Epic 1.3: Development Environment Setup

Epic Overview

This epic creates a seamless local development environment for THE WHEEL design system, enabling efficient development with hot module replacement, optimized build processes, and comprehensive developer tooling.

Priority: P0 (Critical)
Timeline: 1.5 weeks
Dependencies: Epic 1.1 (Monorepo Architecture), Epic 1.2 (Storybook Foundation)

Story 1.3.1: Local Development Configuration

Overview

Set up a comprehensive local development environment with Docker support, environment management, and development tooling for efficient design system development.

AI Developer Prompt

You are setting up the local development environment for THE WHEEL design system. Building on the monorepo structure from Epic 1.1, you need to create a seamless development experience for contributors.

Context

- Monorepo structure with 6 packages (ui, patterns, layouts, themes, workspace, shared)
- Existing 85% complete component library with sophisticated theming
- Real-time collaboration features with WebSocket integration
- TypeScript strict mode with comprehensive type definitions
- Multiple workspace contexts (consultant, client, admin, expert, tool creator, founder)

Requirements

1. Create development environment configuration:

- Docker development environment for consistency
- Node.js and npm/yarn workspace configuration
- Environment variable management for different contexts

- Database setup for theme storage and user contexts
- Real-time service configuration (WebSocket, Firestore)

2. Set up development tools and utilities:

- ESLint and Prettier configuration for code quality
- Husky pre-commit hooks for validation
- VS Code workspace configuration and extensions
- Development scripts for common tasks
- Environment health checks and diagnostics

3. Configure workspace context development:

- Mock workspace data for all 6 contexts
- Permission system configuration
- Theme system development setup
- Real-time collaboration testing environment

Specific Tasks

yaml

```yaml
# docker-compose.yml
version: '3.8'
services:
  design-system:
    build:
      context: .
      dockerfile: Dockerfile.dev
    volumes:
      - .:/app
      - /app/node_modules
    ports:
      - "3000:3000"    # Storybook
      - "3001:3001"    # Development server
      - "3002:3002"    # WebSocket server
    environment:
      - NODE_ENV=development
      - WORKSPACE_CONTEXTS=consultant,client,admin,expert,toolCreator,founder
    command: npm run dev

  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: wheel_design_system
      POSTGRES_USER: wheel_dev
      POSTGRES_PASSWORD: local_dev_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

javascript

```
// .env.development template
# Application
NODE_ENV=development
PORT=3001

# Database
DATABASE_URL=postgresql://wheel_dev:local_dev_password@localhost:5432/wheel_design_system

# Redis
REDIS_URL=redis://localhost:6379

# WebSocket
WEBSOCKET_PORT=3002
WEBSOCKET_URL=ws://localhost:3002

# Workspace Contexts
ENABLE_CONSULTANT=true
ENABLE_CLIENT=true
ENABLE_ADMIN=true
ENABLE_EXPERT=true
ENABLE_TOOL_CREATOR=true
ENABLE_FOUNDER=true

# Development Features
HOT_RELOAD=true
SOURCE_MAPS=true
MOCK_DATA=true
DEBUG_MODE=true
```

Documentation Required

- Development environment setup guide
- Troubleshooting guide for common issues
- Environment configuration reference
- Development workflow documentation
- Contribution guidelines for new developers

Testing Requirements

- Environment validation tests

- Development service health checks
- Workspace context validation tests
- Real-time service connection tests
- Build system validation tests

Integration Points

- Integration with existing Storybook development
- Support for existing theme system
- Compatibility with real-time collaboration features
- Database and service integration
- CI/CD pipeline compatibility

Deliverables

- Complete development environment setup
- Docker configuration for services
- Development scripts and utilities
- Environment validation system
- Comprehensive developer documentation

Performance Requirements

- Development server startup under 30 seconds
- Hot reload response under 2 seconds
- Theme switching in development under 1 second
- Real-time service connection under 5 seconds
- Memory usage under 2GB for full environment

Story 1.3.2: Hot Module Replacement Setup

Overview

Implement comprehensive Hot Module Replacement (HMR) for lightning-fast development iteration across the monorepo with state preservation.

AI Developer Prompt

You are implementing Hot Module Replacement (HMR) for THE WHEEL design system development. Building on the local development configuration from Story 1.3.1, you need to create lightning-fast development iteration.

Context

- Monorepo with 6 packages requiring cross-package HMR
- Existing sophisticated theming system with CSS variables
- Real-time collaboration features that need state preservation
- Workspace context switching that must persist through HMR
- TypeScript components with complex prop interfaces

Requirements

1. Configure HMR for all package types:

- React components with state preservation
- CSS variables and theme system updates
- TypeScript interface changes
- Storybook hot reload integration
- Cross-package dependency updates

2. Implement state preservation during HMR:

- Workspace context state preservation
- Form data and user input preservation
- Real-time connection state management
- Theme and styling state persistence
- Component prop state preservation

3. Configure HMR for workspace features:

- Permission system changes
- Context switching updates
- Real-time collaboration state
- Theme system hot updates
- Component library live updates

Specific Tasks

javascript

