Software Design Documentation

WorkNexus

Submitted by

Name: Ali Irfan

Reg. no.: f22binft1m01285@iub.edu.pk

Submitted to

Mr. Muzamil Ur Rehman

Department of Information Technology

Faculty of Computing

The Islamia University of Bahawalpur

Meeting Details

Sr. No	Details	Date	Supervisor Signature

REVISION HISTORY

Version	Date	Description	Author
1.0	31-12-2025	Initial SDD Draft	Ali Irfan
1.1	07-01-2026	Architecture Section Added	Ali Irfan
1.2	08-01-2026	Additional Sections Added	Ali Irfan
1.3	09-01-2026	Functional and Technical Content Added	Ali Irfan
1.4	10-01-2026	Content Refinement and Updates	Ali Irfan
1.5	10-01-2026	Final Review and Approval Version	Ali Irfan

Summary

WorkNexus implements a modern web architecture designed for scalability, maintainability, and security. The system follows a container-based deployment model with clear separation between frontend, backend, and database components, each running in isolated Docker containers orchestrated by Docker Compose.

The technical design centers around a **React.js frontend** utilizing component-based architecture with TanStack Query for efficient data fetching and Zustand for client-side state management. The **Node.js/Express backend** implements RESTful APIs with comprehensive middleware for authentication (JWT), authorization (RBAC), and request validation. **PostgreSQL** serves as the primary data store with Prisma ORM ensuring type-safe database operations.

Infrastructure design emphasizes **DevOps best practices** including automated CI/CD pipelines via GitHub Actions, containerization for environment consistency, and secure deployment to Oracle Cloud Free Tier with Nginx handling SSL termination and reverse proxy duties. The architecture implements **security-by-design** principles with HTTP-only cookies, role-based access control, and input validation at multiple layers.

This Software Design Document details the architectural decisions, component specifications, data models, and deployment strategies that collectively realize the functional requirements outlined in the Software Requirements Specification.

Table of Contents

1. Application Architecture

1.1. Architectural Pattern Overview

WorkNexus follows a **Three-Tier Architecture** combined with:

- **Backend**: MVC (Model-View-Controller) pattern with Express.js
- **Frontend**: Component-Based Architecture with React.js
- **Infrastructure**: Microservices-like containerization using Docker

1.2. System Architecture Diagram

The WorkNexus system follows a three-tier architecture with microservices-inspired containerization. The architecture is designed for scalability, maintainability, and deployment flexibility using Docker containers. The system comprises four main layers: Presentation Layer (React.js frontend), Application Layer (Node.js/Express API), Data Layer (PostgreSQL database), and Infrastructure Layer (Docker containers orchestrated by Docker Compose).

Deployment is managed through a CI/CD pipeline using GitHub Actions, with the entire stack hosted on Oracle Cloud's Free Tier. The architecture implements role-based access control (RBAC) with JWT authentication, ensuring secure access for Admin, HR Managers, Project Managers, and Employees according to their privileges.

1.3 High-Level System Architecture Diagram

Figure 1 illustrates the complete system architecture of WorkNexus, depicting the flow from user interaction to data persistence. It visualizes the three-tier separation between the React frontend, Node.js/Express API layer, and PostgreSQL database, all orchestrated within Docker containers. The diagram also highlights external integrations such as the CI/CD pipeline and cloud hosting on Oracle Cloud Infrastructure.

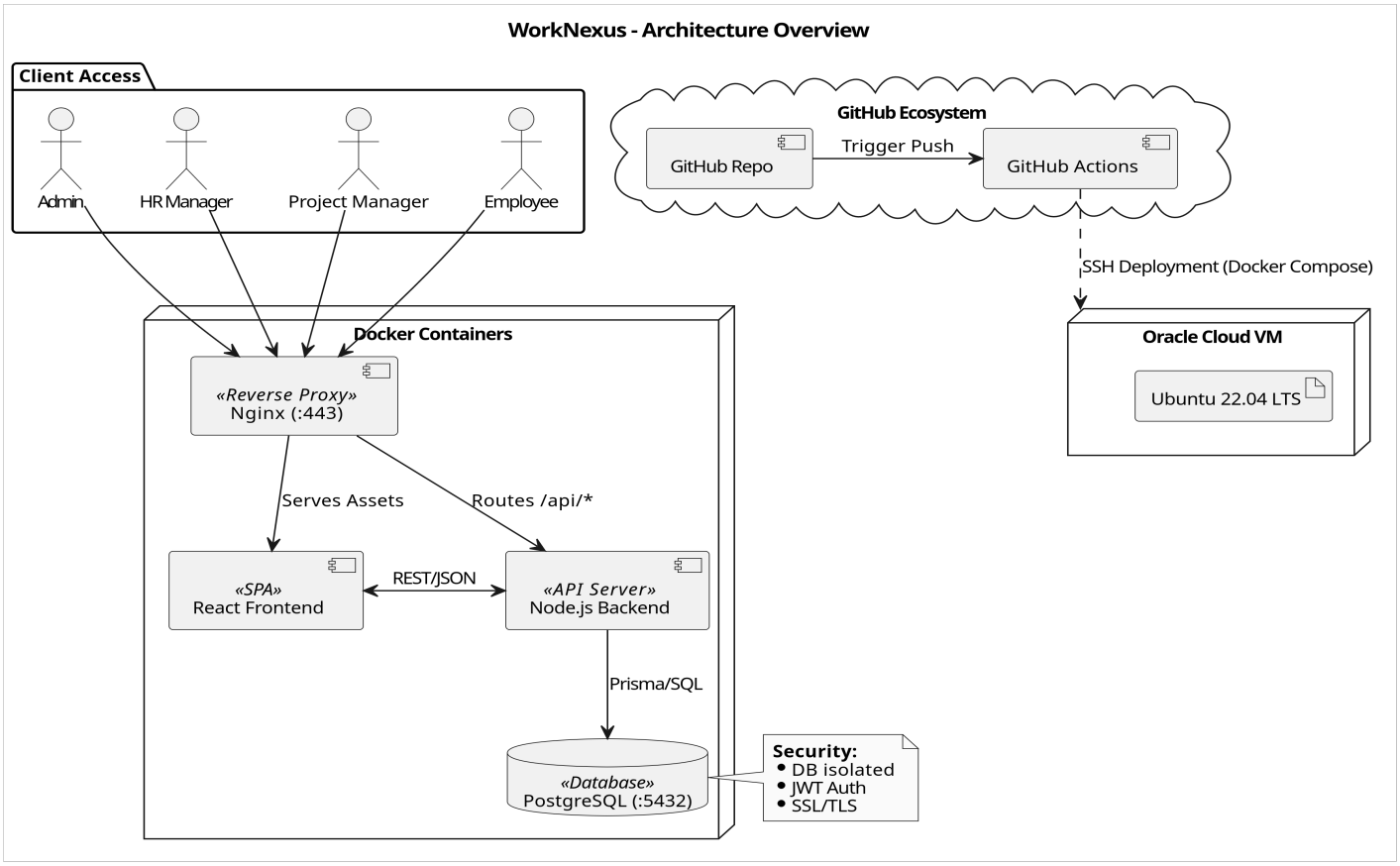

Figure 1: WorkNexus Complete System Architecture

1.4 Technology Stack

The technology stack diagram (Figure 2) provides a categorized breakdown of all tools and frameworks used across the WorkNexus system. It organizes technologies into frontend, backend, and infrastructure layers, showcasing the modern and cohesive selection that enables scalability and developer productivity. Each component is chosen for its reliability, community support, and alignment with the system's non-functional requirements, forming a robust foundation for the application's development and deployment lifecycle.

