PDF format.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| modelIds | string | Yes | Comma-separated model IDs |

**Response (200 OK):**
- Content-Type: application/pdf
- Content-Disposition: attachment; filename="comparison_report.pdf"

#### GET /api/reports/forecast

Generates a forecast report with predictions.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| runId | long | Yes | Prediction run ID |
| format | string | No | Format: pdf, excel, csv (default: pdf) |

**Response (200 OK):**
- Content-Type: application/pdf (or application/vnd.ms-excel, text/csv)
- Content-Disposition: attachment; filename="forecast_report.pdf"

---

### Employee Endpoints

#### GET /api/employees

Retrieves all employees.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| department | string | No | Filter by department |
| status | string | No | Filter by status |
| page | integer | No | Page number (default: 1) |
| size | integer | No | Page size (default: 20) |

**Response (200 OK):**
```


{

"success": true,

"data": [

{

"id": 1,

"employeeId": "EMP001",

"name": "John Doe",

"department": "Operations",

"role": "Manager",

"email": "john.doe@workforce.com",

"hireDate": "2020-01-15",

"status": "ACTIVE"

}

],

"pagination": {

"total": 245,

"page": 1,

"pageSize": 20,

"totalPages": 13

}

}

```
#### POST /api/employees

Creates a new employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Request Body:**
```

{

"employeeId": "EMP246",

"name": "Jane Smith",

"department": "Operations",

"role": "Analyst",

"email": "jane.smith@workforce.com",

"hireDate": "2026-08-19",

"status": "ACTIVE"

}

```
**Response (201 Created):**
```

{

"success": true,

"data": {

"id": 246,

"employeeId": "EMP246",

"name": "Jane Smith",

"department": "Operations",

"role": "Analyst",

"email": "jane.smith@workforce.com",

"hireDate": "2026-08-19",

"status": "ACTIVE"

}

}

```
#### PUT /api/employees/{id}

Updates an existing employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

Content-Type: application/json

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Employee ID |

**Request Body:**
```

{

"name": "Jane Smith",

"department": "Analytics",

"role": "Senior Analyst",

"status": "ACTIVE"

}

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"id": 246,

"name": "Jane Smith",

"department": "Analytics",

"role": "Senior Analyst",

"status": "ACTIVE"

}

}

```
#### DELETE /api/employees/{id}

Deletes an employee record.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Path Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| id | long | Yes | Employee ID |

**Response (200 OK):**
```

{

"success": true,

"message": "Employee deleted successfully"

}

```
---

### Monitoring Endpoints

#### GET /api/monitoring/health

Retrieves system health status.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Response (200 OK):**
```

{

"success": true,

"data": {

"status": "HEALTHY",

"components": {

"database": "UP",

"pythonService": "UP",

"diskSpace": "OK"

},

"uptime": 86400,

"version": "1.0.0"

}

}

```
#### GET /api/monitoring/metrics

Retrieves system performance metrics.

**Request Headers:**
```

Authorization: Bearer <jwt_token>

```
**Query Parameters:**
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| period | string | No | Time period: hour, day, week (default: hour) |

**Response (200 OK):**
```


{

"success": true,

"data": {

"cpuUsage": 45.2,

"memoryUsage": 68.5,

"diskUsage": 52.3,

"requestCount": 1250,

"averageResponseTime": 245,

"errorRate": 0.02

}

}

```
---

### Error Codes

| Code | Description |
|------|-------------|
| INVALID_CREDENTIALS | Invalid username or password |
| TOKEN_EXPIRED | JWT token has expired |
| TOKEN_INVALID | JWT token is invalid |
| INSUFFICIENT_PERMISSIONS | User lacks required permissions |
| RESOURCE_NOT_FOUND | Requested resource does not exist |
| VALIDATION_ERROR | Request validation failed |
| DUPLICATE_RESOURCE | Resource already exists |
| EXTERNAL_SERVICE_ERROR | External service error |
| DATABASE_ERROR | Database operation failed |
| INTERNAL_ERROR | Unexpected server error |

---

## Machine Learning Implementation

### Overview

The Machine Learning module implements a comprehensive forecasting system with multiple algorithms, automate

### Model Architecture
```

■■■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Preprocess- ■ ■ Feature ■ ■ Model ■ ■

■ ■ ing ■ ■ Selection ■ ■ Evaluation■ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Train-Test ■ ■ Hyperparam ■ ■ Model ■ ■

■ ■ Split ■ ■ Optimization■ ■ Selection■ ■

■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■

■■■■■■■■■■■■■■■■■■

from sklearn.pipeline import Pipeline

model = Pipeline([

('scaler', StandardScaler()),

('regressor', LinearRegression())

])

```
**Hyperparameters**:
- `fit_intercept`: Whether to calculate the intercept (default: True)
- `copy_X`: Whether to copy X before fitting (default: True)
- `n_jobs`: Number of jobs for parallel computation (default: None)

**Use Cases**:
- Baseline model for comparison
- Quick prototyping
- Interpretable results
- Small datasets with linear relationships

**Pros**:
- Fast training and prediction
- Highly interpretable
- Low computational cost
- Works well with linear relationships

**Cons**:
- Cannot capture non-linear patterns
- Sensitive to outliers
- Assumes independence of features
- Limited predictive power for complex patterns

---

#### 2. Random Forest

**Description**: An ensemble learning method that constructs multiple decision trees during training and outputs the mean pr
```

```
**Mathematical Formulation**:
```

■ = (1/N) Σ f■(x)

```
Where:
- ■ = predicted value
- N = number of trees
- f■(x) = prediction of i-th tree

**Implementation Details**:
```

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV

model = RandomForestRegressor(

n_estimators=100,

max_depth=10,

min_samples_split=2,

random_state=42

)
```

```
**Hyperparameters**:
- `n_estimators`: Number of trees (default: 100, range: 10-500)
- `max_depth`: Maximum tree depth (default: None, range: 5-30)
- `min_samples_split`: Minimum samples to split (default: 2, range: 2-20)
- `min_samples_leaf`: Minimum samples per leaf (default: 1, range: 1-10)
- `max_features`: Features considered for split (default: sqrt)
- `bootstrap`: Whether to use bootstrap samples (default: True)

**Feature Importance**:
Random Forest provides built-in feature importance scores:
```

```
importances = model.feature_importances_

feature_names = X.columns

for feature, importance in zip(feature_names, importances):

print(f"{feature}: {importance:.4f}")
```

```
**Use Cases**:
- Non-linear relationships
- Feature importance analysis
- Robust to overfitting
- Medium to large datasets

**Pros**:
- Handles non-linear relationships
- Robust to overfitting
- Provides feature importance
- Handles missing values well
- Parallelizable

**Cons**:
- Slower training than linear models
- Less interpretable than linear regression
- Large memory footprint
- Prediction latency higher than simple models

---

#### 3. XGBoost (Extreme Gradient Boosting)

**Description**: An optimized gradient boosting library that implements a gradient boosting framework. It's

**Mathematical Formulation**:
```

■■ = Σ f■(x■)

```
Where:
- ■■ = prediction for i-th instance
- f■ = k-th weak learner (decision tree)
- Objective function: L(ϕ) = Σ l(y■, ■■) + Σ Ω(f■)

**Implementation Details**:
```

import xgboost as xgb

from xgboost import XGBRegressor

model = XGBRegressor(

n_estimators=100,

max_depth=6,

learning_rate=0.1,

subsample=0.8,

colsample_bytree=0.8,

random_state=42

)

```
**Hyperparameters**:
- `n_estimators`: Number of boosting rounds (default: 100, range: 50-500)
- `max_depth`: Maximum tree depth (default: 6, range: 3-15)
- `learning_rate`: Shrinkage factor (default: 0.1, range: 0.01-0.3)
- `subsample`: Row sampling rate (default: 1.0, range: 0.5-1.0)
- `colsample_bytree`: Feature sampling rate (default: 1.0, range: 0.5-1.0)
- `gamma`: Minimum loss reduction (default: 0, range: 0-10)
- `reg_alpha`: L1 regularization (default: 0, range: 0-10)
- `reg_lambda`: L2 regularization (default: 1, range: 0-10)

**Early Stopping**:
```

model.fit(

X_train, y_train,

eval_set=[(X_val, y_val)],

early_stopping_rounds=10,

verbose=False

)

```
**Use Cases**:
- High-accuracy requirements
- Large datasets
- Complex non-linear patterns
- Competitions and production systems

**Pros**:
- State-of-the-art accuracy
- Handles missing values
- Regularization built-in
- Parallel and distributed computing
- Cross-validation support

**Cons**:
- More hyperparameters to tune
- Longer training time
- Can overfit if not regularized
- Less interpretable than simpler models

---

#### 4. LSTM (Long Short-Term Memory)

**Description**: A type of recurrent neural network (RNN) capable of learning long-term dependencies, partic

**Architecture**:
```

Input Layer → LSTM Layer(s) dropout → Dense Layer → Output

```
**Mathematical Formulation**:

LSTM cell equations:
```

f■ = σ(Wf · [h■■■, x■] + bf)

i■ = σ(Wi · [h■■■, x■] + bi)

C■■ = tanh(WC · [h■■■, x■] + bC)

C■ = f■ *C*■■■ + *i*■ C■■

o■ = σ(Wo · [h■■■, x■] + bo)

h■ = o■ * tanh(C■)

```
Where:
- f■ = forget gate
- i■ = input gate
- C■■ = candidate cell state
- C■ = cell state
- o■ = output gate
- h■ = hidden state

**Implementation Details**:
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([

LSTM(50, return_sequences=True, input_shape=(timesteps, features)),

Dropout(0.2),

LSTM(50, return_sequences=False),

Dropout(0.2),

Dense(25),

Dense(1)

])

model.compile(optimizer='adam', loss='mse')
```

```
**Hyperparameters**:
- `units`: Number of LSTM units (default: 50, range: 32-128)
- `return_sequences`: Whether to return sequences (default: True for stacked LSTM)
- `dropout`: Dropout rate (default: 0.2, range: 0.1-0.5)
- `recurrent_dropout`: Recurrent dropout rate (default: 0.0)
- `activation`: Activation function (default: tanh)
- `recurrent_activation`: Recurrent activation (default: sigmoid)
- `optimizer`: Optimizer (default: adam)
- `learning_rate`: Learning rate (default: 0.001)
- `batch_size`: Batch size (default: 32, range: 16-128)
- `epochs`: Number of epochs (default: 100, range: 50-500)

**Data Preparation for LSTM**:
```

```
def create_sequences(data, sequence_length):

sequences = []

targets = []

for i in range(len(data) - sequence_length):

sequences.append(data[i:i + sequence_length])

targets.append(data[i + sequence_length])

return np.array(sequences), np.array(targets)

sequence_length = 30 # 30 days lookback

X_seq, y_seq = create_sequences(scaled_data, sequence_length)
```

```
**Use Cases**:
- Time-series forecasting
- Sequential data modeling
- Long-term dependency learning
- Complex temporal patterns

**Pros**:
- Captures temporal dependencies
- Handles complex patterns
- State-of-the-art for sequences
- Flexible architecture

**Cons**:
- Requires large datasets
- Long training time
- Computationally expensive
- Hard to interpret
- Requires careful tuning

---

### Feature Engineering

#### Time-Based Features
```

```
def extract_time_features(df):

df['year'] = df['date'].dt.year

df['month'] = df['date'].dt.month
```

```
df['day'] = df['date'].dt.day
df['day_of_week'] = df['date'].dt.dayofweek
df['day_of_year'] = df['date'].dt.dayofyear
df['week_of_year'] = df['date'].dt.isocalendar().week
df['quarter'] = df['date'].dt.quarter
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
df['is_month_start'] = df['date'].dt.is_month_start.astype(int)
df['is_month_end'] = df['date'].dt.is_month_end.astype(int)
return df
```

```
#### Lag Features
```

```
def create_lag_features(df, target_column, lags=[1, 7, 30]):
for lag in lags:
df[f'{target_column}_lag_{lag}'] = df[target_column].shift(lag)
return df
```

```
#### Rolling Statistics
```

```
def create_rolling_features(df, target_column, windows=[7, 14, 30]):
for window in windows:
df[f'{target_column}_rolling_mean_{window}'] = df[target_column].rolling(window).mean()
df[f'{target_column}_rolling_std_{window}'] = df[target_column].rolling(window).std()
df[f'{target_column}_rolling_min_{window}'] = df[target_column].rolling(window).min()
df[f'{target_column}_rolling_max_{window}'] = df[target_column].rolling(window).max()
return df
```

```
#### Seasonal Features
```

def create_seasonal_features(df):

import numpy as np

df['sin_month'] = np.sin(2 *np.pi* df['month'] / 12)

df['cos_month'] = np.cos(2 *np.pi* df['month'] / 12)

df['sin_day'] = np.sin(2 *np.pi* df['day'] / 31)

df['cos_day'] = np.cos(2 *np.pi* df['day'] / 31)

df['sin_day_of_week'] = np.sin(2 *np.pi* df['day_of_week'] / 7)

df['cos_day_of_week'] = np.cos(2 *np.pi* df['day_of_week'] / 7)

return df

```
#### External Features

- **Holiday Indicators**: Mark public holidays
- **Weather Data**: Temperature, precipitation
- **Economic Indicators**: GDP, unemployment rate
- **Company Events**: Product launches, promotions

---

### Training Pipeline

#### Step 1: Data Loading
```

import pandas as pd

def load_data(file_path):

