AIBrainFrame Complete Implementation Guide

Overview

You now have complete, enterprise-ready applications for:

- **Web Application**: React-based responsive web interface
- **Mobile Application**: React Native app for iOS and Android
- **Backend API**: FastAPI with Ollama AI integration

Implementation Steps

1. Web Application Setup

```bash
# Create web directory
mkdir -p /opt/aibrainframe/web
cd /opt/aibrainframe/web

# Save the HTML file as index.html
# Copy the complete web application code from the artifact

# For development - serve with Python
python -m http.server 8080

# Access at: http://192.168.1.70:8080
```

Production Web Deployment:

```bash
```

```
# Install Nginx
sudo apt update
sudo apt install nginx

# Configure Nginx for AIBrainFrame
sudo nano /etc/nginx/sites-available/aibrainframe

# Add this configuration:
server {
    listen 80;
    server_name your-domain.com;
    root /opt/aibrainframe/web;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api/ {
        proxy_pass http://127.0.0.1:8000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

# Enable the site
sudo ln -s /etc/nginx/sites-available/aibrainframe /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

2. Mobile Application Setup

bash

```
# Create mobile app directory
mkdir -p /opt/aibrainframe/mobile
cd /opt/aibrainframe/mobile

# Initialize React Native project
npx react-native init AIBrainFrameMobile
cd AIBrainFrameMobile

# Replace default files with provided code
# Copy App.js, package.json, and config files from artifacts

# Install dependencies
npm install

# Install iOS dependencies (macOS only)
cd ios && pod install && cd ..
```

Android Development Setup:

```
bash

# Install Android dependencies
npm install

# Start Metro bundler
npm start

# Run on Android device/emulator
npm run android
```

iOS Development Setup (macOS only):

```
bash

# Install iOS dependencies
cd ios
pod install
cd ..

# Run on iOS device/simulator
npm run ios
```

3. Backend API Configuration Updates

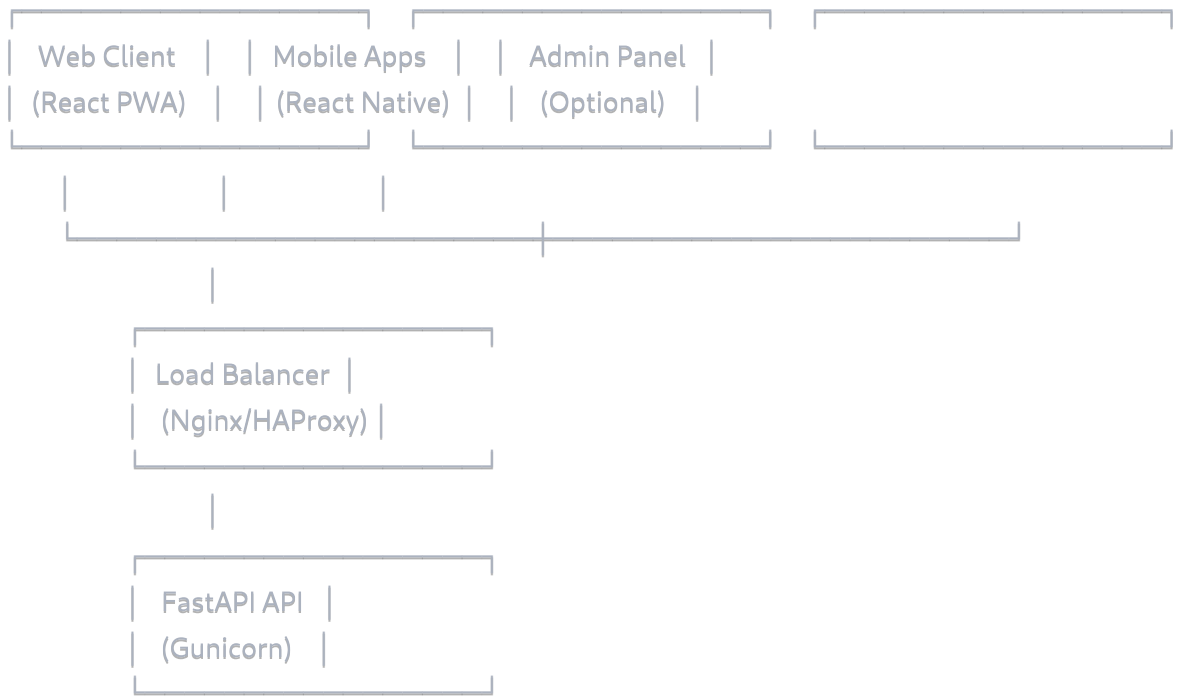
Update your FastAPI CORS settings for production:

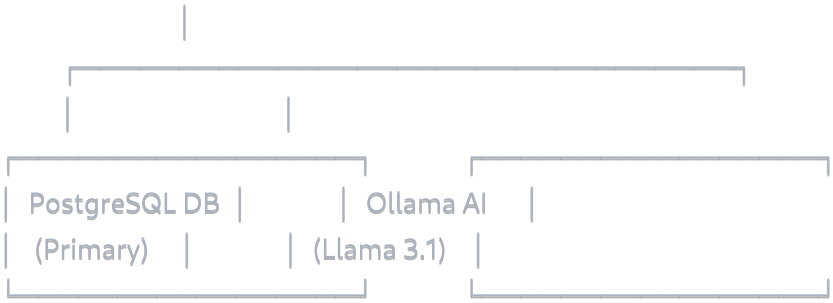
```bash
nvim app/main.py
```

Update CORS middleware:

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "http://localhost:3000",
        "http://192.168.1.70:8080",
        "https://your-domain.com",
        "*"  # Remove in production
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

4. Production Deployment Architecture





5. Production API Deployment

