# AiiA Application Migration Guide for Bolt.New Platform

## Overview

This guide provides comprehensive instructions for migrating the AiiA (AI Investment Assistant) application to the Bolt.New platform. AiiA is a sophisticated Next.js 14 application with TypeScript, featuring AI-powered investment analysis, real-time trading capabilities, user authentication, and comprehensive portfolio management.

### Application Architecture

- **Frontend**: Next.js 14 with TypeScript, Tailwind CSS, Radix UI components
- **Backend**: Next.js API routes with serverless functions
- **Database**: PostgreSQL with Prisma ORM
- **Authentication**: NextAuth.js with multiple providers
- **AI Integration**: OpenAI API for investment analysis
- **Trading**: Alpaca API for stock/crypto trading
- **Real-time Data**: Multiple financial data providers (Alpha Vantage, Finnhub, News API)

## Pre-Migration Checklist

### 1. Project Structure Analysis

The AiiA application contains:

```
aiia-2.0/
├── app/                          # Main Next.js application
│   ├── app/                      # Next.js 14 app directory
│   ├── components/               # React components
│   ├── hooks/                    # Custom React hooks
│   ├── lib/                      # Utility libraries
│   ├── prisma/                   # Database schema and migrations
│   ├── public/                   # Static assets
│   ├── scripts/                  # Database seeding scripts
│   ├── package.json              # Dependencies and scripts
│   ├── .env                      # Environment variables
│   ├── next.config.js            # Next.js configuration
│   ├── tailwind.config.ts        # Tailwind CSS configuration
│   └── tsconfig.json             # TypeScript configuration
└── docs/                         # Documentation
```

### 2. Dependencies Overview

**Key Production Dependencies:**
- Next.js 14.2.28 (App Router)
- React 18.2.0
- Prisma 6.7.0 with PostgreSQL
- NextAuth.js 4.24.11
- Tailwind CSS 3.3.3

- Radix UI components
- Alpaca Trading API
- OpenAI API integration
- Multiple financial data APIs

**Development Dependencies:**
- TypeScript 5.2.2
- ESLint with Next.js config
- PostCSS and Tailwind

## Migration Steps

### Step 1: Prepare Files for Migration

#### Files to Include:

```
# Essential application files
app/
├── app/                    # Next.js app directory (all files)
├── components/             # All React components
├── hooks/                  # Custom hooks
├── lib/                    # Utility functions
├── prisma/                 # Database schema
├── public/                 # Static assets
├── scripts/                # Database scripts
├── package.json            # Dependencies
├── next.config.js          # Next.js config
├── tailwind.config.ts      # Tailwind config
├── tsconfig.json           # TypeScript config
├── postcss.config.js       # PostCSS config
└── components.json         # Shadcn/ui config
```

#### Files to Exclude:

```
# These should NOT be migrated
.env                        # Contains sensitive keys
node_modules/               # Will be reinstalled
.next/                      # Build artifacts
.build/                     # Build artifacts
package-lock.json           # Will be regenerated
tsconfig.tsbuildinfo        # Build cache
```

### Step 2: Environment Variables Setup

Create a new `.env.local` file in Bolt.New with the following variables:

### Required Environment Variables:

```
# Database Configuration
DATABASE_URL="your_postgresql_connection_string"

# Authentication
NEXTAUTH_URL="your_bolt_new_app_url"
NEXTAUTH_SECRET="generate_new_secret_key"

# AI Services
OPENAI_API_KEY="your_openai_api_key"
ABACUSAI_API_KEY="your_abacus_ai_key"

# Financial Data APIs
ALPHADVANTAGE_API_KEY="your_alpha_vantage_key"
NEWS_API_KEY="your_news_api_key"
FINNHUB_API_KEY="your_finnhub_key"

# Trading APIs
ALPACA_API_KEY_ID="your_alpaca_key_id"
ALPACA_API_SECRET_KEY="your_alpaca_secret"
ALPACA_TRADE_BASE_URL="https://paper-api.alpaca.markets"
ALPACA_DATA_BASE_URL="https://data.alpaca.markets"

# Supabase (if using)
SUPABASE_URL="your_supabase_url"
SUPABASE_ANON_KEY="your_supabase_anon_key"
SUPABASE_SERVICE_ROLE_KEY="your_supabase_service_key"

# Payment Processing (if needed)
STRIPE_SECRET_KEY="your_stripe_secret_key"
STRIPE_PUBLISHABLE_KEY="your_stripe_publishable_key"

# Application Configuration
API_BASE_URL="your_bolt_new_app_url"
VITE_API_BASE_URL="your_bolt_new_app_url"
```