"""Load data from CSV or Excel file."""

if file_path.endswith('.csv'):

df = pd.read_csv(file_path)

elif file_path.endswith(('.xlsx', '.xls')):

```
df = pd.read_excel(file_path)

else:

raise ValueError("Unsupported file format")

# Convert date column to datetime

if 'date' in df.columns:

df['date'] = pd.to_datetime(df['date'])

return df
```

```
#### Step 2: Data Preprocessing
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

def preprocess_data(df):

"""Clean and preprocess data."""

# Handle missing values

df = df.fillna(method='ffill').fillna(method='bfill')

# Encode categorical variables

categorical_cols = df.select_dtypes(include=['object']).columns

for col in categorical_cols:

le = LabelEncoder()

df[col] = le.fit_transform(df[col])

# Remove outliers using IQR method

numeric_cols = df.select_dtypes(include=[np.number]).columns

for col in numeric_cols:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)
```

```
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df[col] = df[col].clip(lower_bound, upper_bound)

return df
```

```
#### Step 3: Feature Engineering
```

```
def engineer_features(df):

"""Create features for ML models."""

df = extract_time_features(df)

df = create_lag_features(df, 'demand', lags=[1, 7, 30])

df = create_rolling_features(df, 'demand', windows=[7, 14, 30])

df = create_seasonal_features(df)

# Drop rows with NaN from lag features

df = df.dropna()

return df
```

```
#### Step 4: Train-Test Split
```

```
from sklearn.model_selection import train_test_split

def split_data(df, target_column, test_size=0.2):

"""Split data into train and test sets."""

X = df.drop(columns=[target_column])

y = df[target_column]
```

```
# Time-series split (no shuffling)
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=test_size, shuffle=False
)
return X_train, X_test, y_train, y_test
```

```
#### Step 5: Model Training
```

```
def train_model(X_train, y_train, algorithm='random_forest'):
"""Train a model with the specified algorithm."""
if algorithm == 'linear_regression':
from sklearn.linear_model import LinearRegression
model = LinearRegression()
elif algorithm == 'random_forest':
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
elif algorithm == 'xgboost':
from xgboost import XGBRegressor
model = XGBRegressor(n_estimators=100, random_state=42)
elif algorithm == 'lstm':
# LSTM requires special preprocessing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model = Sequential([
LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
LSTM(50),
```

```
Dense(1)
])
model.compile(optimizer='adam', loss='mse')
else:
raise ValueError(f"Unknown algorithm: {algorithm}")

model.fit(X_train, y_train)
return model
```

```
#### Step 6: Model Evaluation
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

def evaluate_model(model, X_test, y_test):
"""Evaluate model performance."""
y_pred = model.predict(X_test)

metrics = {
'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
'mae': mean_absolute_error(y_test, y_pred),
'mape': np.mean(np.abs((y_test - y_pred) / y_test)) * 100,
'r_squared': r2_score(y_test, y_pred)
}

return metrics
```

```
---

### Hyperparameter Optimization

#### Grid Search
```

```
from sklearn.model_selection import GridSearchCV

def grid_search_optimization(X_train, y_train, algorithm='random_forest'):

"""Perform grid search for hyperparameter optimization."""

if algorithm == 'random_forest':

param_grid = {

'n_estimators': [50, 100, 200],

'max_depth': [10, 20, None],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4]

}

model = RandomForestRegressor(random_state=42)

elif algorithm == 'xgboost':

param_grid = {

'n_estimators': [50, 100, 200],

'max_depth': [3, 6, 9],

'learning_rate': [0.01, 0.1, 0.2],

'subsample': [0.8, 0.9, 1.0]

}

model = XGBRegressor(random_state=42)

grid_search = GridSearchCV(
```

```
model, param_grid, cv=5,
scoring='neg_mean_squared_error',
n_jobs=-1, verbose=1
)
grid_search.fit(X_train, y_train)
return grid_search.best_estimator_, grid_search.best_params_
```

```
#### Random Search
```

```
from sklearn.model_selection import RandomizedSearchCV
def random_search_optimization(X_train, y_train, algorithm='random_forest', n_iter=50):
"""Perform random search for hyperparameter optimization."""
if algorithm == 'random_forest':
param_distributions = {
'n_estimators': [50, 100, 200, 500],
'max_depth': [10, 20, 30, None],
'min_samples_split': [2, 5, 10, 20],
'min_samples_leaf': [1, 2, 4, 8]
}
model = RandomForestRegressor(random_state=42)
random_search = RandomizedSearchCV(
model, param_distributions, n_iter=n_iter, cv=5,
scoring='neg_mean_squared_error',
n_jobs=-1, verbose=1, random_state=42
)
```

random_search.fit(X_train, y_train)

return random_search.best_estimator_, random_search.best_params_

```
---

### Model Evaluation Metrics

#### RMSE (Root Mean Square Error)
```

def calculate_rmse(y_true, y_pred):

"""Calculate Root Mean Square Error."""

return np.sqrt(mean_squared_error(y_true, y_pred))

```
**Interpretation**: Lower is better. Measures the standard deviation of prediction errors.

#### MAE (Mean Absolute Error)
```

def calculate_mae(y_true, y_pred):

"""Calculate Mean Absolute Error."""

return mean_absolute_error(y_true, y_pred)

```
**Interpretation**: Lower is better. Average absolute difference between predicted and actual values.

#### MAPE (Mean Absolute Percentage Error)
```

def calculate_mape(y_true, y_pred):

"""Calculate Mean Absolute Percentage Error."""

return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

```
**Interpretation**: Lower is better. Percentage-based error metric, useful for relative comparison.

#### R² (R-Squared)
```

def calculate_r2(y_true, y_pred):

"""Calculate R-Squared score."""

return r2_score(y_true, y_pred)

```
**Interpretation**: Range from 0 to 1. Higher is better. Proportion of variance explained by the model.

---

### Model Persistence

#### Save Model
```

import joblib

import pickle

def save_model(model, file_path, algorithm):

"""Save trained model to disk."""

if algorithm == 'lstm':

model.save(file_path + '.h5')

else:

joblib.dump(model, file_path + '.joblib')

```
#### Load Model
```

def load_model(file_path, algorithm):

"""Load trained model from disk."""

if algorithm == 'lstm':

```
from tensorflow.keras.models import load_model

return load_model(file_path + '.h5')

else:

return joblib.load(file_path + '.joblib')
```

```
---

### Prediction Service

#### Generate Prediction
```

```
def generate_prediction(model, input_data, algorithm):

"""Generate prediction using trained model."""

if algorithm == 'lstm':

# LSTM requires 3D input

input_data = input_data.reshape(1, input_data.shape[0], input_data.shape[1])

prediction = model.predict(input_data)

return prediction[0] if algorithm != 'lstm' else prediction[0][0]
```

```
#### Batch Prediction
```

```
def batch_predict(model, input_data, algorithm):

"""Generate predictions for multiple inputs."""

if algorithm == 'lstm':

input_data = input_data.reshape(input_data.shape[0], input_data.shape[1], 1)

predictions = model.predict(input_data)

return predictions.flatten() if algorithm != 'lstm' else predictions.flatten()
```

```
---

## Security Architecture

### Security Overview

The security architecture implements defense-in-depth principles with multiple layers of protection includin

### Security Layers
```

■■■

■ ■ • Output Encoding ■ ■

■ ■ • API Key Management ■ ■

■ ■■■

```
### Authentication

#### JWT (JSON Web Token) Implementation

The system uses JWT tokens for stateless authentication. Tokens are generated upon successful login and valid

**Token Structure**:
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

```
**Header**:
```

{

"alg": "HS256",

"typ": "JWT"

}

```
**Payload**:
```

{

"sub": "1234567890",

"username": "admin",

"role": "ADMIN",

"iat": 1516239022,

"exp": 1516325422

}

```
**Signature**:
```

HMACSHA256(

base64UrlEncode(header) + "." + base64UrlEncode(payload),

secret

)

```
#### JWT Service Implementation
```

@Service

public class JwtService {

@Value("${jwt.secret}")

private String jwtSecret;

@Value("${jwt.expiration}")

private long jwtExpiration;

public String generateToken(UserDetails userDetails) {

return generateToken(new HashMap<>(), userDetails);

}

public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {

return Jwts.builder()

.setClaims(extraClaims)

.setSubject(userDetails.getUsername())

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + jwtExpiration))

.signWith(getSignInKey(), SignatureAlgorithm.HS256)

.compact();

}

private Key getSignInKey() {

```
byte[] keyBytes = Decoders.BASE64.decode(jwtSecret);
return Keys.hmacShaKeyFor(keyBytes);
}

public String extractUsername(String token) {
return extractClaim(token, Claims::getSubject);
}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
final Claims claims = extractAllClaims(token);
return claimsResolver.apply(claims);
}

private Claims extractAllClaims(String token) {
return Jwts.parserBuilder()
.setSigningKey(getSignInKey())
.build()
.parseClaimsJws(token)
.getBody();
}

public boolean isTokenValid(String token, UserDetails userDetails) {
final String username = extractUsername(token);
return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
}

private boolean isTokenExpired(String token) {
return extractExpiration(token).before(new Date());
}
```

```
private Date extractExpiration(String token) {
return extractClaim(token, Claims::getExpiration);
}
}
```

```
#### JWT Authentication Filter
```

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtService jwtService;
private final UserDetailsService userDetailsService;

@Override
protected void doFilterInternal(
HttpServletRequest request,
HttpServletResponse response,
FilterChain filterChain
) throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");
final String jwt;
final String username;

if (authHeader == null || !authHeader.startsWith("Bearer ")) {
filterChain.doFilter(request, response);
return;
}
```

```
jwt = authHeader.substring(7);

username = jwtService.extractUsername(jwt);

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if (jwtService.isTokenValid(jwt, userDetails)) {

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails,

null,

userDetails.getAuthorities()

);

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}
```

```
#### Password Hashing

Passwords are hashed using BCrypt before storage in the database.
```

```
@Service

public class PasswordHashGenerator {

private static final BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
```

public String hashPassword(String rawPassword) {

return encoder.encode(rawPassword);

}

public boolean verifyPassword(String rawPassword, String hashedPassword) {

return encoder.matches(rawPassword, hashedPassword);

}

}

```
**BCrypt Configuration**:
- Algorithm: BCrypt
- Strength: 10 (default)
- Salt: Automatically generated and included in hash

### Authorization

#### Role-Based Access Control (RBAC)

The system implements RBAC with three predefined roles:

| Role | Permissions |
|------|-------------|
| **ADMIN** | Full access to all features including user management, system configuration, model training, a
| **MANAGER** | Access to dashboard, forecasting, reports, employee management, and operational settings |
| **VIEWER** | Read-only access to dashboard, forecasts, and reports |

#### Security Configuration
```

@Configuration

@EnableWebSecurity

@RequiredArgsConstructor

public class SecurityConfig {

private final JwtAuthenticationFilter jwtAuthFilter;

private final AuthenticationProvider authenticationProvider;

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
http
.csrf(csrf -> csrf.disable())
.authorizeHttpRequests(auth -> auth
.requestMatchers("/api/auth/login").permitAll()
.requestMatchers("/api/predict/**").permitAll()
.requestMatchers("/api/settings/**").permitAll()
.requestMatchers("/api/ai-models/**").hasAnyRole("ADMIN", "MANAGER")
.requestMatchers("/api/employees/**").hasAnyRole("ADMIN", "MANAGER")
.requestMatchers("/api/reports/**").hasAnyRole("ADMIN", "MANAGER", "VIEWER")
.requestMatchers("/api/dashboard/**").hasAnyRole("ADMIN", "MANAGER", "VIEWER")
.requestMatchers("/api/monitoring/**").hasRole("ADMIN")
.anyRequest().authenticated()
)
.sessionManagement(session -> session
.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
)
.authenticationProvider(authenticationProvider)
.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
return http.build();
}
@Bean
public AuthenticationProvider authenticationProvider(UserDetailsService userDetailsService) {
DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
authProvider.setUserDetailsService(userDetailsService);
```

```
authProvider.setPasswordEncoder(passwordEncoder());
return authProvider;
}

@Bean
public PasswordEncoder passwordEncoder() {
return new BCryptPasswordEncoder();
}
}
```

```
#### Method-Level Security
```

```
@Service
public class AIModelService {

@PreAuthorize("hasRole('ADMIN')")
public AIModel trainModel(MultipartFile file, String algorithm) {
// Training logic
}

@PreAuthorize("hasAnyRole('ADMIN', 'MANAGER')")
public List<AIModel> getAllModels() {
// Fetch models
}

@PreAuthorize("hasAnyRole('ADMIN', 'MANAGER', 'VIEWER')")
public AIModel getModelById(Long id) {
// Fetch model by ID
}
```

}

```
### Data Security

#### Encryption at Rest

Sensitive data is encrypted before storage in the database using AES-256 encryption.
```

@Service

public class EncryptionService {

private static final String ALGORITHM = "AES/GCM/NoPadding";

private static final int TAG_LENGTH_BIT = 128;

private static final int IV_LENGTH_BYTE = 12;

private static final int AES_KEY_BIT = 256;

@Value("${encryption.key}")

private String encryptionKey;

public String encrypt(String plaintext) throws Exception {

SecretKeySpec keySpec = new SecretKeySpec(encryptionKey.getBytes(), "AES");

GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(TAG_LENGTH_BIT, new byte[IV_LENGTH_BYTE]);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmParameterSpec);

byte[] ciphertext = cipher.doFinal(plaintext.getBytes());

return Base64.getEncoder().encodeToString(ciphertext);

}

public String decrypt(String ciphertext) throws Exception {

SecretKeySpec keySpec = new SecretKeySpec(encryptionKey.getBytes(), "AES");

```
GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(TAG_LENGTH_BIT, new byte[IV_LENGTH_BYTE]);

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.DECRYPT_MODE, keySpec, gcmParameterSpec);

byte[] plaintext = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

return new String(plaintext);

}

}
```

```
#### Encryption in Transit

All communications between services use TLS 1.3 for encryption.