1.4.1 Front-end Technologies:

Table 1.1:Front-end Technologies Table

Framework	React.js 18+	UI component library
State Management	Zustand	Client state management
Server State	TanStack React Query	API data fetching/caching
UI Library	shadcn/ui	Production-ready components
Styling	TailwindCSS	Utility-first CSS
Charts	Recharts	Dashboard visualizations
Routing	React Router v6	Navigation
HTTP Client	Axios	API requests
Form Handling	React Hook Form + Zod	Form validation
Authentication	JWT tokens in HTTP-only cookies	Secure auth

1.4.2 Back-end Technologies:

Table 1.2: Back-end Technologies Table

Component	Technology	Purpose
Runtime	Node.js 20+ LTS	JavaScript runtime
Framework	Express.js	Web framework
Database ORM	Prisma	Type-safe database client
Authentication	JWT + RBAC Middleware	Role-based access control
Validation	Zod	Runtime validation
Security	Helmet, CORS, rate-limiter	Security middleware
Logging	Winston + Morgan	Structured logging
Containerization	Docker	Deployment consistency

1.4.3 Infrastructure & DevOps:

Table 1.3: Infrastructure and Deployment

Component	Technology	Purpose
Container Orchestration	Docker Compose	Multi-container management
Reverse Proxy	Nginx	SSL, routing, static files
CI/CD Pipeline	GitHub Actions	Automated testing & deployment
Monitoring	PM2 (for Node)	Process management
SSL Certificates	Let's Encrypt + Certbot	Free HTTPS
Cloud Provider	Oracle Cloud Free Tier	Production hosting
Database Hosting	Containerized PostgreSQL	Self-managed database

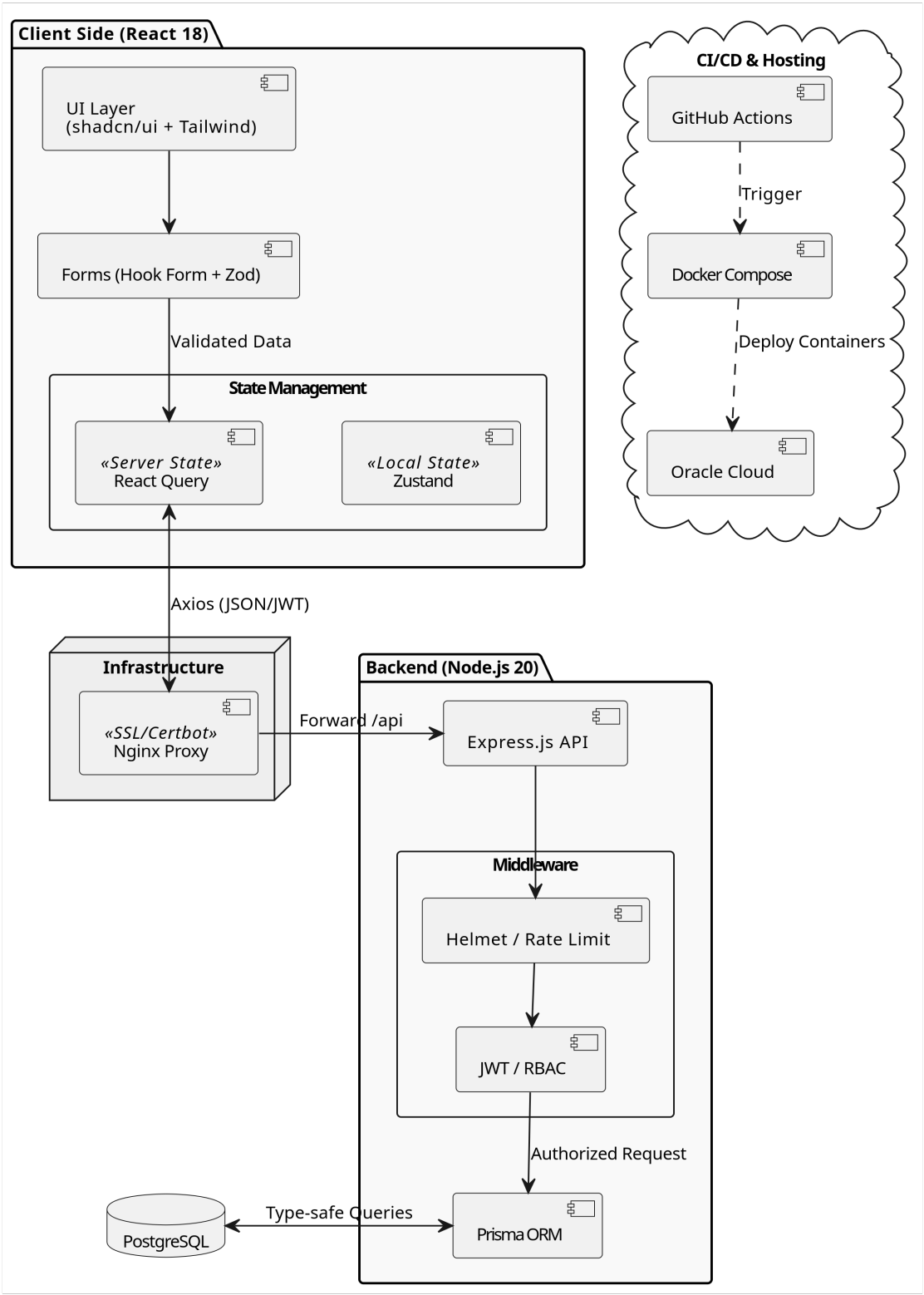

Figure 2: Technology Stack Components

1.5 Authentication & Authorization Flow

The system implements JWT-based authentication with role-based access control (RBAC). Upon login, users receive an HTTP-only cookie containing a JWT token. Subsequent API requests include this token, which is verified by middleware to ensure proper authorization based on user roles (Admin, HR, Project Manager, Employee).

The authentication and authorization flow (Figure 3) details the secure process from user login to resource access. It begins with credential submission, proceeds through JWT token generation and HTTP-only cookie storage, and concludes with middleware-based role verification for each API request. This structured approach ensures that only authenticated users with appropriate permissions can access specific features, maintaining system security and data integrity across all user interactions and role-based dashboards.