### Environment Variable Security:

- **Never commit** the `.env` file to version control
- Use Bolt.New's environment variable management interface
- Generate new secrets for NEXTAUTH_SECRET
- Update all URLs to match your Bolt.New deployment

## Step 3: Database Migration Strategy

### Option A: PostgreSQL via Supabase (Recommended)

1. **Create Supabase Project:**
   `bash`
   ```
   # Visit https://supabase.com
   # Create new project
   # Note down the connection details
   ```

2. **Update DATABASE_URL:**
   `bash`
   ```
   DATABASE_URL="postgresql://postgres:[password]@[host]:5432/[database]"
   ```

3. **Run Prisma Migrations:**
   ```bash
   npx prisma migrate deploy
   npx prisma generate
   npx prisma db seed  # Optional: seed with initial data
   ```

### Option B: External PostgreSQL

1. Set up PostgreSQL instance (AWS RDS, DigitalOcean, etc.)
2. Update DATABASE_URL with connection string
3. Ensure network access from Bolt.New platform

## Step 4: Bolt.New Platform Deployment

### Method 1: Direct Upload to Bolt.New

1. **Prepare Project Archive:**
   ```bash
   # Create a zip file with essential files only
   cd aiia-2.0/app
   zip -r aiia-bolt-migration.zip . \
     -x "node_modules/*" ".next/*" ".build/*" ".env" "package-lock.json"
   ```

2. **Upload to Bolt.New:**
   - Visit https://bolt.new
   - Create new project
   - Upload the zip file
   - Bolt.New will automatically detect Next.js project

### Method 2: GitHub Integration

1. **Push to GitHub:**
   ```bash
   git init
   git add .
   git commit -m "Initial AiiA migration to Bolt.New"
   git remote add origin https://github.com/yourusername/aiia-bolt.git
   git push -u origin main
   ```

2. **Connect to Bolt.New:**
   - In Bolt.New, select “Import from GitHub”
   - Authorize GitHub access
   - Select your repository

## Step 5: Platform-Specific Configuration

### Next.js Configuration Updates:

```
// next.config.js - Update for Bolt.New
const nextConfig = {
  output: 'standalone',  // For Bolt.New deployment
  images: {
    unoptimized: true,   // Required for static deployment
    domains: ['your-domain.com'] // Add your domains
  },
  eslint: {
    ignoreDuringBuilds: true,
  },
  typescript: {
    ignoreBuildErrors: false,
  },
  experimental: {
    serverComponentsExternalPackages: ['@prisma/client']
  }
};
```

### Package.json Scripts:

```
{
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "db:migrate": "prisma migrate deploy",
    "db:generate": "prisma generate",
    "db:seed": "tsx --require dotenv/config scripts/seed.ts",
    "postinstall": "prisma generate"
  }
}
```

## Step 6: Build and Deployment Process

### Automated Build Process:

1. **Install Dependencies:**
   ```
   bash
     npm install
     # or
     pnpm install
   ```

2. **Generate Prisma Client:**
   ```
   bash
     npx prisma generate
   ```

3. **Run Database Migrations:**
   ```
   bash
     npx prisma migrate deploy
   ```

4. **Build Application:**
   ```
   bash
     npm run build
   ```

5. **Deploy:**
   - Bolt.New handles deployment automatically
   - Monitor build logs for any issues

### Step 7: Post-Migration Verification

#### Functionality Checklist:

- [ ] Application loads without errors
- [ ] Database connection established
- [ ] User authentication working
- [ ] API routes responding correctly
- [ ] Trading functionality operational
- [ ] AI analysis features working
- [ ] Real-time data feeds active
- [ ] Payment processing (if applicable)

#### Testing Procedures:

1. **User Registration/Login:**
   - Test new user signup
   - Verify email authentication
   - Check user session persistence

2. **Core Features:**
   - Portfolio dashboard loading
   - Stock/crypto search functionality
   - AI recommendations generation
   - Trade execution (paper trading)
   - Watchlist management

3. **API Integrations:**
   - Financial data retrieval
   - AI analysis responses
   - Trading API connectivity
   - News feed updates

## Platform-Specific Considerations

### Bolt.New Limitations and Workarounds:

#### 1. File System Access:

- **Limitation:** Limited file system operations
- **Workaround:** Use external storage services (Supabase Storage, AWS S3)