**TLS Configuration**:
- Protocol: TLS 1.3
- Cipher Suites: TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256
- Certificate: X.509 with 2048-bit RSA or ECDSA
- HSTS: Enabled with max-age=31536000

#### Sensitive Data Masking
```

```
public class DataMaskingUtil {

public static String maskEmail(String email) {

if (email == null || email.isEmpty()) return email;

int atIndex = email.indexOf('@');

if (atIndex <= 1) return email;

return email.charAt(0) + "****" + email.substring(atIndex - 1);

}

public static String maskPhoneNumber(String phone) {

if (phone == null || phone.length() < 4) return phone;

return "****" + phone.substring(phone.length() - 4);
```

```
}

public static String maskCardNumber(String cardNumber) {

if (cardNumber == null || cardNumber.length() < 4) return cardNumber;

return "*--*-" + cardNumber.substring(cardNumber.length() - 4);

}

}
```

```
### API Security

#### Rate Limiting
```

```
@Component

public class RateLimitingFilter implements Filter {

private static final Map<String, RateLimiter> limiters = new ConcurrentHashMap<>();

private static final int REQUESTS_PER_MINUTE = 100;

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

String clientIp = getClientIp(httpRequest);

RateLimiter limiter = limiters.computeIfAbsent(clientIp,

k -> RateLimiter.create(REQUESTS_PER_MINUTE / 60.0));

if (!limiter.tryAcquire()) {

HttpServletResponse httpResponse = (HttpServletResponse) response;

httpResponse.setStatus(429);

httpResponse.getWriter().write("{\"error\":\"Too many requests\"}");
```

```
return;
}
chain.doFilter(request, response);
}
private String getClientIp(HttpServletRequest request) {
String xfHeader = request.getHeader("X-Forwarded-For");
if (xfHeader == null) {
return request.getRemoteAddr();
}
return xfHeader.split(",")[0];
}
}
```

```
#### Input Validation
```

```
@RestController
@RequestMapping("/api/ai-models")
public class AIModelController {
@PostMapping("/train")
public ResponseEntity<?> trainModel(
@RequestParam("file") MultipartFile file,
@RequestParam("algorithm") @Pattern(regexp = "LINEAR_REGRESSION|RANDOM_FOREST|XGBOOST|LSTM") String algorithm,
@RequestParam("modelName") @Size(min = 3, max = 100) String modelName
) {
// Validation logic
```

```
if (file.isEmpty()) {
return ResponseEntity.badRequest().body("File is required");
}
if (!isValidFileType(file)) {
return ResponseEntity.badRequest().body("Invalid file type");
}
// Training logic
}
private boolean isValidFileType(MultipartFile file) {
String contentType = file.getContentType();
return contentType.equals("text/csv") ||
contentType.equals("application/vnd.ms-excel") ||
contentType.equals("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");
}
}
```

```
#### CORS Configuration
```

```
@Configuration
public class CorsConfig {
@Bean
public CorsConfigurationSource corsConfigurationSource() {
CorsConfiguration configuration = new CorsConfiguration();
configuration.setAllowedOrigins(Arrays.asList(
"http://localhost:5173",
```

```
"https://workforce-system.com"
));
configuration.setAllowedMethods(Arrays.asList(
"GET", "POST", "PUT", "DELETE", "OPTIONS"
));
configuration.setAllowedHeaders(Arrays.asList(
"Authorization", "Content-Type", "Accept"
));
configuration.setAllowCredentials(true);
configuration.setMaxAge(3600L);
UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
source.registerCorsConfiguration("/**", configuration);
return source;
}
}
```

```
### Security Best Practices

#### OWASP Top 10 Compliance

1. **Injection Prevention**
   - Use parameterized queries (JPA/Hibernate)
   - Input validation and sanitization
   - ORM for database operations

2. **Broken Authentication**
   - Strong password policy (minimum 8 characters, complexity requirements)
   - JWT token expiration (24 hours)
   - Secure password storage (BCrypt)

3. **Sensitive Data Exposure**
   - Encryption at rest (AES-256)
   - Encryption in transit (TLS 1.3)
   - Data masking for logs

4. **XML External Entities (XXE)**
   - Disable DTD processing
   - Use secure XML parsers

5. **Broken Access Control**
   - RBAC implementation
   - Method-level security
   - JWT validation on all endpoints

6. **Security Misconfiguration**
```

```
   - Remove default credentials
   - Disable debug mode in production
   - Secure headers implementation

7. **Cross-Site Scripting (XSS)**
   - Output encoding
   - Content Security Policy (CSP)
   - Input sanitization

8. **Insecure Deserialization**
   - Use safe serialization formats
   - Validate serialized data
   - Avoid deserialization of untrusted data

9. **Using Components with Known Vulnerabilities**
   - Regular dependency updates
   - Vulnerability scanning
   - Security patches

10. **Insufficient Logging & Monitoring**
    - Security event logging
    - Audit trail
    - Alerting for suspicious activities

#### Secure Headers
```

@Configuration

public class SecurityHeadersConfig implements WebMvcConfigurer {

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(new SecurityHeadersInterceptor());

}

}

public class SecurityHeadersInterceptor implements HandlerInterceptor {

@Override

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {

response.setHeader("X-Content-Type-Options", "nosniff");

```
response.setHeader("X-Frame-Options", "DENY");
response.setHeader("X-XSS-Protection", "1; mode=block");
response.setHeader("Strict-Transport-Security", "max-age=31536000; includeSubDomains");
response.setHeader("Content-Security-Policy", "default-src 'self'");
response.setHeader("Referrer-Policy", "strict-origin-when-cross-origin");
response.setHeader("Permissions-Policy", "geolocation=(), microphone=(), camera=()");
}
}
```

Audit Logging

```
@Aspect
@Component
public class AuditLogAspect {

@Autowired
private AuditLogRepository auditLogRepository;

@AfterReturning(pointcut = "@annotation(Auditable)", returning = "result")
public void auditAfterReturning(JoinPoint joinPoint, Auditable auditable, Object result) {
AuditLog auditLog = new AuditLog();
auditLog.setAction(auditable.action());
auditLog.setResource(auditable.resource());
auditLog.setUserId(getCurrentUserId());
auditLog.setTimestamp(LocalDateTime.now());
auditLog.setDetails(joinPoint.getSignature().toString());

auditLogRepository.save(auditLog);
```

```
}

@AfterThrowing(pointcut = "@annotation(Auditable)", throwing = "exception")

public void auditAfterThrowing(JoinPoint joinPoint, Auditable auditable, Exception exception) {

AuditLog auditLog = new AuditLog();

auditLog.setAction(auditable.action());

auditLog.setResource(auditable.resource());

auditLog.setUserId(getCurrentUserId());

auditLog.setTimestamp(LocalDateTime.now());

auditLog.setStatus("FAILED");

auditLog.setErrorMessage(exception.getMessage());

auditLogRepository.save(auditLog);

}

private Long getCurrentUserId() {

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

if (authentication != null && authentication.isAuthenticated()) {

return ((UserDetails) authentication.getPrincipal()).getId();

}

return null;

}

}
```

```
### Security Monitoring

#### Security Event Types

| Event Type | Description | Severity |
|------------|-------------|----------|
| LOGIN_SUCCESS | Successful user login | INFO |
| LOGIN_FAILURE | Failed login attempt | WARNING |
| TOKEN_EXPIRED | JWT token expiration | INFO |
| UNAUTHORIZED_ACCESS | Attempted access without authorization | HIGH |
| RATE_LIMIT_EXCEEDED | Rate limit threshold exceeded | MEDIUM |
| DATA_BREACH_ATTEMPT | Suspicious data access pattern | CRITICAL |
| CONFIGURATION_CHANGE | System configuration modified | MEDIUM |

#### Alert Thresholds

| Metric | Threshold | Action |
|--------|-----------|--------|
| Failed login attempts | 5 in 5 minutes | Lock account |
| Rate limit violations | 100 in 1 minute | Block IP |
| Unusual data access | 3x normal volume | Alert admin |
| Configuration changes | Any change | Log and notify |

---

## Logging Implementation

### Backend Logging (SLF4J)
```

```
The backend uses SLF4J with Lombok's `@Slf4j` annotation for logging.

**Implementation Example**:
```

@Service

@RequiredArgsConstructor

@Slf4j

public class CSVImportService {

private final EmployeeRepository employeeRepository;

public void importCSV(File file) {

log.info("Starting CSV import for file: {}", file.getName());

try {

// Import logic

log.debug("Processed {} records", count);

log.info("CSV import completed successfully");

} catch (Exception e) {

log.error("Error during CSV import: {}", e.getMessage(), e);

}

}

}

```
**Logging Levels Used**:
- `log.info()`: Informational messages for normal operations
- `log.debug()`: Debug information for troubleshooting
- `log.error()`: Error messages with exceptions
- `log.warn()`: Warning messages for potential issues

**Files with Logging**:
- `CSVImportService.java` - CSV import operations
- `PdfTemplateService.java` - PDF generation operations
- `AuthenticationController.java` - Authentication operations (System.out.println for debugging)

### Frontend Logging (Console)

The frontend uses browser console logging for debugging and monitoring.

**Implementation Example**:
```

// In Vue components

console.log('Login attempt for user:', username);

console.error('API request failed:', error);

console.warn('Configuration warning:', message);

console.debug('Debug information:', data);

```
**Files with Console Logging**:
- `reportComponent.vue` - Report generation
- `aiModelsComponent.vue` - AI model operations
- `loginComponent.vue` - Authentication
- `dashboardComponent.vue` - Dashboard operations
- Various service files for API calls

### Python ML Service Logging

The Python ML service uses Python's built-in `logging` module.

**Implementation Example**:
```

import logging

logger = logging.getLogger(__name__)

class DatasetService:

def read_csv(self, df):

try:

logger.info("Inside DatasetService")

Process data

logger.info(f"Returning {len(result)} records")

return result

except Exception as e:

logger.exception("DatasetService Error")

raise Exception(str(e))

```
**Files with Logging**:
- `dataset_service.py` - Dataset operations

### Log Configuration

**Backend (application.properties)**:
```

Logging configuration

logging.level.root=INFO

logging.level.com.boostphysioclinic.workforceapplication=DEBUG

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n

logging.file.name=logs/workforce-application.log

```
**Python (logging configuration)**:
```

import logging

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler('logs/ml-service.log'),

logging.StreamHandler()

]

)

```
### Log Levels Summary

| Level | Backend (SLF4J) | Frontend (Console) | Python (logging) | Usage |
|-------|-----------------|-------------------|------------------|-------|
| DEBUG | log.debug() | console.debug() | logger.debug() | Detailed debugging information |
| INFO | log.info() | console.log() | logger.info() | General informational messages |
| WARN | log.warn() | console.warn() | logger.warning() | Warning messages |
| ERROR | log.error() | console.error() | logger.error() | Error messages |
| FATAL | log.error() | - | logger.critical() | Critical errors |

---

## Installation & Setup

### Prerequisites

- Java 17+
- Node.js 18+
- Python 3.9+
- MySQL 8.0+
- Gradle 7+

### Backend Setup
```

Navigate to backend directory

cd Backend/workForceApplication

Configure database in src/main/resources/application.properties

Update MySQL credentials

Build the project

gradle clean build

Run the application

gradle bootRun

```
Backend will start on port 5233.

### Frontend Setup
```

Navigate to frontend directory

cd Frontend/workforce-forecaasting-x

Install dependencies

npm install

Run development server

npm run dev

```
Frontend will start on port 5173.

### Python ML Service Setup
```

Navigate to Python service directory

cd workforce-forecasting-python

Create virtual environment

python -m venv venv

Activate virtual environment

On Windows:

venv\Scripts\activate

On Linux/Mac:

source venv/bin/activate

Install dependencies

pip install -r requirements.txt

Run FastAPI service

uvicorn serviceFast.main:app --reload --port 8000

```
ML Service will start on port 8000.