```bash
# Install Gunicorn for production
pip install gunicorn

# Create systemd service
sudo nano /etc/systemd/system/aibrainframe.service

# Add service configuration:
[Unit]
Description=AIBrainFrame FastAPI
After=network.target

[Service]
User=csprinks
Group=csprinks
WorkingDirectory=/opt/aibrainframe
Environment="PATH=/opt/aibrainframe/venv/bin"
ExecStart=/opt/aibrainframe/venv/bin/gunicorn -w 4 -k uvicorn.workers.UvicornWorker -b 0.0.0.0:8000 ap
Restart=always

[Install]
WantedBy=multi-user.target

# Enable and start service
sudo systemctl enable aibrainframe
sudo systemctl start aibrainframe
```

6. Mobile App Store Deployment

Android (Google Play Store):

```bash
```

```
# Generate signing key
keytool -genkey -v -keystore aibrainframe-release-key.keystore -alias aibrainframe -keyalg RSA -keysize 20

# Build release APK
cd android
./gradlew assembleRelease

# APK location: android/app/build/outputs/apk/release/app-release.apk
```

iOS (Apple App Store - macOS only):

```
bash

# Open Xcode project
open ios/AIBrainFrameMobile.xcworkspace

# Configure signing in Xcode
# Archive and upload to App Store Connect
```

7. Security and Performance

SSL/HTTPS Setup:

```
bash

# Install Certbot for Let's Encrypt
sudo apt install certbot python3-certbot-nginx

# Get SSL certificate
sudo certbot --nginx -d your-domain.com

# Auto-renewal
sudo crontab -e
# Add: 0 12 * * * /usr/bin/certbot renew --quiet
```

Database Security:

```
bash
```

```
# PostgreSQL security hardening
sudo nano /etc/postgresql/16/main/postgresql.conf

# Update settings:
ssl = on
password_encryption = scram-sha-256
log_statement = 'mod'
log_min_duration_statement = 1000
```

Firewall Configuration:

```
bash

# Configure UFW
sudo ufw allow 22/tcp     # SSH
sudo ufw allow 80/tcp     # HTTP
sudo ufw allow 443/tcp    # HTTPS
sudo ufw allow 8000/tcp   # API (internal)
sudo ufw --force enable
```

8. Monitoring and Maintenance

Log Management:

```
bash

# Create log rotation
sudo nano /etc/logrotate.d/aibrainframe

# Add configuration:
/opt/aibrainframe/logs/*.log {
    daily
    missingok
    rotate 52
    compress
    delaycompress
    notifempty
    create 644 csprinks csprinks
}
```

Health Monitoring:

```
bash
```

```
# Create monitoring script
nano /opt/aibrainframe/scripts/health_check.sh

#!/bin/bash
curl -f http://localhost:8000/health || systemctl restart aibrainframe

# Add to crontab
crontab -e
# Add: */5 * * * * /opt/aibrainframe/scripts/health_check.sh
```

9. Feature Enhancements

Push Notifications (Mobile):

- Integrate Firebase Cloud Messaging for Android
- Configure Apple Push Notification Service for iOS
- Implement notification triggers for urgent system alerts

Offline Support:

- Implement Redux/AsyncStorage for offline message queuing
- Cache critical troubleshooting guides locally
- Sync data when connection restored

Advanced LBOB Features:

- Voice interaction using React Native Voice
- Camera integration for equipment photo recognition
- AR overlay for equipment identification

10. Testing and Quality Assurance

Automated Testing:

```
bash
```

```
# Backend API tests
pytest app/tests/

# Mobile app testing
npm test

# End-to-end testing
npm install --save-dev detox
detox test
```

Performance Testing:

```bash
# Load testing with Artillery
npm install -g artillery
artillery quick --count 10 --num 100 http://localhost:8000/health
```

Maintenance Schedule

Daily:

- Monitor system logs
- Check database performance
- Verify API health endpoints

Weekly:

- Update dependencies
- Review user feedback
- Analyze usage patterns

Monthly:

- Security updates
- Database optimization
- Performance tuning
- Backup verification

Support and Documentation

User Documentation:

- Create user guides for technicians
- Video tutorials for LBOB interaction
- Troubleshooting FAQ

Developer Documentation:

- API documentation with Swagger
- Code architecture diagrams
- Deployment procedures

This implementation provides an enterprise-grade AI troubleshooting assistant that scales with your technician workforce while maintaining security, performance, and reliability standards.