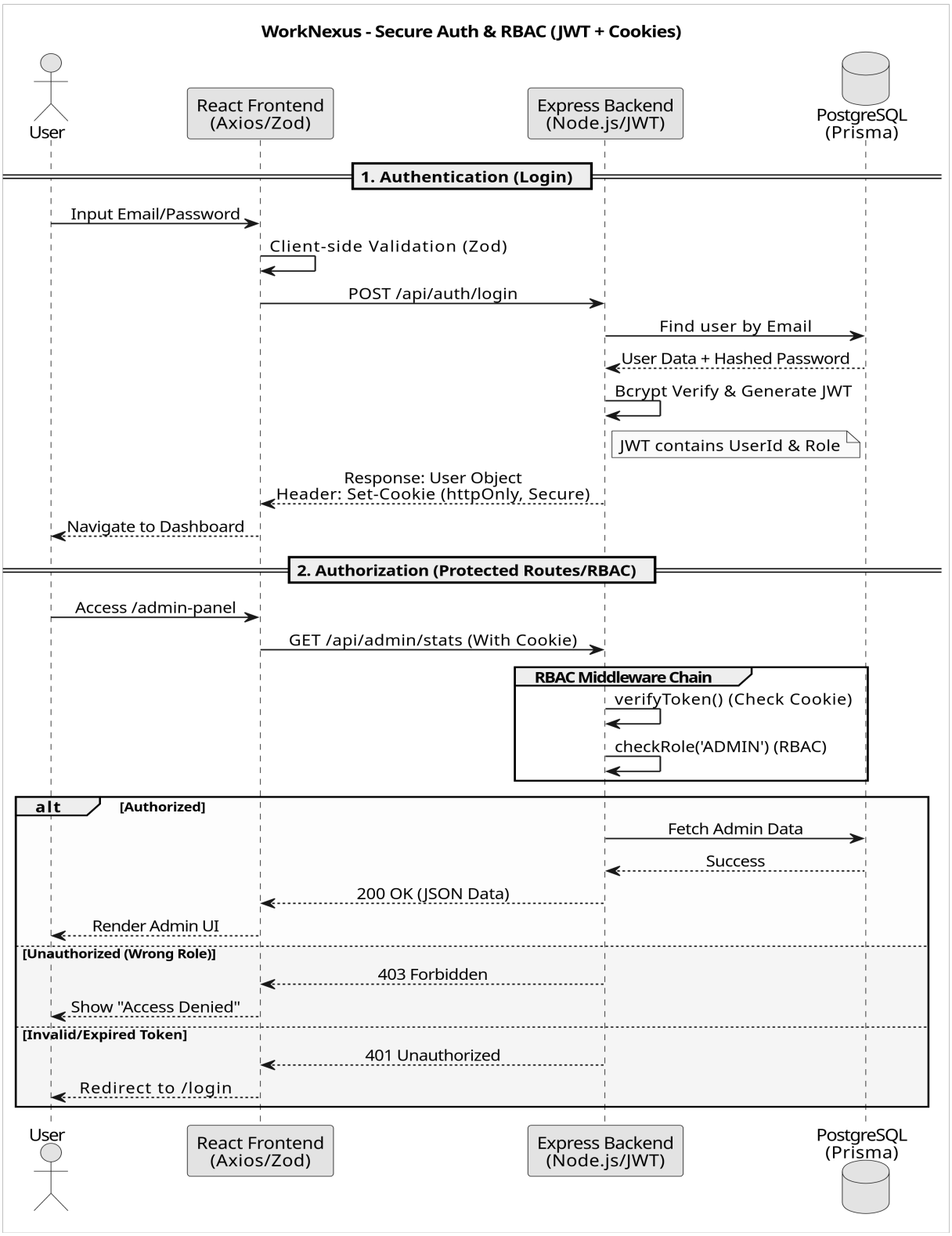

Figure 3: Authentication and Authorization Flow

1.5.1 API Specification

The system implements RESTful APIs with consistent response formats and error handling. All APIs are versioned and require authentication unless explicitly marked as public.

API Response Format:

```
{
  "success": true,
  "data": {},
  "message": "Operation successful",
  "timestamp": "2025-12-05T10:30:00Z"
}
```

Error Response Format:

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": {}
  },
  "timestamp": "2025-12-05T10:30:00Z"
}
```

Table 1.4: Core API Endpoints

Method	Endpoint	Description	Authentication	Role Required
POST	/api/v1/auth/login	User login	None	None
POST	/api/v1/auth/register	Create new user	JWT	Admin/HR
GET	/api/v1/users	List all users	JWT	Admin/HR
GET	/api/v1/users/:id	Get user details	JWT	Admin/HR/Self
PUT	/api/v1/users/:id	Update user	JWT	Admin/HR/Self
GET	/api/v1/employees	List employees	JWT	Admin/HR/PM
POST	/api/v1/employees	Create employee	JWT	HR
GET	/api/v1/projects	List projects	JWT	All roles

POST	/api/v1/projects	Create project	JWT	PM/Admin
GET	/api/v1/tasks	List tasks	JWT	All roles
POST	/api/v1/tasks	Create task	JWT	PM/Admin
PUT	/api/v1/tasks/:id/status	Update task status	JWT	Assigned Employee/PM
POST	/api/v1/tasks/:id/log-hours	Log work hours	JWT	Assigned Employee
GET	/api/v1/payroll	Calculate payroll	JWT	HR/Admin
GET	/api/v1/dashboard/:role	Role-based dashboard	JWT	All roles

JWT Token Structure:

json

```
{
  "userId": "123",
  "role": "admin",
  "email": "admin@example.com",
  "iat": 1672531200,
  "exp": 1672617600
}
```

Role Permissions Matrix:

Table 1.5: Roles and their Permissions

Permission	Admin	HR Manager	Project Manager	Employee
Create Users	√	√ (Employee only)	✗	✗
Manage Employees	√	√	✗	✗
Create Projects	√	✗	√	✗
Assign Tasks	√	✗	√	✗
Log Hours	√	✗	✗	√ (Own tasks)
View All Projects	√	√	√ (Assigned only)	√ (Assigned only)

Permission	Admin	HR Manager	Project Manager	Employee
Generate Payroll	√	√	✗	✗
System Configuration	√	✗	✗	✗

1.6 Deployment Architecture

The application is containerized using Docker, with each service running in isolated containers. Nginx acts as a reverse proxy handling SSL termination and routing. The entire stack is managed through Docker Compose and deployed automatically via GitHub Actions CI/CD pipeline to Oracle Cloud Free Tier.

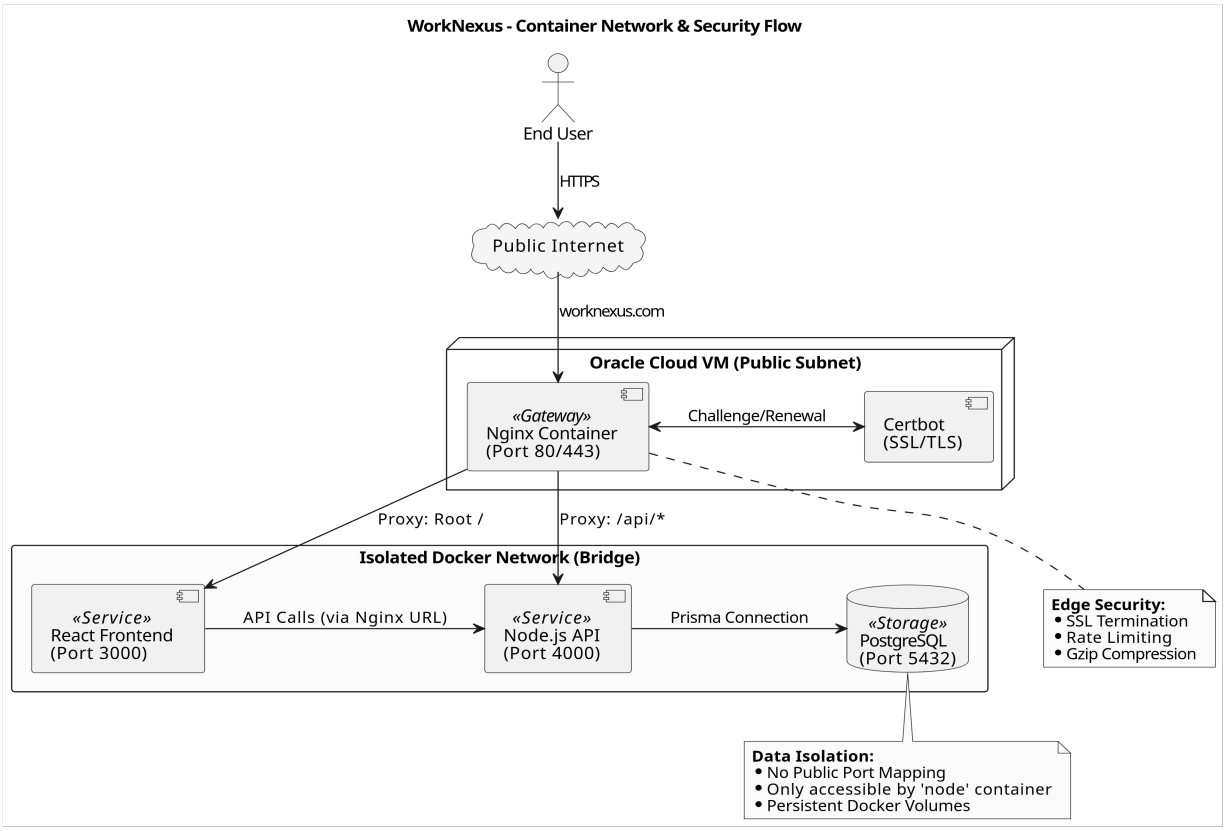

Figure 4: Container Network Architecture

CI/CD Pipeline Stages:

1. **Code Commit**: Developer pushes to main branch

2. **Automated Testing**: Run unit and integration tests
3. **Docker Build**: Create Docker images for each service
4. **Security Scan**: Vulnerability scanning of images
5. **Deployment**: Automated deployment to Oracle Cloud
6. **Health Check**: Verify deployment success

1.7 Component Design

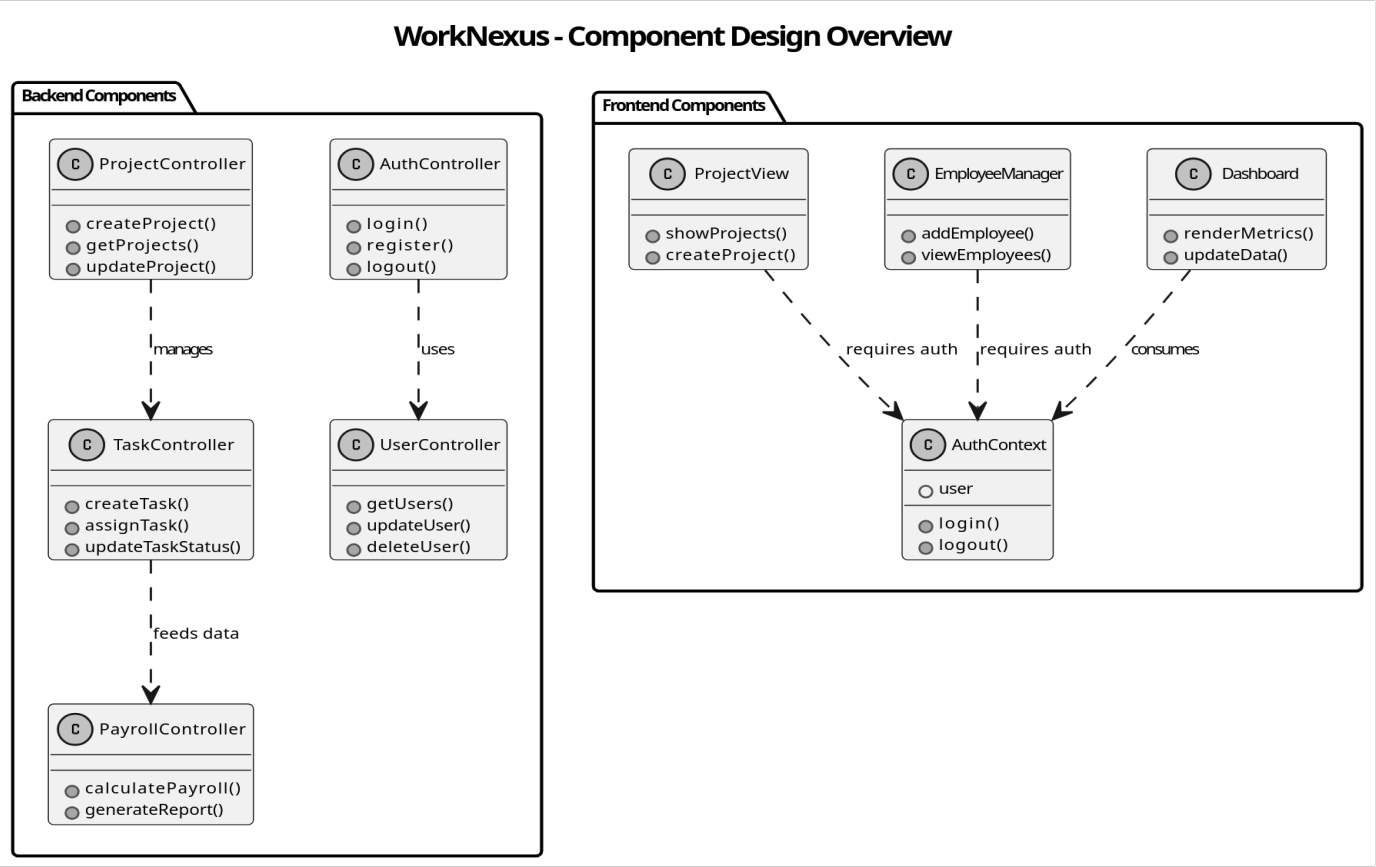

Figure 5: Key System Components

Frontend Component Structure:

```
src/
├── components/
│   ├── layout/          # Sidebar, Header
│   ├── ui/              # shadcn/ui based components
│   └── shared/          # Modal, Table
├── features/
│   ├── auth/
│   ├── dashboard/
│   ├── employees/
│   ├── projects/
│   ├── tasks/
│   ├── payroll/
│   └── clients/
├── hooks/
├── lib/
├── stores/
└── types/
```

Backend Service Structure:

```
src/
├── controllers/
│   ├── auth.controller.ts
│   ├── user.controller.ts
│   ├── employee.controller.ts
│   ├── project.controller.ts
│   ├── task.controller.ts
│   ├── payroll.controller.ts
│   └── client.controller.ts
├── middleware/
│   ├── auth.middleware.ts
│   ├── rbac.middleware.ts
│   └── validation.middleware.ts
├── services/
├── models/
├── routes/
├── utils/
└── config/
```

2. Data Model Schema

2.1 Database Design Overview

The database follows a normalized relational design with appropriate indexes and constraints. PostgreSQL 15 is used for its reliability, ACID compliance, and JSON support. Prisma ORM provides type-safe database access and migration management.

Database Design Principles:

- Third Normal Form (3NF) for data integrity
- Appropriate indexing for performance optimization
- Foreign key constraints for referential integrity
- Soft deletes using deleted_at timestamps
- Audit trails for critical operations

2.2 Entity-Relationship Diagram

The Entity-Relationship Diagram (Figure 6) models the core database structure of WorkNexus, defining tables, attributes, and relationships between entities such as Users, Employees, Projects, and Tasks. It reflects a normalized relational design that ensures data consistency, supports complex queries, and facilitates efficient data management. This schema serves as the blueprint for the PostgreSQL database, implemented using Prisma ORM to enforce type safety and relational integrity throughout the application.

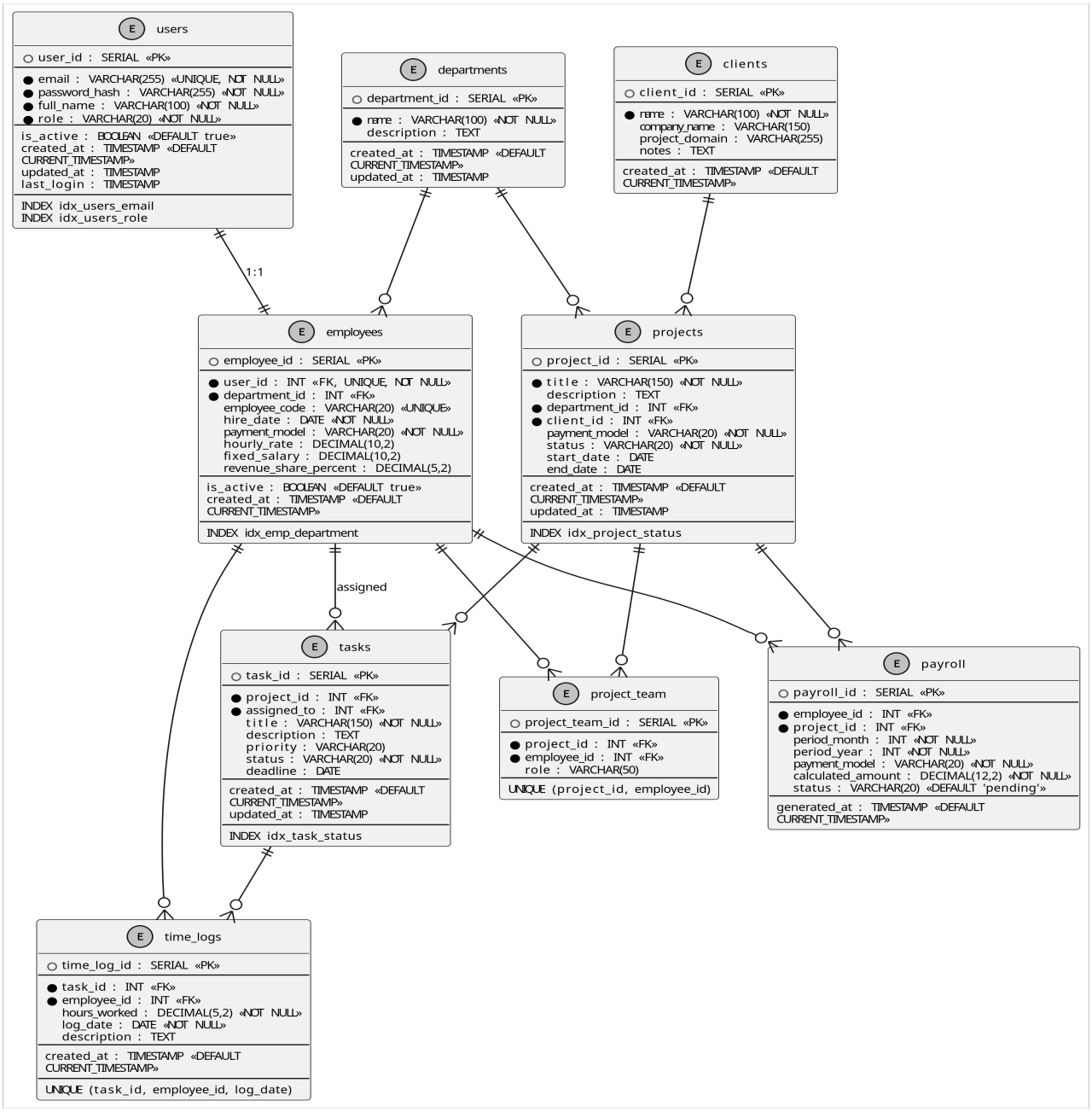

Figure 6: WorkNexus Entity Relationship Diagram

2.3 Data Dictionary

Table 2.1: Key Data Fields and Descriptions

Table	Field	Type	Description	Constraints
users	role	VARCHAR(20)	User role in system	admin, hr, pm, employee
employees	payment_model	VARCHAR(20)	Compensation method	fixed, hourly, revenue_share
projects	status	VARCHAR(20)	Project progress state	pending, active, completed, cancelled
tasks	priority	VARCHAR(20)	Task importance level	low, medium, high, critical
time_logs	hours	DECIMAL(5,2)	Work hours logged	Positive value
payroll	payment_status	VARCHAR(20)	Payment processing state	pending, processed, paid

3. User Interface Design

3.1 Design Philosophy

The WorkNexus interface follows **Material Design 3 principles** with a focus on usability, accessibility, and role-based user experience. The design prioritizes:

1. **Clarity Over Decoration**: Clean, functional interface with minimal distractions
2. **Role-Based Personalization**: Different visual cues and layouts for Admin, HR, PM, and Employee roles
3. **Progressive Disclosure**: Complex features revealed as needed, not all at once
4. **Accessibility First**: WCAG 2.1 AA compliance from the start
5. **Mobile-Responsive**: Single codebase that adapts from mobile to desktop

3.2 Design System Standards

3.2.1 Color System (Proposed)

The color system will follow a **semantic approach** where colors indicate meaning rather than arbitrary choices:

Table 3.1: Proposed Color Palette

Purpose	Token Name	Proposed Value	Usage
Primary	primary-500	#2563eb (Blue-600)	Main actions, navigation
Secondary	secondary-500	#64748b (Slate-500)	Secondary actions, borders
Success	success-500	#22c55e (Green-500)	Positive states, completions
Warning	warning-500	#f59e0b (Amber-500)	Warnings, pending actions
Error	error-500	#ef4444 (Red-500)	Errors, destructive actions
Background	background	#f8fafc (Slate-50)	Main background
Surface	surface	#ffffff (White)	Cards, modals, panels
Text Primary	text-primary	#0f172a (Slate-900)	Main text
Text Secondary	text-secondary	#475569 (Slate-600)	Secondary text

Color Application Rules:

1. **Role Differentiation**: Each user role may have subtle accent colors (Admin: indigo, HR: teal, PM: violet, Employee: blue)
2. **Status Indicators**: Project/task statuses will use consistent colors (Pending: gray, Active: blue, Completed: green, Blocked: red)