### Database Setup
```

-- Create database

CREATE DATABASE workforce;

-- Run schema.sql to create tables

-- Located in Backend/workForceApplication/src/main/resources/schema.sql

```
### Default Users

| Username | Password | Role |
|----------|----------|------|
| admin | admin123 | ADMIN |
| manager | admin123 | MANAGER |
| viewer | admin123 | VIEWER |

---

## Performance Optimization

### Performance Overview

The system implements multiple layers of performance optimization to ensure low latency, high throughput, and

### Frontend Performance Optimization

#### Code Splitting
```

// router/index.ts

import { createRouter, createWebHistory } from 'vue-router'

const routes = [

{

path: '/dashboard',

component: () => import('../settings/dashboard/dashboardComponent.vue')

},

{

path: '/ai-models',

component: () => import('../settings/aiModels/aiModelsComponent.vue')

}

]

```
**Benefits**:
- Reduced initial bundle size
- Faster page load times
- On-demand loading of routes

#### Lazy Loading Components
```

<script setup lang="ts">

import { defineAsyncComponent } from 'vue'

const HeavyComponent = defineAsyncComponent(() =>

import('./HeavyComponent.vue')

)

</script>

```
#### Image Optimization

- Use WebP format where supported
- Implement lazy loading for images
- Responsive images with srcset
- Image compression pipeline
```

<img

src="/images/dashboard.webp"

loading="lazy"

width="800"

height="600"

alt="Dashboard"

/>

```
#### Caching Strategy
```

// apiClient.ts

const apiClient = axios.create({

baseURL: API_BASE_URL,

timeout: 10000,

headers: {

'Cache-Control': 'no-cache'

}

})

// Response interceptor for caching

apiClient.interceptors.response.use(

(response) => {

// Cache GET requests

if (response.config.method === 'get') {

const cacheKey = `${response.config.url}_${JSON.stringify(response.config.params)}`

localStorage.setItem(cacheKey, JSON.stringify(response.data))

}

return response

}

)

```
#### Virtual Scrolling

For large lists (e.g., employee lists, prediction results):
```

```
<template>

<VirtualList

:items="largeDataList"

:item-size="50"

:viewport-height="400"

>

<template #default="{ item }">

<div class="list-item">{{ item.name }}</div>

</template>

</VirtualList>

</template>
```

```
#### State Management Optimization
```

```
// Use computed properties for derived state

const filteredData = computed(() => {

return rawData.value.filter(item => item.active)

})

// Debounce expensive computations

import { debounce } from 'lodash-es'

const searchResults = ref([])

const debouncedSearch = debounce((query: string) => {
```

searchResults.value = performSearch(query)

}, 300)

```
### Backend Performance Optimization

#### Database Query Optimization

**Indexing Strategy**:
```

@Entity

@Table(name = "prediction_results", indexes = {

@Index(name = "idx_run_id", columnList = "prediction_run_id"),

@Index(name = "idx_attendance_date", columnList = "attendance_date"),

@Index(name = "idx_department", columnList = "department")

})

public class PredictionResult {

// Entity fields

}

```
**Query Optimization**:
```

@Repository

public interface PredictionResultRepository extends JpaRepository<PredictionResult, Long> {

// Use derived query methods

List<PredictionResult> findByPredictionRunIdOrderByAttendanceDateAsc(Long runId);

// Use @Query for complex queries

@Query("SELECT pr FROM PredictionResult pr WHERE pr.predictionRunId = :runId AND pr.attendanceDate BETWEEN :start AND :end")

List<PredictionResult> findByRunIdAndDateRange(

@Param("runId") Long runId,

@Param("start") LocalDate start,

@Param("end") LocalDate end

);

// Use @EntityGraph for eager loading

@EntityGraph(attributePaths = {"predictionRun"})

Optional<PredictionResult> findByIdWithRun(Long id);

}

```
#### Connection Pooling
```

application.properties

spring.datasource.hikari.maximum-pool-size=20

spring.datasource.hikari.minimum-idle=5

spring.datasource.hikari.idle-timeout=30000

spring.datasource.hikari.connection-timeout=20000

spring.datasource.hikari.max-lifetime=1800000

```
**HikariCP Configuration**:
- Maximum pool size: 20 connections
- Minimum idle: 5 connections
- Connection timeout: 20 seconds
- Max lifetime: 30 minutes

#### Caching

**Spring Cache Configuration**:
```

@Configuration

```
@EnableCaching
public class CacheConfig {
@Bean
public CacheManager cacheManager() {
CaffeineCacheManager cacheManager = new CaffeineCacheManager();
cacheManager.setCaffeine(Caffeine.newBuilder()
.expireAfterWrite(10, TimeUnit.MINUTES)
.maximumSize(1000));
return cacheManager;
}
}
```

```
**Cache Usage**:
```

```
@Service
public class DashboardService {
@Cacheable(value = "dashboardMetrics", key = "#period")
public DashboardMetrics getMetrics(String period) {
// Expensive computation
}
@CacheEvict(value = "dashboardMetrics", allEntries = true)
public void clearCache() {
// Clear cache on data updates
}
}
```

```
#### Asynchronous Processing
```

```
@Service
public class PredictionService {

@Async
public CompletableFuture<PredictionResponse> generatePredictionAsync(MultipartFile file) {
// Long-running prediction task
PredictionResponse response = processPrediction(file);
return CompletableFuture.completedFuture(response);
}
}
```

```
#### Batch Processing
```

```
@Service
public class EmployeeService {

@Transactional
public void batchImportEmployees(List<EmployeeDTO> employees) {
List<Employee> employeeList = employees.stream()
.map(this::convertToEntity)
.collect(Collectors.toList());

employeeRepository.saveAll(employeeList);
}
}
```

```
### Python ML Service Performance Optimization

#### Model Optimization

**Model Quantization**:
```

import tensorflow as tf

Convert model to TensorFlow Lite for faster inference

converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

converter.target_spec.supported_types = [tf.float16]

tflite_model = converter.convert()

```
**Batch Prediction**:
```

def batch_predict(model, input_data, batch_size=32):

predictions = []

for i in range(0, len(input_data), batch_size):

batch = input_data[i:i + batch_size]

batch_predictions = model.predict(batch)

predictions.extend(batch_predictions)

return predictions

```
#### Data Processing Optimization
```

import pandas as pd

```
import numpy as np
```

Use efficient data types

```
df['column'] = df['column'].astype('float32') # Instead of float64

df['category'] = df['category'].astype('category') # For categorical data
```

Vectorized operations instead of loops

```
df['new_column'] = df['column1'] + df['column2'] # Vectorized
```

Instead of:

for i in range(len(df)):

df.at[i, 'new_column'] = df.at[i, 'column1'] + df.at[i, 'column2']

Use numpy for numerical operations

```
import numpy as np

result = np.dot(matrix1, matrix2) # Faster than Python loops
```

```
#### Caching
```

from functools import lru_cache

import joblib

@lru_cache(maxsize=128)

def expensive_computation(param):

Cache results of expensive computations

return complex_calculation(param)

Cache loaded models

model_cache = {}

def get_model(model_path):

if model_path not in model_cache:

model_cache[model_path] = joblib.load(model_path)

return model_cache[model_path]

```
#### Parallel Processing
```

from multiprocessing import Pool

from concurrent.futures import ProcessPoolExecutor

def parallel_process(data, func, workers=4):

with ProcessPoolExecutor(max_workers=workers) as executor:

results = list(executor.map(func, data))

return results

```
### Network Performance Optimization

#### HTTP/2

Configure servers to use HTTP/2 for multiplexing requests.

#### Compression

Enable Gzip compression for API responses:
```

application.properties

server.compression.enabled=true

server.compression.mime-types=application/json,application/xml,text/html,text/xml,text/plain

```
#### CDN Integration

Serve static assets through CDN for reduced latency.

### Performance Monitoring

#### Metrics Collection
```

@Component

public class PerformanceMetrics {

private final MeterRegistry meterRegistry;

public void recordApiCall(String endpoint, long duration) {

Timer.builder("api.call.duration")

.tag("endpoint", endpoint)

.register(meterRegistry)

.record(duration, TimeUnit.MILLISECONDS);

}

}

```
#### Performance Profiling
```

@Aspect

@Component

public class PerformanceMonitorAspect {

@Around("@annotation(Monitored)")

public Object monitorPerformance(ProceedingJoinPoint joinPoint) throws Throwable {

long startTime = System.currentTimeMillis();

Object result = joinPoint.proceed();

long duration = System.currentTimeMillis() - startTime;

log.info("Method {} executed in {} ms",

joinPoint.getSignature().getName(), duration);

return result;

}

}

```
### Performance Benchmarks

| Metric | Target | Current |
|--------|--------|---------|
| API Response Time (p95) | < 500ms | 350ms |
| API Response Time (p99) | < 1000ms | 750ms |
| Database Query Time | < 100ms | 65ms |
| Model Training Time | < 5min | 3.5min |
| Model Prediction Time | < 100ms | 45ms |
| Page Load Time | < 2s | 1.2s |
| Time to Interactive | < 3s | 2.1s |

---

## Scalability & High Availability

### Scalability Overview

The system is designed for horizontal scalability to handle increasing load through distributed deployment,

### Scalability Architecture
```

■■■

■ ■■■

```
### Horizontal Scaling Strategies

#### Frontend Scaling

**Deployment Strategy**:
- Static files served from CDN (CloudFront/S3)
- Application deployed to multiple instances
- Load balancer distributes traffic

**Configuration**:
```

nginx.conf

upstream frontend {

least_conn;

server frontend-1:5173;

server frontend-2:5173;

server frontend-3:5173;

}

server {

listen 80;

server_name workforce-system.com;

location / {

proxy_pass http://frontend;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

}

}

```
**Auto-scaling Policy**:
- Scale up when CPU > 70% for 5 minutes
- Scale down when CPU < 30% for 15 minutes
- Minimum instances: 2
- Maximum instances: 10

#### Backend Scaling

**Deployment Strategy**:
- Stateless application design
- Session management via JWT (no server affinity required)
- Database connection pooling

**Configuration**:
```

application.properties

spring.datasource.hikari.maximum-pool-size=50

spring.datasource.hikari.minimum-idle=10

```
**Auto-scaling Policy**:
- Scale up when CPU > 60% for 3 minutes
- Scale up when memory > 80% for 3 minutes
- Scale down when CPU < 20% for 10 minutes
- Minimum instances: 2
- Maximum instances: 8

#### Python ML Service Scaling

**Deployment Strategy**:
- Containerized deployment (Docker)
- Horizontal Pod Autoscaler (Kubernetes)
- Model serving with multiple workers

**Configuration**:
```

serviceFast/main.py

import uvicorn

if __name__ == "__main__":

uvicorn.run(

"main:app",

host="0.0.0.0",

port=8000,

workers=4, # Number of worker processes

limit_concurrency=100 # Max concurrent connections per worker

)

```
**Auto-scaling Policy**:
- Scale up when request queue > 50
- Scale up when CPU > 70% for 5 minutes
- Minimum instances: 2
- Maximum instances: 6

### Database Scaling

#### Read Replicas
```

■■■■■■■■■■■■■■■■■

■ Primary ■ (Write operations)

■ (Master) ■

■■■■■■■■■■■■■■■■■

■

■■■■■■■■■■■■■

■ ■

■■■■■■■■▼■■■■■■■■ ■▼■■■■■■■■■■■■■

■ Replica 1 ■ ■ Replica 2 ■ (Read operations)

■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■

■ Replica 3 ■

```
**Configuration**:
```

application.properties

spring.datasource.primary.url=jdbc:mysql://primary-db:3306/workforce

spring.datasource.replica.url=jdbc:mysql://replica-db:3306/workforce

```
**Implementation**:
```

@Configuration

public class DataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource.primary")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean

@ConfigurationProperties(prefix = "spring.datasource.replica")

public DataSource replicaDataSource() {

return DataSourceBuilder.create().build();

}

}

```
#### Database Sharding (Future)

Partition data by:
- Department
- Date range
- Geographic region

#### Connection Pooling

**HikariCP Configuration**:
```

spring.datasource.hikari.maximum-pool-size=50

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.connection-timeout=30000

spring.datasource.hikari.idle-timeout=600000

spring.datasource.hikari.max-lifetime=1800000

spring.datasource.hikari.leak-detection-threshold=60000

```
### Caching Strategy

#### Distributed Caching with Redis

**Architecture**:
```

■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■

■ Backend 1■ ■ Backend 2■ ■ Backend 3■

■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■

■ ■ ■

■■■■■■■■■■■■■

■ (3 nodes) ■

■■■■■■■■■■■■■■■■■■■■

```
**Configuration**:
```

@Configuration

@EnableCaching

public class RedisConfig {

@Bean

public RedisCacheManager cacheManager(RedisConnectionFactory connectionFactory) {

RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()

.entryTtl(Duration.ofMinutes(10))

.disableCachingNullValues()

.serializeValuesWith(RedisSerializationContext.SerializationPair

.fromSerializer(new GenericJackson2JsonRedisSerializer()));

return RedisCacheManager.builder(connectionFactory)

.cacheDefaults(config)

.build();

}

}

```
**Cache Strategy**:
- **Read-through**: Cache miss triggers database load
- **Write-through**: Write updates both cache and database
- **Write-behind**: Write to cache, async to database
- **Cache invalidation**: TTL-based + event-driven