#### 2. Background Jobs:

- **Limitation:** No persistent background processes
- **Workaround:** Use external cron services or serverless functions

#### 3. WebSocket Connections:

- **Limitation:** Limited WebSocket support
- **Workaround:** Use Server-Sent Events or polling for real-time updates

### 4. Memory Limits:

- **Limitation:** Serverless function memory constraints
- **Workaround:** Optimize data processing, use external services for heavy computations

## Performance Optimizations:

### 1. Database Queries:

```
// Optimize Prisma queries
const optimizedQuery = await prisma.user.findMany({
  select: {
    id: true,
    name: true,
    email: true,
    // Only select needed fields
  },
  where: {
    // Add proper indexing
  }
});
```

### 2. API Route Optimization:

```
// Implement caching
export async function GET(request: Request) {
  const cached = await redis.get(cacheKey);
  if (cached) return Response.json(cached);

  // Fetch fresh data
  const data = await fetchData();
  await redis.setex(cacheKey, 300, data); // 5-minute cache

  return Response.json(data);
}
```

### 3. Client-Side Optimization:

```
// Use React Query for data fetching
import { useQuery } from '@tanstack/react-query';

const { data, isLoading } = useQuery({
  queryKey: ['portfolio'],
  queryFn: fetchPortfolio,
  staleTime: 5 * 60 * 1000, // 5 minutes
});
```

# Troubleshooting Guide

## Common Issues and Solutions:

### 1. Build Failures:

```
# Issue: Prisma client not generated
# Solution:
npm run postinstall

# Issue: TypeScript errors
# Solution:
npm run lint
npx tsc --noEmit
```

### 2. Database Connection Issues:

```
# Test database connection
npx prisma db pull

# Reset database if needed
npx prisma migrate reset
```

### 3. Environment Variable Issues:

- Verify all required variables are set
- Check for typos in variable names
- Ensure proper escaping of special characters

### 4. API Integration Failures:

- Verify API keys are valid and active
- Check rate limits and quotas
- Implement proper error handling

## Monitoring and Logging:

### 1. Application Monitoring:

```
// Add error tracking
import { captureException } from '@sentry/nextjs';

try {
  // Your code
} catch (error) {
  captureException(error);
  console.error('Application error:', error);
}
```

### 2. Performance Monitoring:

```
// Add performance tracking
console.time('database-query');
const result = await prisma.user.findMany();
console.timeEnd('database-query');
```

## Security Considerations

### 1. Environment Variables:

- Never expose sensitive keys in client-side code
- Use NEXT_PUBLIC_ prefix only for non-sensitive variables
- Regularly rotate API keys

### 2. Database Security:

- Use connection pooling
- Implement proper access controls
- Regular security updates

### 3. API Security:

- Implement rate limiting
- Use proper authentication middleware
- Validate all inputs

## Rollback Plan

### If Migration Fails:

1. **Immediate Rollback:**
   - Keep original application running
   - Document all issues encountered
   - Revert DNS changes if applicable

2. **Data Recovery:**
   - Backup database before migration
   - Export user data if needed
   - Maintain data consistency

3. **Communication Plan:**
   - Notify users of any downtime
   - Provide status updates
   - Document lessons learned

## Maintenance and Updates

### Regular Maintenance Tasks:

1. **Weekly:**
   - Monitor application performance
   - Check error logs
   - Verify API integrations

2. **Monthly:**
   - Update dependencies
   - Review security patches
   - Optimize database queries

3. **Quarterly:**
   - Performance audit
   - Security review
   - Feature updates

## Support and Resources

### Bolt.New Resources:

- Official Documentation: https://support.bolt.new
- Community Forum: https://github.com/stackblitz/bolt.new/discussions
- GitHub Repository: https://github.com/stackblitz/bolt.new

### AiiA-Specific Support:

- Database Schema: `/prisma/schema.prisma`
- API Documentation: Review `/app/api/` routes
- Component Library: `/components/` directory

## Conclusion

This migration guide provides a comprehensive roadmap for successfully moving the AiiA application to the Bolt.New platform. The process involves careful preparation of files, proper environment configuration, database migration, and thorough testing.

Key success factors:
- Proper environment variable management
- Database migration strategy
- Thorough testing of all features
- Performance optimization
- Security considerations

Follow this guide step-by-step, and don't hesitate to reach out to the Bolt.New community for platform-specific questions or the development team for AiiA-specific issues.

---

**Last Updated:** July 23, 2025
**Version:** 1.0
**Author:** AI Migration Assistant