3. **Accessibility**: Minimum contrast ratio of 4.5:1 for normal text, 3:1 for large text
4. **Dark Mode**: Future implementation planned using CSS variables

3.2.2 Typography System (Proposed)

The typography system prioritizes readability and hierarchy:

Table 3.2: Proposed Typography Scale

Element	Font Family	Size	Weight	Line Height	Use Case
H1	Inter	2.5rem (40px)	700	1.2	Page titles
H2	Inter	2rem (32px)	600	1.3	Section headers
H3	Inter	1.5rem (24px)	600	1.4	Sub-section headers
H4	Inter	1.25rem (20px)	600	1.5	Card titles
Body Large	Inter	1.125rem (18px)	400	1.6	Lead paragraphs
Body Regular	Inter	1rem (16px)	400	1.7	Main content
Body Small	Inter	0.875rem (14px)	400	1.7	Secondary text
Caption	Inter	0.75rem (12px)	400	1.5	Labels, metadata
Button	Inter	0.875rem (14px)	500	1	Button text
Code	JetBrains Mono	0.875rem (14px)	400	1.5	Code snippets

Typography Rules:

1. **Font Stack**: Inter, -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif
2. **Responsive Scaling**: Base font size scales from 16px (desktop) to 14px (mobile)
3. **Line Length**: Maximum 70 characters per line for optimal readability
4. **Hierarchy**: Clear visual hierarchy using size, weight, and spacing

3.3 Component Library Architecture

The UI will be built using **shadcn/ui** as the foundation, extended with WorkNexus-specific components:

Table 3.3: Component Library Structure

Component Type	Base Library	Custom Extensions	Purpose
Layout	shadcn/ui	Role-based sidebar, Dashboard grid	Page structure
Data Display	shadcn/ui + Recharts	Gantt chart, Time tracking widget	Data visualization
Form Controls	shadcn/ui	Employee form, Project wizard	User input
Navigation	shadcn/ui	Role-based menu, Breadcrumbs	User navigation

Component Type	Base Library	Custom Extensions	Purpose
Feedback	shadcn/ui	Status indicators, Toast notifications	User feedback
Overlays	shadcn/ui	Task detail modal, Employee quick view	Contextual information

Component Design Principles:

1. **Consistency**: Same patterns across all features
2. **Reusability**: Components designed for multiple use cases
3. **Composability**: Small components combined into complex interfaces
4. **Accessibility**: ARIA labels, keyboard navigation, screen reader support

3.4 Screen Specifications

3.4.1 Layout Templates

Three main layout templates will be used:

1. **Dashboard Layout**: Sidebar navigation + main content area
2. **Form Layout**: Centered form with breadcrumbs and actions
3. **Detail Layout**: Master-detail view for complex data

3.4.2 Key Screen Wireframes *(Conceptual)*

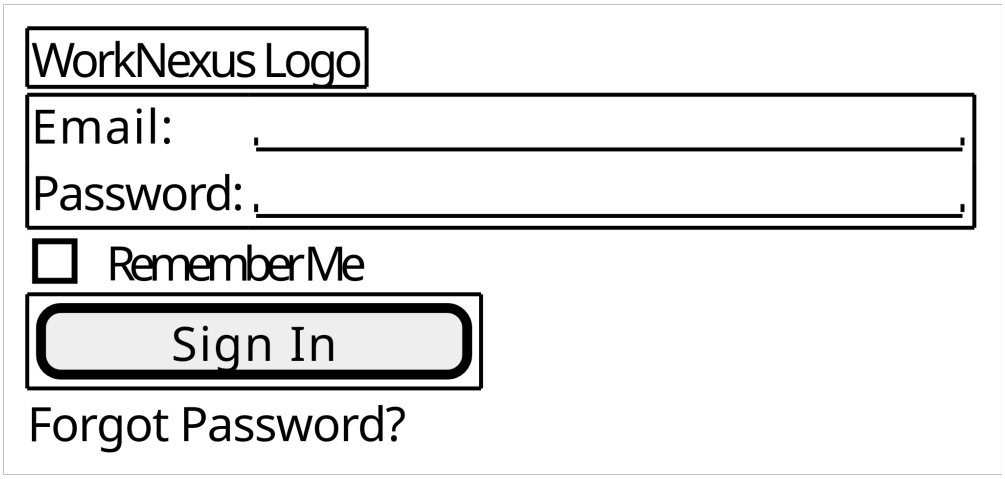

Figure 7: Login Screen Layout Concept

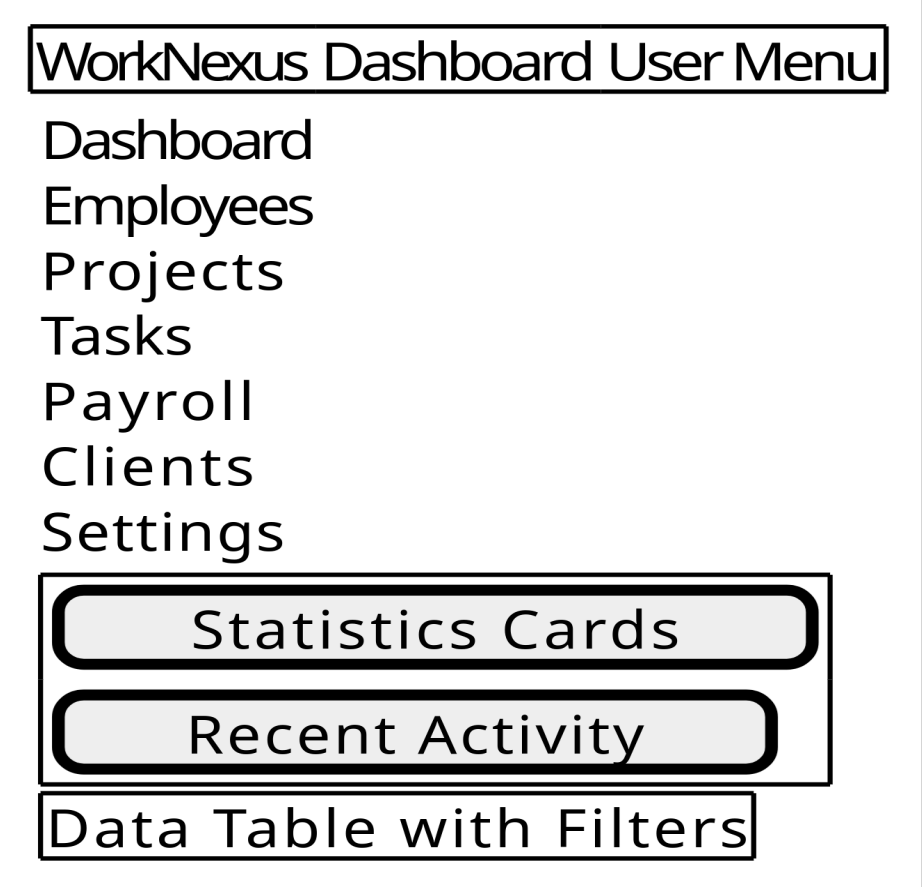

Figure 8: Dashboard Layout Concept

3.5 Responsive Design Strategy

3.5.1 Breakpoints

Table 3.4: Responsive Breakpoints

Device	Breakpoint	Layout Strategy
Mobile	< 640px	Single column, stacked navigation
Tablet	640px - 1024px	Two columns, collapsed sidebar
Desktop	1024px - 1280px	Full sidebar, multi-column content
Wide Desktop	> 1280px	Expanded layout, more visible columns

3.5.2 Mobile-First Implementation

1. **CSS Methodology**: Utility-first with TailwindCSS
2. **Component Adaptation**: Components modify behavior at breakpoints
3. **Touch Optimization**: Minimum 44×44px touch targets
4. **Performance**: Lazy loading for below-the-fold content

3.6 Accessibility Requirements

3.6.1 WCAG 2.1 AA Compliance

- **Perceivable**: Text alternatives, captions, adaptable content
- **Operable**: Keyboard accessible, enough time, no seizures
- **Understandable**: Readable, predictable, input assistance
- **Robust**: Compatible with assistive technologies

3.6.2 Specific Implementations

1. **Keyboard Navigation**: Full tab navigation, skip links
2. **Screen Reader**: ARIA labels, live regions for updates
3. **Color Contrast**: Minimum 4.5:1 for normal text
4. **Focus Management**: Logical focus order, visible focus indicators

3.7 Future Design Considerations

3.7.1 Planned Enhancements

1. **Dark Mode**: Color scheme inversion with CSS variables
2. **Theming**: Company branding customization
3. **High Contrast Mode**: Accessibility enhancement
4. **Internationalization**: RTL language support

3.7.2 Design Tokens

The design system will use CSS custom properties for maintainability:

```
//css
:root {
  /* Colors */
  --primary-500: #2563eb;
  --surface: #ffffff;

  /* Spacing */
  --spacing-unit: 0.25rem;

  /* Typography */
  --font-family: 'Inter', sans-serif;

  /* Breakpoints */
  --breakpoint-md: 640px;
}
```

4. System Deployment

4.1 Infrastructure Requirements

4.1.1 Minimum Server Specifications

Table 4.1: Infrastructure Requirements Specification

Component	Specification	Rationale	Verification Method
Processor	2+ cores (x86_64/ARM)	Concurrent request handling	lscpu command
Memory	4GB minimum, 8GB recommended	Container memory allocation	free -h command
Storage	20GB SSD minimum	OS, Docker images, logs	df -h command
Operating System	Ubuntu 22.04 LTS	Long-term support, stability	lsb_release -a
Network	Public IP with open ports 80/443	External accessibility	netstat -tulpn

Component	Specification	Rationale	Verification Method
Swap Space	2GB minimum	Memory overflow protection	swapon --show

4.1.2 Oracle Cloud Free Tier Configuration

The system leverages Oracle Cloud's Always Free Tier, providing a robust production environment without cost:

Table 4.2: Oracle Cloud Instance Configuration

Resource	Specification	Justification
Instance Type	VM.Standard.A1.Flex (ARM)	Cost-effective, modern ARM architecture
vCPU Cores	4 OCPUs (Arm-based)	Handles concurrent user sessions efficiently
Memory	24GB RAM	Ample for multiple containers and caching
Block Storage	200GB (Boot Volume)	Sufficient for OS, applications, and database
Network Bandwidth	1Gbps	High-speed data transfer for responsive UI
Availability Domain	Single AD (Free Tier)	Sufficient for MVP deployment
Public IP	Ephemeral (Static optional)	External access for web application

4.1.3 Software Dependencies

Table 4.3: Required Software Stack

Software	Version	Installation Command	Purpose
Docker Engine	24.0+	sudo apt install docker.io	Container runtime
Docker Compose	2.20+	sudo apt install docker-compose	Multi-container orchestration
Nginx	Latest stable	sudo apt install nginx	Reverse proxy, SSL termination
Certbot	Latest	sudo snap install certbot	SSL certificate management
Git	Latest	sudo apt install git	Version control operations

Software	Version	Installation Command	Purpose
UFW (Firewall)	Latest	sudo apt install ufw	Network security
Node.js	20 LTS	Via nvm or repository	Development environment

4.2 Container Configuration

4.2.1 Docker Compose Orchestration

The application uses Docker Compose to manage multi-container deployment with defined dependencies and networking:

```
yaml
# docker-compose.yml - Multi-container orchestration
version: '3.8'
services:
  postgres:
    image: postgres:15-alpine
    container_name: worknexus-db
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5

  backend:
    build: ./backend
    container_name: worknexus-api
    environment:
      DATABASE_URL: postgresql://${DB_USER}:${DB_PASSWORD}@postgres:5432/$
{DB_NAME}
      JWT_SECRET: ${JWT_SECRET}
```

```
      NODE_ENV: production
    ports:
      - "3000:3000"
    depends_on:
      postgres:
        condition: service_healthy
    restart: unless-stopped
    volumes:
      - ./backend:/app
      - /app/node_modules

  frontend:
    build: ./frontend
    container_name: worknexus-ui
    environment:
      REACT_APP_API_URL: /api
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    container_name: worknexus-proxy
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./frontend/build:/usr/share/nginx/html
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - backend
      - frontend
    restart: unless-stopped

volumes:
  postgres_data:
```

Key Configuration Notes:

- **Service Dependencies**: Backend waits for PostgreSQL health check
- **Volume Persistence**: Database data persists across container restarts
- **Environment Variables**: Sensitive data injected via .env file
- **Health Checks**: Automated service availability monitoring
- **Network Isolation**: Default bridge network for inter-container communication

4.2.2 Frontend Container Configuration

The React application uses a multi-stage build for optimized production deployment:

dockerfile

```
# Frontend Dockerfile - Optimized Production Build
FROM node:20-alpine AS builder

# Install dependencies
WORKDIR /app
COPY package*.json ./
RUN npm ci --silent

# Build application
COPY . .
RUN npm run build

# Production stage with Nginx
FROM nginx:alpine

# Copy built assets
COPY --from=builder /app/build /usr/share/nginx/html

# Custom Nginx configuration
COPY nginx.conf /etc/nginx/nginx.conf

# Security: Non-root user
RUN chown -R nginx:nginx /usr/share/nginx/html && \
    chmod -R 755 /usr/share/nginx/html

# Expose port
EXPOSE 80

# Start Nginx
CMD ["nginx", "-g", "daemon off;"]
```

4.2.3 Backend Container Configuration

The Node.js API server is optimized for production with security hardening:

dockerfile

```
# Backend Dockerfile - Secure Production Setup
FROM node:20-alpine AS builder
```

```
# Install dependencies
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

# Copy application code
COPY . .

# Generate Prisma client
RUN npx prisma generate

# Production stage
FROM node:20-alpine

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

WORKDIR /app

# Copy built dependencies
COPY --from=builder --chown=nodejs:nodejs /app/node_modules ./node_modules
COPY --from=builder --chown=nodejs:nodejs /app/package*.json ./
COPY --from=builder --chown=nodejs:nodejs /app/prisma ./prisma
COPY --chown=nodejs:nodejs . .

# Switch to non-root user
USER nodejs

# Expose port
EXPOSE 3000

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD node healthcheck.js

# Start application
CMD ["npm", "start"]
```

4.2.4 Nginx Reverse Proxy Configuration

Nginx handles SSL termination, load balancing, and static file serving:

nginx

```
# nginx.conf - Reverse Proxy Configuration
events {
    worker_connections 1024;
    use epoll;
    multi_accept on;
}

http {
    # MIME types
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Logging
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # Performance optimizations
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    client_max_body_size 10M;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css text/xml text/javascript
               application/javascript application/xml+rss
               application/json;

    # Backend upstream
    upstream backend {
        server backend:3000;
        keepalive 32;
    }

    # HTTP to HTTPS redirect
    server {
        listen 80;
        server_name worknexus.example.com;
        return 301 https://$server_name$request_uri;
    }

    # HTTPS server
    server {
```

```
    listen 443 ssl http2;
    server_name worknexus.example.com;

    # SSL certificates
    ssl_certificate /etc/nginx/ssl/fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/privkey.pem;

    # SSL configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    ssl_session_tickets off;

    # Frontend static files
    location / {
        root /usr/share/nginx/html;
        index index.html;
        try_files $uri $uri/ /index.html;

        # Cache static assets
        location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }
    }

    # Backend API proxy
    location /api {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_cache_bypass $http_upgrade;

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # Security headers
```

```
        add_header X-Frame-Options "DENY" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header Referrer-Policy "strict-origin-when-cross-origin" always;
        add_header Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline';" always;
    }
}
```

4.3 CI/CD Pipeline

4.3.1 Pipeline Architecture

The continuous integration and deployment pipeline automates testing, building, and deployment:

Figure 4.1: CI/CD Pipeline Flow Diagram

Code Commit → GitHub → Automated Tests → Docker Build →

Security Scan → Push to Registry → Deploy to Server →

Health Check → Notify Team

4.3.2 GitHub Actions Workflow

yaml

```
# .github/workflows/deploy.yml - CI/CD Pipeline Configuration
name: Deploy to Production

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

# Environment variables
env:
  REGISTRY: docker.io
  IMAGE_NAME: worknexus

jobs:
  # Test stage
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
```

```
      uses: actions/checkout@v3

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '20'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run unit tests
      run: npm test -- --coverage

    - name: Upload coverage report
      uses: codecov/codecov-action@v3

# Build and scan stage
build-and-scan:
  needs: test
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

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

    - name: Build and push frontend
      uses: docker/build-push-action@v4
      with:
        context: ./frontend
        push: true
        tags: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-frontend:latest
        cache-from: type=gha
        cache-to: type=gha,mode=max

    - name: Build and push backend
      uses: docker/build-push-action@v4
      with:
        context: ./backend
        push: true
```

```
          tags: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-backend:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max

      - name: Run security scan
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: '${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-backend:latest'
          format: 'sarif'
          output: 'trivy-results.sarif'

      - name: Upload security scan results
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: 'trivy-results.sarif'

  # Deployment stage
  deploy:
    needs: build-and-scan
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Deploy to Oracle Cloud
        uses: appleboy/ssh-action@v0.1.5
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ubuntu
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            # Pull latest images
            cd /opt/worknexus
            docker-compose pull

            # Backup database before update
            docker exec worknexus-db pg_dump -U $DB_USER $DB_NAME > backup_$(date +%Y%m%d_%H%M
%S).sql

            # Update containers
            docker-compose up -d --build

            # Clean up old images
            docker image prune -f

            # Verify deployment
            sleep 10
            curl -f http://localhost:3000/health || exit 1

            # Send notification
```

```
        echo "Deployment successful at $(date)"
```

4.4 Monitoring and Maintenance

4.4.1 Health Monitoring Endpoints

Table 4.4: Health Check Endpoints

Endpoint	Method	Response Format	Purpose
/health	GET	JSON	Overall system health
/metrics	GET	Prometheus format	Performance metrics
/version	GET	JSON	Application version info
/readiness	GET	JSON	Service readiness status
/liveness	GET	JSON	Service liveness check

Sample Health Check Implementation:

javascript

```
// healthcheck.js - Health monitoring endpoint
app.get('/health', (req, res) => {
  const healthcheck = {
    status: 'UP',
    timestamp: new Date().toISOString(),
    services: {
      database: checkDatabase(),
      memory: checkMemoryUsage(),
      disk: checkDiskSpace()
    },
    version: process.env.npm_package_version
  };

  const status = healthcheck.services.database.status === 'UP' ? 200 : 503;
  res.status(status).json(healthcheck);
});
```

4.4.2 Logging Strategy

Table 4.5: Logging Configuration

Log Type	Tool	Retention	Storage	Purpose
Application Logs	Winston	30 days	File + CloudWatch	Debugging, audit
Access Logs	Nginx	14 days	Rotated files	Security analysis
Error Logs	Sentry	90 days	Cloud service	Error tracking
Audit Logs	Custom	1 year	Encrypted DB	Compliance

4.4.3 Backup Strategy

bash