### Message Queue Integration

#### RabbitMQ for Asynchronous Processing

**Use Cases**:
- Model training jobs
- Report generation
- Email notifications
- Data synchronization

**Configuration**:
```

@Configuration

public class RabbitMQConfig {

@Bean

public Queue trainingQueue() {

return new Queue("model.training", true);

}

@Bean

public Queue reportQueue() {

return new Queue("report.generation", true);

}

@Bean

public TopicExchange exchange() {

return new TopicExchange("workforce.exchange");

}

@Bean

```
public Binding trainingBinding() {
return BindingBuilder.bind(trainingQueue())
.to(exchange())
.with("model.training");
}
}
```

```
**Producer**:
```

```
@Service
public class TrainingProducer {
@Autowired
private RabbitTemplate rabbitTemplate;
public void submitTrainingJob(TrainingRequest request) {
rabbitTemplate.convertAndSend(
"workforce.exchange",
"model.training",
request
);
}
}
```

```
**Consumer**:
```

```
@RabbitListener(queues = "model.training")
public class processTrainingJob(TrainingRequest request) {
// Process training job
```

}

```
### High Availability (HA)

#### Redundancy Strategy

**Component Redundancy**:
- **Load Balancer**: Active-passive with health checks
- **Application Servers**: Active-active with auto-scaling
- **Database**: Primary-replica with automatic failover
- **Cache**: Redis cluster with replication
- **Message Queue**: RabbitMQ cluster with mirrored queues

#### Failover Mechanisms

**Database Failover**:
```

application.properties

spring.datasource.hikari.fail-fast=true

spring.datasource.hikari.connection-test-query=SELECT 1

```
**Circuit Breaker Pattern**:
```

@Configuration

public class CircuitBreakerConfig {

@Bean

public CircuitBreakerRegistry circuitBreakerRegistry() {

CircuitBreakerConfig config = CircuitBreakerConfig.custom()

.failureRateThreshold(50)

.waitDurationInOpenState(Duration.ofSeconds(30))

.slidingWindowSize(10)

.build();

```
return CircuitBreakerRegistry.of(config);
}
}
```

```
**Usage**:
```

```
@Service
public class ExternalService {
private final CircuitBreaker circuitBreaker;
public ExternalService(CircuitBreakerRegistry registry) {
this.circuitBreaker = registry.circuitBreaker("externalService");
}
public String callExternalAPI() {
return circuitBreaker.executeSupplier(() -> {
// External API call
return restTemplate.getForObject(url, String.class);
});
}
}
```

```
#### Health Checks
**Spring Boot Actuator**:
```

```
management.endpoints.web.exposure.include=health,metrics
management.endpoint.health.show-details=always
```

```
**Custom Health Indicators**:
```

```
@Component
public class PythonServiceHealthIndicator implements HealthIndicator {

@Override
public Health health() {
try {
ResponseEntity<String> response = restTemplate.getForEntity(
"http://python-service:8000/health",
String.class
);

if (response.getStatusCode() == HttpStatus.OK) {
return Health.up().withDetail("status", "Python service is up").build();
}
} catch (Exception e) {
return Health.down().withDetail("error", e.getMessage()).build();
}

return Health.down().build();
}
}
```

```
### Disaster Recovery

#### Backup Strategy

**Database Backups**:
- Full daily backups at 2:00 AM
- Incremental backups every 4 hours
- Binary logs for point-in-time recovery
- 30-day retention policy

**Configuration Backup**:
- Version control for all configuration files
- Automated backup to S3
- Configuration drift detection

#### Recovery Procedures

**Database Recovery**:
```

Restore from full backup

mysql -u root -p workforce < backup_2026_08_19.sql

Point-in-time recovery

mysqlbinlog --start-datetime="2026-08-19 10:00:00" --stop-datetime="2026-08-19 14:00:00" binlog.000001 | mysql -u root -p

```
**Application Recovery**:
- Blue-green deployment for zero-downtime updates
- Rollback capability within 5 minutes
- Automated health checks before traffic routing

### Scalability Metrics

| Metric | Target | Current |
|--------|--------|---------|
| Concurrent Users | 10,000 | 2,500 |
| Requests/Second | 1,000 | 350 |
| Database Connections | 100 | 45 |
| Cache Hit Rate | > 90% | 85% |
| Response Time (p95) | < 500ms | 350ms |
| Uptime | 99.9% | 99.5% |

### Scaling Triggers

| Trigger | Scale Up Condition | Scale Down Condition |
|---------|-------------------|---------------------|
| CPU Usage | > 70% for 5 min | < 30% for 15 min |
| Memory Usage | > 80% for 5 min | < 40% for 15 min |
| Request Queue | > 100 pending | < 10 pending |
| Response Time | p95 > 1s | p95 < 300ms |

---

## Monitoring & Observability
```

```
### Monitoring Overview

The system implements comprehensive monitoring and observability using industry-standard tools and practices

### Monitoring Architecture
```

■■■

■ ■ Application Layer ■ ■

■ ■ • Spring Boot Actuator ■ ■

■ ■ • Custom Metrics ■ ■

■ ■ • Health Checks ■ ■

■ ■■■

■ ■ • Jaeger/OpenTelemetry ■ ■

■ ■ • Distributed Tracing ■ ■

■ ■ • Request Correlation ■ ■

■ ■■■

```
**Custom Metrics**:
```

```
@Component
public class ApplicationMetrics {

private final MeterRegistry meterRegistry;
private final Counter predictionCounter;
private final Timer predictionTimer;

public ApplicationMetrics(MeterRegistry meterRegistry) {
this.meterRegistry = meterRegistry;
this.predictionCounter = Counter.builder("predictions.total")
.description("Total number of predictions generated")
.tag("type", "workforce")
.register(meterRegistry);

this.predictionTimer = Timer.builder("prediction.duration")
.description("Time taken to generate predictions")
.register(meterRegistry);
}

public void recordPrediction(String algorithm, long duration) {
predictionCounter.increment();
predictionTimer.record(duration, TimeUnit.MILLISECONDS);

Counter.builder("predictions.by.algorithm")
.tag("algorithm", algorithm)
.register(meterRegistry)
.increment();
```

}

}

```
#### System Metrics

**Key Metrics to Monitor**:

| Metric | Type | Description | Threshold |
|--------|------|-------------|-----------|
| `http.server.requests` | Counter | Total HTTP requests | - |
| `http.server.requests.duration` | Histogram | Request duration | p95 < 500ms |
| `jvm.memory.used` | Gauge | JVM memory usage | < 80% |
| `jvm.gc.pause` | Timer | GC pause duration | < 100ms |
| `database.connections.active` | Gauge | Active DB connections | < 50 |
| `database.connections.idle` | Gauge | Idle DB connections | < 20 |
| `cache.hits` | Counter | Cache hits | - |
| `cache.misses` | Counter | Cache misses | - |
| `predictions.total` | Counter | Total predictions | - |
| `predictions.duration` | Timer | Prediction time | < 100ms |
| `model.training.duration` | Timer | Model training time | < 5min |

#### Prometheus Configuration

**prometheus.yml**:
```

global:

scrape_interval: 15s

evaluation_interval: 15s

scrape_configs:

• job_name: 'backend'

metrics_path: '/actuator/prometheus'

static_configs:

• targets: ['backend:5233']

relabel_configs:

• source_labels: [__address__]

target_label: instance

replacement: 'backend'

• job_name: 'python-ml'

static_configs:

• targets: ['python-ml:8000']

metrics_path: '/metrics'

• job_name: 'mysql'

static_configs:

• targets: ['mysql-exporter:9104']

```
#### Grafana Dashboards

**Dashboard Panels**:

1. **System Health**
   - CPU Usage
   - Memory Usage
   - Disk I/O
   - Network I/O

2. **Application Performance**
   - Request Rate
   - Response Time (p50, p95, p99)
   - Error Rate
   - Throughput

3. **Database Performance**
   - Connection Pool Usage
   - Query Duration
   - Slow Queries
   - Transaction Rate

4. **ML Service Metrics**
   - Prediction Rate
   - Model Training Time
   - Model Accuracy
   - Feature Importance

### Logging
```

```
#### Structured Logging Configuration

**logback-spring.xml**:
```

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<springProperty scope="context" name="appName" source="spring.application.name"/>

<springProperty scope="context" name="logLevel" source="logging.level.root"/>

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level/logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/workforce.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/workforce.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">

<maxFileSize>100MB</maxFileSize>

</timeBasedFileNamingAndTriggeringPolicy>

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level/logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="JSON" class="ch.qos.logback.core.ConsoleAppender">

<encoder class="net.logstash.logback.encoder.LogstashEncoder">
```

<fieldNames>

<timestamp>timestamp</timestamp>

<level>level</level>

<logger>logger</logger>

<message>message</message>

</fieldNames>

</encoder>

</appender>

<root level="${logLevel}">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

</root>

</configuration>

```
#### Log Levels

| Level | Usage | Examples |
|-------|-------|----------|
| ERROR | Errors that require attention | Failed predictions, database errors |
| WARN | Potentially harmful situations | Deprecated API usage, slow queries |
| INFO | Informational messages | Application startup, user actions |
| DEBUG | Detailed diagnostic information | Method entry/exit, variable values |
| TRACE | Very detailed diagnostic information | Fine-grained events |

#### Structured Logging Implementation
```

@Slf4j

@Service

public class PredictionService {

public PredictionResponse generatePrediction(MultipartFile file) {

log.info("Starting prediction generation for file: {}", file.getOriginalFilename());

```
try {
long startTime = System.currentTimeMillis();
PredictionResponse response = processPrediction(file);
long duration = System.currentTimeMillis() - startTime;

log.info("Prediction completed successfully in {}ms", duration);
log.debug("Prediction result: {}", response);

return response;
} catch (Exception e) {
log.error("Prediction generation failed", e);
throw new PredictionException("Failed to generate prediction", e);
}
}
}
```

```
#### ELK Stack Integration
**Logstash Configuration**:
```

```
input {
tcp {
port => 5000
codec => json_lines
}
}

filter {
if [logger_name] =~ /workforce/ {
mutate {
```

```
add_field => { "application" => "workforce-forecasting" }
}
}
date {
match => ["timestamp", "ISO8601"]
}
}
output {
elasticsearch {
hosts => ["elasticsearch:9200"]
index => "workforce-logs-%{+YYYY.MM.dd}"
}
}
```

```
### Tracing (Future Implementation)
#### Distributed Tracing with OpenTelemetry
**Configuration**:
```

```
@Configuration
public class TracingConfig {
@Bean
public OpenTelemetry openTelemetry() {
Resource resource = Resource.getDefault()
.merge(Resource.create(Attributes.of(
AttributeKey.stringKey("service.name"), "workforce-backend",
AttributeKey.stringKey("service.version"), "1.0.0"
```

)));

JaegerExporter exporter = JaegerGrpcExporter.builder()

.setEndpoint("http://jaeger:14250")

.build();

SpanProcessor spanProcessor = BatchSpanProcessor.builder(exporter).build();

return OpenTelemetrySdk.builder()

.setTracerProvider(SdkTracerProvider.builder()

.addSpanProcessor(spanProcessor)

.setResource(resource)

.build())

.buildAndRegisterGlobal();

}

}

```
### Alerting

#### Alert Rules

**Prometheus Alert Rules**:
```

groups:

• name: workforce_alerts

rules:

• alert: HighErrorRate

expr: rate(http_server_requests_seconds_count{status=~"5.."}[5m]) > 0.05

for: 5m

labels:

severity: critical

annotations:

summary: "High error rate detected"

description: "Error rate is {{ $value }} for the last 5 minutes"

• alert: HighResponseTime

expr: histogram_quantile(0.95, rate(http_server_requests_seconds_bucket[5m])) > 1

for: 5m

labels:

severity: warning

annotations:

summary: "High response time detected"

description: "P95 response time is {{ $value }}s"

• alert: DatabaseConnectionPoolExhausted

expr: hikaricp_connections_active / hikaricp_connections_max > 0.9

for: 2m

labels:

severity: critical

annotations:

summary: "Database connection pool nearly exhausted"

description: "{{ $value | humanizePercentage }} of connections are in use"

• alert: HighMemoryUsage

expr: jvm_memory_used_bytes / jvm_memory_max_bytes > 0.85

for: 5m

labels:

severity: warning

annotations:

summary: "High memory usage detected"

description: "JVM memory usage is {{ $value | humanizePercentage }}"

```
#### Alertmanager Configuration
```

global:

resolve_timeout: 5m

route:

group_by: ['alertname', 'severity']

group_wait: 10s

group_interval: 10s

repeat_interval: 12h

receiver: 'default'

routes:

• match:

severity: critical

receiver: 'pagerduty'

• match:

severity: warning

receiver: 'slack'

receivers:

• name: 'default'

email_configs:

• to: 'alerts@workforce.com'

• name: 'pagerduty'

pagerduty_configs:

• service_key: '<PAGERDUTY_SERVICE_KEY>'

• name: 'slack'

slack_configs:

• api_url: '<SLACK_WEBHOOK_URL>'

channel: '#alerts'

```
### Health Checks

#### Spring Boot Actuator Health Indicators
```

@Component

public class CustomHealthIndicator implements HealthIndicator {

@Override

public Health health() {

try {

// Check external dependencies

boolean databaseHealthy = checkDatabase();

boolean pythonServiceHealthy = checkPythonService();

if (databaseHealthy && pythonServiceHealthy) {

return Health.up()

.withDetail("database", "UP")

.withDetail("python-service", "UP")

.build();

} else {

return Health.down()

```
.withDetail("database", databaseHealthy ? "UP" : "DOWN")
.withDetail("python-service", pythonServiceHealthy ? "UP" : "DOWN")
.build();
}
} catch (Exception e) {
return Health.down().withException(e).build();
}
}

private boolean checkDatabase() {
// Database health check logic
return true;
}

private boolean checkPythonService() {
// Python service health check logic
return true;
}
}
```

```
#### Health Check Endpoints

| Endpoint | Description | Response |
|----------|-------------|----------|
| `/actuator/health` | Overall health | JSON with status |
| `/actuator/health/db` | Database health | Connection status |
| `/actuator/health/redis` | Redis health | Connection status |
| `/actuator/health/python` | Python service health | Service status |

### Observability Best Practices

#### 1. Metric Naming Conventions

- Use dot notation: `http.server.requests`
- Include units: `duration.seconds`, `memory.bytes`
- Use consistent naming: `predictions.total`, `predictions.duration`

#### 2. Log Structuring

- Use structured logging (JSON format)
- Include correlation IDs for request tracing
- Add context information (user ID, request ID)

#### 3. Alert Design

- Alert on symptoms, not causes
- Set appropriate thresholds to avoid alert fatigue
- Include actionable information in alert messages
- Use severity levels appropriately
```

```
#### 4. Dashboard Design

- Create role-specific dashboards
- Include both real-time and historical data
- Use consistent color schemes
- Add annotations for deployments and incidents

### Monitoring Tools Summary

| Tool | Purpose | Integration |
|------|---------|-------------|
| Prometheus | Metrics collection | Spring Boot Actuator |
| Grafana | Visualization | Dashboards |
| ELK Stack | Log aggregation | Logstash, Filebeat |
| Jaeger | Distributed tracing | OpenTelemetry |
| Alertmanager | Alert management | Prometheus |

---

## Testing Strategy

### Testing Overview

The system implements a comprehensive testing strategy following the testing pyramid principle, with a focus

### Testing Pyramid
```

■■■■■■■■■■■■■■■■■■

■ E2E Tests ■

■ (10%) ■

■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■

■ Integration ■

■ Tests ■

■ (30%) ■

■■■■■■■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■ Unit Tests ■

■ (60%) ■

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

```
### Backend Testing

#### Unit Testing with JUnit 5

**Test Structure**:
```

Backend/workForceApplication/src/test/java/com/boostphysioclinic/workforceapplication/

■■■ controller/

■ ■■■ AIModelControllerTest.java

■ ■■■ DashboardControllerTest.java

■ ■■■ PredictionControllerTest.java

■■■ service/

■ ■■■ AIModelServiceTest.java

■ ■■■ PredictionServiceTest.java

■ ■■■ JwtServiceTest.java

■■■ repository/

■ ■■■ AIModelRepositoryTest.java

■ ■■■ EmployeeRepositoryTest.java

■■■ security/

■■■ JwtAuthenticationFilterTest.java

```
**Example Unit Test**:
```

@SpringBootTest

@AutoConfigureMockMvc

class AIModelControllerTest {

```
@Autowired
private MockMvc mockMvc;

@MockBean
private AIModelService aiModelService;

@Test
void getAllModels_shouldReturnModels() throws Exception {
List<AIModel> models = Arrays.asList(
new AIModel(1L, "RandomForest_v1", "RANDOM_FOREST"),
new AIModel(2L, "XGBoost_v1", "XGBOOST")
);

when(aiModelService.getAllModels()).thenReturn(models);

mockMvc.perform(get("/api/ai-models"))
.andExpect(status().isOk())
.andExpect(jsonPath("$.data", hasSize(2)))
.andExpect(jsonPath("$.data[0].modelName", is("RandomForest_v1")));
}

@Test
void trainModel_shouldReturnBadRequest_whenFileIsEmpty() throws Exception {
MockMultipartFile emptyFile = new MockMultipartFile(
"file",
"".getBytes()
);

mockMvc.perform(multipart("/api/ai-models/train")
.file(emptyFile)
```

```
.param("algorithm", "RANDOM_FOREST")
.param("modelName", "TestModel"))
.andExpect(status().isBadRequest());
}
}
```

```
**Service Layer Test**:
```

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

@Mock
private AIModelRepository aiModelRepository;

@Mock
private RestClient restClient;

@InjectMocks
private AIModelService aiModelService;

@Test
void trainModel_shouldSaveModel_whenTrainingSuccessful() {
// Arrange
MultipartFile file = mock(MultipartFile.class);
TrainingResponse trainingResponse = new TrainingResponse();
trainingResponse.setRmse(3.5);
trainingResponse.setMae(2.8);

when(restClient.trainModel(any(), any())).thenReturn(trainingResponse);
when(aiModelRepository.save(any())).thenReturn(new AIModel());
```

```
// Act
AIModel result = aiModelService.trainModel(file, "RANDOM_FOREST", "TestModel");

// Assert
verify(aiModelRepository).save(any(AIModel.class));
assertNotNull(result);
}
}
```

```
#### Integration Testing

**Test Configuration**:
```

```
@SpringBootTest
@TestPropertySource(locations = "classpath:application-test.properties")
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
class PredictionServiceIntegrationTest {

@Autowired
private PredictionService predictionService;

@Autowired
private PredictionResultRepository predictionResultRepository;

@Test
@Transactional
void generatePrediction_shouldSaveResults() {
// Arrange
MultipartFile file = mock(MultipartFile.class);

// Act
```

PredictionResponse response = predictionService.generatePrediction(file);

// Assert

assertNotNull(response);

List<PredictionResult> results = predictionResultRepository.findAll();

assertFalse(results.isEmpty());

}

}

```
**Test Configuration File**:
```

application-test.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

```
#### Repository Testing
```

@DataJpaTest

class AIModelRepositoryTest {

@Autowired

private TestEntityManager entityManager;

@Autowired

private AIModelRepository aiModelRepository;

```
@Test
void findByAlgorithm_shouldReturnModels() {
// Arrange
AIModel model = new AIModel();
model.setModelName("TestModel");
model.setAlgorithm("RANDOM_FOREST");
entityManager.persist(model);
entityManager.flush();

// Act
List<AIModel> found = aiModelRepository.findByAlgorithm("RANDOM_FOREST");

// Assert
assertThat(found).extracting("algorithm").containsOnly("RANDOM_FOREST");
}
}
```

```
### Frontend Testing

#### Unit Testing with Vitest

**Test Configuration**:
```

```
// vitest.config.ts
import { defineConfig } from 'vitest/config'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
plugins: [vue()],
test: {
environment: 'jsdom',
```

```
coverage: {
provider: 'v8',
reporter: ['text', 'json', 'html'],
exclude: ['node_modules/', 'src/main.ts']
}
}
})
```

```
**Component Test**:
```

```
// dashboardComponent.spec.ts
import { describe, it, expect, beforeEach } from 'vitest'
import { mount } from '@vue/test-utils'
import DashboardComponent from './dashboardComponent.vue'
import { createPinia } from 'pinia'

describe('DashboardComponent', () => {
let wrapper: any

beforeEach(() => {
wrapper = mount(DashboardComponent, {
global: {
plugins: [createPinia()]
}
})
})

it('renders dashboard title', () => {
expect(wrapper.find('h1').text()).toContain('Dashboard')
```

```
})

it('displays KPI cards', () => {

const kpiCards = wrapper.findAll('.kpi-card')

expect(kpiCards.length).toBeGreaterThan(0)

})

it('calls fetchMetrics on mount', async () => {

const fetchMetrics = vi.spyOn(wrapper.vm, 'fetchMetrics')

await wrapper.vm.$nextTick()

expect(fetchMetrics).toHaveBeenCalled()

})

})
```

```
**Service Test**:
```

```
// dashboardAPI.spec.ts

import { describe, it, expect, vi, beforeEach } from 'vitest'

import { fetchDashboardMetrics } from './dashboardAPI'

import axios from 'axios'

vi.mock('axios')

describe('dashboardAPI', () => {

beforeEach(() => {

vi.clearAllMocks()

})

it('fetches dashboard metrics successfully', async () => {

const mockData = {
```

```
totalEmployees: 245,
activeShifts: 18,
predictedDemand: 156
}

vi.mocked(axios.get).mockResolvedValue({ data: mockData })

const result = await fetchDashboardMetrics()

expect(result).toEqual(mockData)
expect(axios.get).toHaveBeenCalledWith('/api/dashboard/metrics')
})

it('handles API errors', async () => {
vi.mocked(axios.get).mockRejectedValue(new Error('Network error'))

await expect(fetchDashboardMetrics()).rejects.toThrow('Network error')
})
})
```

```
#### End-to-End Testing with Playwright

**Test Configuration**:
```

```
// playwright.config.ts
import { defineConfig, devices } from '@playwright/test'

export default defineConfig({
testDir: './e2e',
fullyParallel: true,
forbidOnly: !!process.env.CI,
retries: process.env.CI ? 2 : 0,
```

workers: process.env.CI ? 1 : undefined,

reporter: 'html',

use: {

baseURL: 'http://localhost:5173',

trace: 'on-first-retry',

},

projects: [

{

name: 'chromium',

use: { ...devices['Desktop Chrome'] },

},

{

name: 'firefox',

use: { ...devices['Desktop Firefox'] },

},

],

})

```
**E2E Test Example**:
```

// e2e/login.spec.ts

import { test, expect } from '@playwright/test'

test.describe('Authentication', () => {

test('should login successfully with valid credentials', async ({ page }) => {

await page.goto('/login')

await page.fill('input[name="username"]', 'admin')

```
await page.fill('input[name="password"]', 'admin123')

await page.click('button[type="submit"]')

await expect(page).toHaveURL('/dashboard')

await expect(page.locator('h1')).toContainText('Dashboard')

})

test('should show error with invalid credentials', async ({ page }) => {

await page.goto('/login')

await page.fill('input[name="username"]', 'invalid')

await page.fill('input[name="password"]', 'invalid')

await page.click('button[type="submit"]')

await expect(page.locator('.error-message')).toBeVisible()

})

})
```

```
### Python ML Service Testing

#### Unit Testing with pytest

**Test Configuration**:
```

pytest.ini

```
[pytest]

testpaths = tests

python_files = test_*.py

python_classes = Test*

python_functions = test_*
```

addopts = --cov=. --cov-report=html --cov-report=term

```
**Model Test**:
```

tests/test_models.py

import pytest

import numpy as np

from models.linear_regression_model import LinearRegressionModel

class TestLinearRegressionModel:

@pytest.fixture

def model(self):

return LinearRegressionModel()

@pytest.fixture

def sample_data(self):

X = np.random.rand(100, 5)

y = np.random.rand(100)

return X, y

def test_train(self, model, sample_data):

X, y = sample_data

model.train(X, y)

assert model.is_trained()

def test_predict(self, model, sample_data):

X, y = sample_data

model.train(X, y)

```
predictions = model.predict(X[:10])

assert len(predictions) == 10

assert all(isinstance(p, (int, float)) for p in predictions)

def test_predict_before_train_raises_error(self, model):

X = np.random.rand(10, 5)

with pytest.raises(ValueError):

model.predict(X)
```

```
**API Test**:
```

tests/test_api.py

```
from fastapi.testclient import TestClient

from serviceFast.main import app

client = TestClient(app)

def test_health_check():

response = client.get("/health")

assert response.status_code == 200

assert response.json()["status"] == "healthy"

def test_train_model():

response = client.post("/train", json={

"algorithm": "linear_regression",

"hyperparameters": {}

})

assert response.status_code == 200
```

assert "model_id" in response.json()

def test_predict():

response = client.post("/predict", json={

"model_id": 1,

"data": [[1, 2, 3, 4, 5]]

})

assert response.status_code == 200

assert "prediction" in response.json()

```
### Test Coverage

#### Coverage Goals

| Component | Target Coverage | Current Coverage |
|-----------|----------------|------------------|
| Backend Unit Tests | 80% | 75% |
| Backend Integration Tests | 70% | 65% |
| Frontend Unit Tests | 80% | 70% |
| Frontend E2E Tests | 60% | 50% |
| Python Unit Tests | 85% | 80% |

#### Coverage Configuration

**Backend (JaCoCo)**:
```

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.10</version>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

```
</execution>
<execution>
<id>report</id>
<phase>test</phase>
<goals>
<goal>report</goal>
</goals>
</execution>
</executions>
</plugin>
```

```
**Frontend (Vitest)**:
```

```
// vitest.config.ts
export default defineConfig({
test: {
coverage: {
provider: 'v8',
reporter: ['text', 'json', 'html'],
statements: 80,
branches: 80,
functions: 80,
lines: 80
}
}
})
```