```
# backup.sh - Automated Backup Script
#!/bin/bash
BACKUP_DIR="/opt/backups"
DATE=$(date +%Y%m%d_%H%M%S)

# Database backup
docker exec worknexus-db pg_dump -U $DB_USER $DB_NAME > \
  $BACKUP_DIR/db_backup_$DATE.sql

# Configuration backup
tar -czf $BACKUP_DIR/config_backup_$DATE.tar.gz \
  /opt/worknexus/nginx/nginx.conf \
  /opt/worknexus/docker-compose.yml \
  /opt/worknexus/.env

# Upload to cloud storage (optional)
aws s3 cp $BACKUP_DIR/db_backup_$DATE.sql s3://worknexus-backups/
aws s3 cp $BACKUP_DIR/config_backup_$DATE.tar.gz s3://worknexus-backups/

# Cleanup old backups (keep 7 days)
find $BACKUP_DIR -name "*.sql" -mtime +7 -delete
find $BACKUP_DIR -name "*.tar.gz" -mtime +7 -delete
```

4.4.4 Maintenance Schedule

Table 4.6: Maintenance Tasks Schedule

Task	Frequency	Duration	Impact	Automation
SSL Renewal	90 days	5 minutes	None	Certbot auto-renew
Security Updates	Weekly	15 minutes	Brief restart	Unattended upgrades

Task	Frequency	Duration	Impact	Automation
Database Optimization	Monthly	30 minutes	Read-only mode	Scheduled job
Log Rotation	Daily	1 minute	None	Logrotate
Backup Verification	Weekly	10 minutes	None	Automated script
Container Cleanup	Daily	2 minutes	None	Cron job

4.5 Disaster Recovery Plan

4.5.1 Recovery Procedures

1. **Database Corruption**:

bash

```
# Restore from latest backup
cat latest_backup.sql | docker exec -i worknexus-db psql -U $DB_USER $DB_NAME
```

Container Failure:

bash

```
# Restart all services
cd /opt/worknexus
docker-compose down
docker-compose up -d
```

3. **Server Failure**:
    - Provision new Oracle Cloud instance
    - Restore from S3 backups
    - Update DNS records

4.5.2 Recovery Time Objectives

Scenario	RTO (Recovery Time Objective)	RPO (Recovery Point Objective)
Database failure	15 minutes	1 hour
Application failure	5 minutes	None
Server failure	1 hour	24 hours
Data center failure	4 hours	24 hours

5. Testing Strategy

5.1 Testing Approach

The testing strategy follows the testing pyramid with emphasis on automated testing at all levels:

Testing Levels:

1. **Unit Testing**: Individual components and functions
2. **Integration Testing**: Component interactions and API endpoints
3. **End-to-End Testing**: Complete user workflows
4. **Performance Testing**: System load and response times
5. **Security Testing**: Vulnerability assessment and penetration testing

Test Automation Tools:

- **Jest**: Unit and integration testing
- **React Testing Library**: Component testing
- **Supertest**: API endpoint testing
- **Cypress**: End-to-end testing
- **Lighthouse**: Performance auditing

5.2 Test Cases

Authentication Test Cases:

1. Valid user login with correct credentials
2. Invalid login attempts with wrong credentials

3. Password reset functionality
4. Session timeout and automatic logout
5. Concurrent session management

Employee Management Test Cases:

1. Create new employee with valid data
2. Update employee information
3. Department assignment validation
4. Employee deactivation and reactivation
5. Search and filter functionality

Project Management Test Cases:

1. Create project with all required fields
2. Assign team members to projects
3. Project status transitions
4. Budget validation and alerts
5. Timeline scheduling constraints

Task Management Test Cases:

1. Task creation and assignment
2. Time logging with validation
3. Task status updates
4. Priority and deadline management
5. Task dependencies and relationships

Payroll Test Cases:

1. Hourly rate calculation accuracy
2. Fixed salary proration
3. Revenue share percentage calculations
4. Payroll report generation
5. Data export functionality

5.3 Performance Testing

Load Testing Scenarios:

1. **Concurrent User Login**: 50+ users logging in simultaneously
2. **Dashboard Loading**: Multiple users accessing dashboards
3. **Report Generation**: Payroll report generation under load
4. **Data Export**: CSV/PDF export with large datasets

Performance Benchmarks:

- Page load time: < 3 seconds
- API response time: < 500ms for 95% of requests
- Database query time: < 100ms for common queries
- Concurrent users: Support for 50+ simultaneous users

Optimization Strategies:

- Database query optimization with indexes
- Frontend code splitting and lazy loading
- API response caching
- CDN for static assets
- Database connection pooling

6. Security Considerations

6.1 Authentication Mechanism

JWT Implementation:

- Token expiration: 24 hours for regular users, 8 hours for admin
- Refresh token mechanism for extended sessions
- HTTP-only cookies for token storage
- CSRF protection with same-site cookie policy

Password Security:

- Minimum password length: 8 characters
- Password complexity requirements
- Bcrypt hashing with appropriate work factor

- Password history prevention (last 5 passwords)
- Account lockout after 5 failed attempts

6.2 Authorization Framework

Role-Based Access Control (RBAC):

- Four distinct roles with clear permission boundaries
- Permission inheritance through role hierarchy
- Context-based authorization for resource access
- Audit logging for all authorization decisions

API Authorization:

- Route-level permission checking
- Resource ownership validation
- Query parameter sanitization
- Rate limiting per user and IP

6.3 Data Protection

Data Encryption:

- TLS 1.2+ for data in transit
- Database encryption at rest
- Sensitive field encryption (passwords, payment info)
- Secure key management with environment variables

Input Validation:

- Server-side validation for all inputs
- SQL injection prevention with parameterized queries
- XSS protection with output encoding
- File upload validation and virus scanning

Data Privacy:

- GDPR compliance for user data
- Data minimization principles
- User data export and deletion capabilities

- Privacy policy integration

6.4 Compliance Measures

Security Headers:

- Content Security Policy (CSP)
- HTTP Strict Transport Security (HSTS)
- X-Frame-Options for clickjacking protection
- Referrer-Policy for privacy

Vulnerability Management:

- Regular dependency updates
- Security scanning in CI/CD pipeline
- Penetration testing schedule
- Security patch management process

Incident Response:

- Security incident reporting procedure
- Data breach notification process
- Forensic analysis capabilities
- Recovery and remediation plans

7. References

1. IEEE Standard 830-1998 - Software Requirements Specification
2. React Documentation - https://react.dev
3. Node.js Documentation - https://nodejs.org/en/docs
4. PostgreSQL Documentation - https://www.postgresql.org/docs
5. Docker Documentation - https://docs.docker.com
6. Prisma Documentation - https://www.prisma.io/docs
7. TailwindCSS Documentation - https://tailwindcss.com/docs
8. JWT RFC 7519 - https://tools.ietf.org/html/rfc7519
9. OWASP Security Guidelines - https://owasp.org
10. Material Design Guidelines - https://material.io/design
11. WCAG 2.1 Accessibility Guidelines - https://www.w3.org/TR/WCAG21/
12. Oracle Cloud Documentation - https://docs.oracle.com/en/cloud
13. GitHub Actions Documentation - https://docs.github.com/en/actions
14. Nginx Documentation - https://nginx.org/en/docs/
15. ISO/IEC 27001 Information Security Management

Academic References:

16. Sommerville, I. (2015). Software Engineering. 10th Edition.
17. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice.
18. Fowler, M. (2018). Refactoring: Improving the Design of Existing Code.
19. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship.

Tools and Libraries:

20. Jest Testing Framework - https://jestjs.io
21. Cypress Testing Framework - https://www.cypress.io
22. Recharts Charting Library - https://recharts.org
23. shadcn/ui Component Library - https://ui.shadcn.com
24. Zod Validation Library - https://zod.dev