```
### Test Data Management

#### Test Data Fixtures

**Backend**:
```

```
@Component

public class TestDataFactory {

public static AIModel createTestModel() {

AIModel model = new AIModel();

model.setModelName("TestModel_v1");

model.setAlgorithm("RANDOM_FOREST");

model.setRmse(3.5);

model.setMae(2.8);

model.setStatus("COMPLETED");

return model;

}

public static Employee createTestEmployee() {

Employee employee = new Employee();

employee.setEmployeeId("EMP001");

employee.setName("John Doe");

employee.setDepartment("Operations");

employee.setRole("Analyst");

employee.setStatus("ACTIVE");

return employee;

}

}
```

```
**Frontend**:
```

```
// test-utils.ts
export const mockDashboardMetrics = {
totalEmployees: 245,
activeShifts: 18,
predictedDemand: 156,
capacityUtilization: 78.5
}

export const mockAIModels = [
{
id: 1,
modelName: 'RandomForest_v1',
algorithm: 'RANDOM_FOREST',
rmse: 3.5,
mae: 2.8,
status: 'COMPLETED'
}
]
```

```
### Continuous Testing
#### Pre-commit Hooks
```

```
#!/bin/bash
```

.git/hooks/pre-commit

Backend tests

cd Backend/workForceApplication

mvn test

Frontend tests

cd ../../Frontend/workforce-forecaasting-x

npm run test

Python tests

cd ../../workforce-forecasting-python

pytest

Check exit codes

if [$? -ne 0]; then

echo "Tests failed. Commit aborted."

exit 1

fi

```
### Testing Best Practices

#### 1. Test Independence

- Each test should be independent
- Tests should not rely on execution order
- Use proper setup and teardown

#### 2. Test Naming

- Use descriptive test names
- Follow pattern: `methodName_shouldExpectedResult_whenCondition`

#### 3. Test Organization

- Group related tests
- Use test suites for complex features
- Maintain clear test structure

#### 4. Mocking

- Mock external dependencies
- Use test doubles for slow operations
- Avoid over-mocking

#### 5. Test Maintenance

- Keep tests updated with code changes
- Remove obsolete tests
```

```
- Refactor tests for readability

---

## CI/CD Pipeline

### CI/CD Overview

The system implements a comprehensive CI/CD pipeline using GitHub Actions for continuous integration and con

### CI/CD Architecture
```

■■■

■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Code Push ■■■■■■■ Build & ■■■■■■■ Test ■ ■

■ ■ ■ ■ Compile ■ ■ Stage ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Trigger ■ ■ Docker ■ ■ Deploy ■ ■

■ ■ Detection ■ ■ Build ■ ■ Stage ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ ■ ■ ■

■ ▼ ▼ ▼ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■ Branch ■ ■ Image Push ■ ■ Notify ■ ■

■ ■ Validation ■ ■ to Registry■ ■ Team ■ ■

■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■

■ ■

■■■■■■■■■■■■■■■■■

```
push:
branches: [ main, develop ]
paths:
• 'Backend/**'
pull_request:
branches: [ main, develop ]
paths:
• 'Backend/**'

jobs:
build:
runs-on: ubuntu-latest

steps:
• name: Checkout code
uses: actions/checkout@v3

• name: Set up JDK 17
uses: actions/setup-java@v3
with:
java-version: '17'
distribution: 'temurin'
cache: maven

• name: Build with Maven
run: |
cd Backend/workForceApplication
mvn clean compile

• name: Run unit tests
```

```
run: |
cd Backend/workForceApplication
mvn test
• name: Generate test coverage
run: |
cd Backend/workForceApplication
mvn jacoco:report
• name: Upload coverage to Codecov
uses: codecov/codecov-action@v3
with:
file: Backend/workForceApplication/target/site/jacoco/jacoco.xml
flags: backend
• name: Build Docker image
run: |
cd Backend/workForceApplication
docker build -t workforce-backend:${{ github.sha }} .
• name: Log in to Docker Hub
uses: docker/login-action@v2
with:
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
• name: Push Docker image
run: |
docker tag workforce-backend:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-backend:latest
```

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-backend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-backend:${{ github.sha }}

• name: Deploy to staging (main branch only)

if: github.ref == 'refs/heads/main'

run: |

Deployment script for staging environment

echo "Deploying to staging"

• name: Deploy to production (manual trigger)

if: github.ref == 'refs/heads/main' && github.event_name == 'workflow_dispatch'

run: |

Deployment script for production environment

echo "Deploying to production"

```
#### Frontend CI/CD Pipeline

**.github/workflows/frontend-ci.yml**:
```

name: Frontend CI/CD

on:

push:

branches: [main, develop]

paths:

• 'Frontend/**'

pull_request:

branches: [main, develop]

paths:

• 'Frontend/**'

```
jobs:
build:
runs-on: ubuntu-latest
steps:
• name: Checkout code
uses: actions/checkout@v3
• name: Set up Node.js
uses: actions/setup-node@v3
with:
node-version: '18'
cache: 'npm'
cache-dependency-path: Frontend/workforce-forecaasting-x/package-lock.json
• name: Install dependencies
run: |
cd Frontend/workforce-forecaasting-x
npm ci
• name: Run linter
run: |
cd Frontend/workforce-forecaasting-x
npm run lint
• name: Run unit tests
run: |
cd Frontend/workforce-forecaasting-x
npm run test:unit
```

• name: Generate coverage report

run: |

cd Frontend/workforce-forecaasting-x

npm run test:coverage

• name: Upload coverage to Codecov

uses: codecov/codecov-action@v3

with:

file: Frontend/workforce-forecaasting-x/coverage/coverage-final.json

flags: frontend

• name: Build application

run: |

cd Frontend/workforce-forecaasting-x

npm run build

• name: Build Docker image

run: |

cd Frontend/workforce-forecaasting-x

docker build -t workforce-frontend:${{ github.sha }} .

• name: Log in to Docker Hub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKER_USERNAME }}

password: ${{ secrets.DOCKER_PASSWORD }}

• name: Push Docker image

run: |

docker tag workforce-frontend:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-frontend:${{ github.sha }}

• name: Deploy to staging

if: github.ref == 'refs/heads/main'

run: |

Deployment script for staging environment

echo "Deploying frontend to staging"

```
#### Python ML Service CI/CD Pipeline

**.github/workflows/python-ci.yml**:
```

name: Python ML Service CI/CD

on:

push:

branches: [main, develop]

paths:

• 'workforce-forecasting-python/**'

pull_request:

branches: [main, develop]

paths:

• 'workforce-forecasting-python/**'

jobs:

build:

runs-on: ubuntu-latest

steps:

• name: Checkout code

uses: actions/checkout@v3

• name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.9'

cache: 'pip'

cache-dependency-path: workforce-forecasting-python/requirements.txt

• name: Install dependencies

run: |

cd workforce-forecasting-python

pip install -r requirements.txt

pip install pytest pytest-cov

• name: Run linter

run: |

cd workforce-forecasting-python

flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics

flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

• name: Run unit tests

run: |

cd workforce-forecasting-python

pytest --cov=. --cov-report=xml --cov-report=term

• name: Upload coverage to Codecov

uses: codecov/codecov-action@v3

with:

file: workforce-forecasting-python/coverage.xml

flags: python

• name: Build Docker image

run: |

cd workforce-forecasting-python

docker build -t workforce-ml-service:${{ github.sha }} .

• name: Log in to Docker Hub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKER_USERNAME }}

password: ${{ secrets.DOCKER_PASSWORD }}

• name: Push Docker image

run: |

docker tag workforce-ml-service:${{ github.sha }} ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:latest

docker push ${{ secrets.DOCKER_USERNAME }}/workforce-ml-service:${{ github.sha }}

```
### Pipeline Stages

#### 1. Build Stage

**Purpose**: Compile source code and generate artifacts

**Backend**:
```

mvn clean compile

```
**Frontend**:
```

npm run build

```
**Python**:
```

pip install -r requirements.txt

```
#### 2. Test Stage

**Purpose**: Execute automated tests

**Backend**:
```

mvn test

mvn jacoco:report

```
**Frontend**:
```

npm run test:unit

npm run test:coverage

```
**Python**:
```

pytest --cov=. --cov-report=xml

```
#### 3. Package Stage

**Purpose**: Create deployable artifacts

**Backend**:
```

mvn package

```
**Frontend**:
```

npm run build

```
**Python**:
```

python setup.py sdist bdist_wheel

```
#### 4. Deploy Stage

**Purpose**: Deploy to environments

**Staging Deployment**:
- Automatic on merge to main branch
- Runs integration tests
- Updates staging environment

**Production Deployment**:
- Manual trigger required
- Runs full regression tests
- Blue-green deployment
- Zero-downtime rollout

### Environment Configuration

#### Staging Environment

**Configuration**:
- Database: Staging MySQL instance
- Cache: Staging Redis instance
- Message Queue: Staging RabbitMQ instance
- Domain: staging.workforce-system.com

**Deployment Strategy**:
- Rolling updates
- Health checks before traffic routing
- Automatic rollback on failure
```

```
#### Production Environment

**Configuration**:
- Database: Production MySQL with replicas
- Cache: Production Redis cluster
- Message Queue: Production RabbitMQ cluster
- Domain: workforce-system.com

**Deployment Strategy**:
- Blue-green deployment
- Canary releases (10% traffic initially)
- Gradual traffic shift
- Full monitoring during rollout

### Deployment Strategies

#### Blue-Green Deployment
```

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

■ Blue ■ ■ Green ■

■ (Active) ■ ■ (Inactive) ■

■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■

■ ■

■■■■■■■■■■■■■■■■■■■■■■■

■

■■■■■■■▼■■■■■■■

■ Load ■

■ Balancer ■

■■■■■■■■■■■■■■

```
**Process**:
1. Deploy new version to Green environment
2. Run health checks on Green
3. Switch traffic to Green
4. Keep Blue for rollback
5. Decommission Blue after successful deployment

#### Canary Deployment
```

■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■

■ Stable ■ ■ Canary ■

■ (90% traffic)■ ■ (10% traffic)■

■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■

```
**Process**:
1. Deploy new version to subset of instances
2. Route 10% traffic to new version
3. Monitor metrics and errors
4. Gradually increase traffic if healthy
5. Rollback if issues detected

### Quality Gates

#### Pre-merge Checks

- All unit tests must pass
- Code coverage > 80%
- No critical security vulnerabilities
- Linter checks pass
- Build succeeds

#### Pre-deployment Checks

- All integration tests pass
- E2E tests pass
- Performance benchmarks met
- Security scan passes
- Manual approval required for production

### Rollback Strategy

#### Automatic Rollback Triggers
```

```
- Error rate > 5% for 5 minutes
- Response time p95 > 2 seconds
- Health check failures > 3 consecutive
- Critical security vulnerability detected

#### Manual Rollback Procedure
```

Revert to previous Docker image

docker pull workforce-backend:previous-tag

kubectl rollout undo deployment/backend

```
### Pipeline Monitoring

#### Key Metrics

| Metric | Threshold | Alert |
|--------|-----------|-------|
| Build Success Rate | > 95% | < 95% |
| Test Pass Rate | 100% | < 100% |
| Deployment Success Rate | > 98% | < 98% |
| Deployment Time | < 10 min | > 15 min |
| Rollback Rate | < 5% | > 10% |

### CI/CD Best Practices

#### 1. Pipeline as Code

- Version control all pipeline configurations
- Use reusable workflow templates
- Document pipeline steps

#### 2. Security

- Use secrets management
- Scan for vulnerabilities
- Sign artifacts
- Implement approval gates

#### 3. Performance
```

```
- Parallelize independent jobs
- Use caching for dependencies
- Optimize Docker layer caching

#### 4. Reliability

- Implement retry logic
- Use idempotent operations
- Validate before deployment
- Monitor pipeline health

---

## Deployment Architecture

### Deployment Overview

The system deployment architecture supports multiple deployment environments (development, staging, productio

### Deployment Architecture Diagram
```

■■■

■ ■ Kubernetes Cluster ■ ■

■ ■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■ ■

■ ■ ■ Frontend ■ ■ Backend ■ ■ Python ■ ■ ■

■ ■ ■ Pods ■ ■ Pods ■ ■ ML Pods ■ ■ ■

■ ■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■■■■■■■■■■■■■ ■ ■

■ ■■■■■■■■■■■■■

[redacted]

```
### Docker Configuration

#### Backend Dockerfile

**Backend/workForceApplication/Dockerfile**:
```

FROM maven:3.8.6-eclipse-temurin-17 AS builder

WORKDIR /app

COPY pom.xml .

COPY src ./src

RUN mvn clean package -DskipTests

FROM eclipse-temurin:17-jre-alpine

WORKDIR /app

COPY --from=builder /app/target/*.jar app.jar

EXPOSE 5233

ENTRYPOINT ["java", "-jar", "app.jar"]

```
#### Frontend Dockerfile

**Frontend/workforce-forecaasting-x/Dockerfile**:
```

FROM node:18-alpine AS builder

WORKDIR /app

COPY package*.json ./

RUN npm ci

COPY . .

RUN npm run build

FROM nginx:alpine

COPY --from=builder /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```
**nginx.conf**:
```

server {

listen 80;

server_name localhost;

root /usr/share/nginx/html;

index index.html;

location / {

try_files $uri $uri/ /index.html;

}

location /api {

proxy_pass http://backend:5233;

proxy_set_header Host $host;

proxy_set_header X-Real-IP $remote_addr;

}

}

```
#### Python ML Service Dockerfile

**workforce-forecasting-python/Dockerfile**:
```

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "serviceFast.main:app", "--host", "0.0.0.0", "--port", "8000"]

```
### Kubernetes Deployment

#### Backend Deployment

**k8s/backend-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: backend

labels:

app: backend

spec:

replicas: 3

selector:

matchLabels:

app: backend

template:

metadata:

labels:

app: backend

spec:

containers:

• name: backend

image: workforce-backend:latest

ports:

• containerPort: 5233

env:

• name: SPRING_DATASOURCE_URL

valueFrom:

secretKeyRef:

name: db-secret

key: url

• name: SPRING_DATASOURCE_USERNAME

valueFrom:

secretKeyRef:

name: db-secret

key: username

• name: SPRING_DATASOURCE_PASSWORD

valueFrom:

secretKeyRef:

name: db-secret

key: password

resources:

requests:

memory: "512Mi"

cpu: "500m"

limits:

memory: "1Gi"

cpu: "1000m"

livenessProbe:

httpGet:

path: /actuator/health

port: 5233

initialDelaySeconds: 60

periodSeconds: 10

readinessProbe:

httpGet:

path: /actuator/health

port: 5233

initialDelaySeconds: 30

periodSeconds: 5

_

---

apiVersion: v1

kind: Service

metadata:

name: backend

spec:

selector:

app: backend

ports:

• protocol: TCP

port: 5233

targetPort: 5233

type: ClusterIP

```
#### Frontend Deployment

**k8s/frontend-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

labels:

app: frontend

spec:

replicas: 2

selector:

matchLabels:

app: frontend

template:

metadata:

labels:

app: frontend

spec:

containers:

• name: frontend

image: workforce-frontend:latest

ports:

• containerPort: 80

resources:

requests:

memory: "128Mi"

cpu: "100m"

limits:

memory: "256Mi"

cpu: "200m"

_

---

apiVersion: v1

kind: Service

metadata:

name: frontend

spec:

selector:

app: frontend

ports:

• protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

```
#### Python ML Service Deployment

**k8s/python-ml-deployment.yaml**:
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: python-ml

labels:

app: python-ml

spec:

replicas: 2

selector:

matchLabels:

app: python-ml

template:

metadata:

labels:

app: python-ml

spec:

containers:

• name: python-ml

image: workforce-ml-service:latest

ports:

• containerPort: 8000

resources:

requests:

memory: "1Gi"

cpu: "500m"

limits:

memory: "2Gi"

cpu: "2000m"

_

apiVersion: v1

kind: Service

metadata:

name: python-ml

spec:

selector:

app: python-ml

ports:

• protocol: TCP

port: 8000

targetPort: 8000

type: ClusterIP

```
### Infrastructure as Code (Terraform)

#### VPC and Networking

**terraform/vpc.tf**:
```

resource "aws_vpc" "workforce_vpc" {

cidr_block = "10.0.0.0/16"

enable_dns_hostnames = true

enable_dns_support = true

tags = {

Name = "workforce-vpc"

}

}

```
resource "aws_subnet" "public_subnet" {
count = 2
vpc_id = aws_vpc.workforce_vpc.id
cidr_block = "10.0.${count.index}.0/24"
availability_zone = data.aws_availability_zones.available.names[count.index]
map_public_ip_on_launch = true
tags = {
Name = "public-subnet-${count.index}"
}
}
resource "aws_subnet" "private_subnet" {
count = 2
vpc_id = aws_vpc.workforce_vpc.id
cidr_block = "10.0.${count.index + 10}.0/24"
availability_zone = data.aws_availability_zones.available.names[count.index]
tags = {
Name = "private-subnet-${count.index}"
}
}
```

```
#### EKS Cluster
**terraform/eks.tf**:
```

```
resource "aws_eks_cluster" "workforce_cluster" {
name = "workforce-cluster"
role_arn = aws_iam_role.eks_cluster.arn
```

```
version = "1.27"

vpc_config {

subnet_ids = aws_subnet.private_subnet[*].id

}

depends_on = [

aws_iam_role_policy_attachment.eks_cluster_policy

]

}

resource "aws_eks_node_group" "workforce_nodes" {

cluster_name = aws_eks_cluster.workforce_cluster.name

node_group_name = "workforce-nodes"

node_role_arn = aws_iam_role.eks_nodes.arn

subnet_ids = aws_subnet.private_subnet[*].id

scaling_config {

desired_size = 3

max_size = 5

min_size = 2

}

instance_types = ["t3.medium"]

}
```

```
#### RDS Database

**terraform/rds.tf**:
```

```
resource "aws_db_instance" "workforce_db" {
```

```
identifier = "workforce-db"

engine = "mysql"

engine_version = "8.0"

instance_class = "db.t3.medium"

allocated_storage = 20

max_allocated_storage = 100

storage_type = "gp2"

storage_encrypted = true

db_name = "workforce"

username = var.db_username

password = var.db_password

vpc_security_group_ids = [aws_security_group.db_sg.id]

db_subnet_group_name = aws_db_subnet_group.main.name

backup_retention_period = 30

multi_az = true

skip_final_snapshot = false

final_snapshot_identifier = "workforce-db-final-snapshot"

tags = {

Name = "workforce-database"

}

}
```

```
### Environment Configuration

#### Development Environment

**Infrastructure**:
- Local Docker Compose
- Local MySQL instance
- Local Redis instance
- No load balancer

**Configuration**:
```

docker-compose.yml

version: '3.8'

services:

backend:

build: ./Backend/workForceApplication

ports:

• "5233:5233"

environment:

• SPRING_PROFILES_ACTIVE=dev

• SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/workforce

depends_on:

• mysql

frontend:

build: ./Frontend/workforce-forecaasting-x

ports:

• "5173:80"

python-ml:

build: ./workforce-forecasting-python

ports:

• "8000:8000"

mysql:

image: mysql:8.0

environment:

• MYSQL_ROOT_PASSWORD=root

• MYSQL_DATABASE=workforce

volumes:

• mysql-data:/var/lib/mysql

redis:

image: redis:alpine

ports:

• "6379:6379"

volumes:

mysql-data:

```
#### Staging Environment

**Infrastructure**:
- AWS EKS cluster (small)
- RDS MySQL (multi-AZ)
- ElastiCache Redis
- Application Load Balancer

**Configuration**:
- 2 replicas per service
- Development database
- Staging-specific secrets

#### Production Environment

**Infrastructure**:
- AWS EKS cluster (large)
- RDS MySQL (multi-AZ with read replicas)
- ElastiCache Redis cluster
- Network Load Balancer
- CloudFront CDN

**Configuration**:
- 3+ replicas per service
- Production database with replicas
- Production secrets
- Enhanced monitoring

### Deployment Process
```

```
#### 1. Build and Push Images
```

Build images

docker build -t workforce-backend:latest ./Backend/workForceApplication

docker build -t workforce-frontend:latest ./Frontend/workforce-forecaasting-x

docker build -t workforce-ml-service:latest ./workforce-forecasting-python

Tag for registry

docker tag workforce-backend:latest registry.workforce.com/workforce-backend:latest

docker tag workforce-frontend:latest registry.workforce.com/workforce-frontend:latest

docker tag workforce-ml-service:latest registry.workforce.com/workforce-ml-service:latest

Push to registry

docker push registry.workforce.com/workforce-backend:latest

docker push registry.workforce.com/workforce-frontend:latest

docker push registry.workforce.com/workforce-ml-service:latest

```
#### 2. Apply Kubernetes Manifests
```

Apply deployments

kubectl apply -f k8s/backend-deployment.yaml

kubectl apply -f k8s/frontend-deployment.yaml

kubectl apply -f k8s/python-ml-deployment.yaml

Apply services

kubectl apply -f k8s/backend-service.yaml

kubectl apply -f k8s/frontend-service.yaml

kubectl apply -f k8s/python-ml-service.yaml

Apply secrets

kubectl apply -f k8s/secrets.yaml

```
#### 3. Verify Deployment
```

Check pod status

kubectl get pods

Check services

kubectl get services

Check logs

kubectl logs -f deployment/backend

Port forward for testing

kubectl port-forward service/backend 5233:5233

```
### Monitoring and Logging in Production

#### Centralized Logging

- ELK Stack for log aggregation
- Logstash for log processing
- Kibana for log visualization
- Filebeat for log shipping

#### Metrics Collection

- Prometheus for metrics collection
- Grafana for visualization
- Alertmanager for alerting

#### Distributed Tracing

- Jaeger for distributed tracing
- OpenTelemetry for instrumentation

### Disaster Recovery

#### Backup Strategy

- Automated daily backups
- Point-in-time recovery
- Cross-region replication
- 30-day retention
```

```
#### Recovery Procedures

1. **Database Recovery**
   - Restore from RDS snapshot
   - Replay binary logs
   - Verify data integrity

2. **Application Recovery**
   - Deploy previous Docker image
   - Restore configuration
   - Verify functionality

3. **Infrastructure Recovery**
   - Use Terraform to rebuild
   - Restore from infrastructure backups
   - Validate networking

### Security in Production

#### Network Security

- VPC with private subnets
- Security groups with least privilege
- Network ACLs
- WAF for web application firewall

#### Secrets Management

- AWS Secrets Manager
- Kubernetes secrets
```

```
- Environment-specific secrets
- Rotation policies

#### Compliance

- SOC 2 Type II compliance
- GDPR compliance
- Regular security audits
- Penetration testing

### Cost Optimization

#### Resource Optimization

- Right-sizing instances
- Auto-scaling policies
- Spot instances for non-critical workloads
- Reserved instances for baseline load

#### Storage Optimization

- S3 lifecycle policies
- EBS volume optimization
- Database storage optimization
- Log retention policies

### Deployment Checklist

#### Pre-Deployment
```

```
- [ ] All tests pass
- [ ] Code coverage meets threshold
- [ ] Security scan passes
- [ ] Performance benchmarks met
- [ ] Documentation updated
- [ ] Rollback plan prepared

#### During Deployment

- [ ] Health checks pass
- [ ] Metrics within normal range
- [ ] No errors in logs
- [ ] Traffic routing successful
- [ ] Monitoring active

#### Post-Deployment

- [ ] Verify functionality
- [ ] Monitor for 30 minutes
- [ ] Check error rates
- [ ] Validate performance
- [ ] Update runbook
- [ ] Notify team

---

## Configuration

### Backend Configuration (application.properties)
```

Server Configuration

server.port=5233

Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/workforce

spring.datasource.username=root

spring.datasource.password=your_password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

JPA Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=false

Python API Configuration

fastapi.base-url=http://localhost:8000

python.api.url=http://localhost:8000/train

File Upload Configuration

spring.servlet.multipart.max-file-size=100MB

spring.servlet.multipart.max-request-size=100MB

JWT Configuration

jwt.secret=your_jwt_secret_key

jwt.expiration=86400000

```
### Frontend Configuration (apiClient.ts)
```

const API_BASE_URL = 'http://localhost:5233/api'

```
---

## Security

### Authentication Flow

1. User submits credentials to `/api/auth/login`
2. Backend validates credentials
3. JWT token generated and returned
4. Token stored in localStorage
5. Subsequent requests include token in Authorization header
6. JwtAuthenticationFilter validates token on each request

### Role-Based Access Control

- **ADMIN**: Full access to all features
- **MANAGER**: Access to dashboard, forecasting, reports
- **VIEWER**: Read-only access to dashboard and reports

### Security Configuration

- CSRF protection disabled for API
- Stateless session management
- JWT-based authentication
- CORS enabled for frontend origin
- Password hashing with BCrypt

---
```

```
## Development Guidelines

### Code Style

- **Java**: Follow Google Java Style Guide
- **TypeScript**: ESLint with Prettier
- **Python**: PEP 8 guidelines
- **Vue**: Vue.js Style Guide

### Git Workflow

1. Create feature branch from main
2. Make changes and commit
3. Push to remote
4. Create pull request
5. Code review
6. Merge to main

### Testing

- Backend: JUnit 5, Mockito
- Frontend: Vitest, Vue Test Utils
- Python: pytest, unittest

---

## Deployment

### Backend Deployment
```

Build JAR file

mvn clean package

Run JAR

java -jar target/workForceApplication.jar

```
### Frontend Deployment
```

Build for production

npm run build

Deploy dist/ folder to web server

```
### Python Service Deployment
```

Use Gunicorn for production

gunicorn serviceFast.main:app -w 4 -k uvicorn.workers.UvicornWorker --_bind 0.0.0.0:8000

```
---

## Troubleshooting

### Common Issues

1. **Database Connection Error**
   - Verify MySQL is running
   - Check credentials in application.properties
   - Ensure database exists

2. **CORS Error**
   - Verify CorsConfig allows frontend origin
   - Check frontend API base URL

3. **Python Service Not Responding**
   - Verify FastAPI is running on port 8000
   - Check Python dependencies are installed
   - Review service logs

4. **JWT Token Expired**
   - Token expires after 24 hours (configurable)
   - User must re-login

---

## Future Enhancements

- [ ] Real-time WebSocket updates
```