```
// vite.config.ts for HMR configuration
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import { resolve } from 'path';

export default defineConfig({
  plugins: [
    react({
      fastRefresh: true,
      // Preserve component state during HMR
      babel: {
        plugins: [
          ['@babel/plugin-transform-react-jsx', { runtime: 'automatic' }]
        ]
      }
    })
  ],
  server: {
    hmr: {
      overlay: true,
      port: 3003
    }
  },
  optimizeDeps: {
    include: ['@wheel/ui', '@wheel/themes', '@wheel/workspace']
  },
  resolve: {
    alias: {
      '@wheel/ui': resolve(__dirname, './packages/ui/src'),
      '@wheel/themes': resolve(__dirname, './packages/themes/src'),
      '@wheel/workspace': resolve(__dirname, './packages/workspace/src')
    }
  }
});

// HMR state preservation utilities
export const preserveState = {
  workspace: {
    save: (state) => sessionStorage.setItem('hmr-workspace', JSON.stringify(state)),
    restore: () => JSON.parse(sessionStorage.getItem('hmr-workspace') || '{}')
  },
  theme: {
    save: (theme) => sessionStorage.setItem('hmr-theme', theme),
    restore: () => sessionStorage.getItem('hmr-theme') || 'default'
  },
  forms: {
    save: (formData) => sessionStorage.setItem('hmr-forms', JSON.stringify(formData)),
    restore: () => JSON.parse(sessionStorage.getItem('hmr-forms') || '{}')
  }
};
```

Documentation Required

- HMR configuration guide
- State preservation implementation
- Troubleshooting HMR issues
- Performance optimization tips
- Development workflow with HMR

Testing Requirements

- HMR functionality tests across packages
- State preservation validation tests
- Theme system hot update tests
- Real-time feature HMR tests
- Performance impact tests

Integration Points

- Integration with existing theme system
- Compatibility with real-time features
- Support for workspace context system
- Storybook integration
- Build system compatibility

Deliverables

- Complete HMR configuration
- State preservation system
- Development server with HMR
- Performance monitoring for HMR

- Developer documentation and guides

Performance Requirements

- HMR update response under 200ms
- State preservation accuracy 99%+
- CSS variable updates under 100ms
- Component updates under 500ms
- Memory usage increase under 50MB during HMR

Story 1.3.3: Development Server Configuration

Overview

Configure a robust development server that handles complex workspace features, real-time collaboration, and provides excellent developer experience.

AI Developer Prompt

You are configuring the development server for THE WHEEL design system. Building on the HMR setup from Story 1.3.2, you need to create a robust development server that handles the complex workspace features.

Context

- Monorepo with cross-package dependencies
- Real-time collaboration requiring WebSocket support
- Multiple workspace contexts requiring different server configurations
- Theme system with dynamic CSS variable generation
- Sophisticated routing for different workspace types

Requirements

1. Configure development server architecture:

- Express.js server with WebSocket support
- Proxy configuration for external services
- Static file serving for assets and themes
- API endpoint mocking for development
- CORS configuration for cross-origin development

2. Implement workspace context support:

- Multi-tenant development simulation
- Permission system endpoint mocking
- User context switching simulation
- Theme system endpoint support
- Real-time collaboration service mocking

3. Configure development middleware:

- Authentication simulation middleware
- Workspace context injection
- Real-time event simulation
- Error handling and logging
- Performance monitoring

Specific Tasks

javascript

```javascript
// dev-server.js
const express = require('express');
const { createServer } = require('http');
const { Server } = require('socket.io');
const cors = require('cors');
const morgan = require('morgan');

const app = express();
const httpServer = createServer(app);
const io = new Server(httpServer, {
  cors: {
    origin: ['http://localhost:3000', 'http://localhost:3001'],
    credentials: true
  }
});

// Middleware
app.use(cors());
app.use(morgan('dev'));
app.use(express.json());

// Workspace context middleware
app.use((req, res, next) => {
  const workspaceContext = req.headers['x-workspace-context'] || 'consultant';
  const userRole = req.headers['x-user-role'] || 'admin';

  req.workspace = {
    context: workspaceContext,
    user: {
      role: userRole,
      permissions: getPermissionsForRole(userRole)
    },
    theme: getThemeForContext(workspaceContext)
  };

  next();
});

// API Routes
app.get('/api/workspace/contexts', (req, res) => {
  res.json({
    contexts: ['consultant', 'client', 'admin', 'expert', 'toolCreator', 'founder']
  });
```

```
});

app.get('/api/theme/:context', (req, res) => {
  const theme = getThemeForContext(req.params.context);
  res.json(theme);
});

// WebSocket for real-time features
io.on('connection', (socket) => {
  console.log('Client connected:', socket.id);

  socket.on('join-workspace', (workspaceId) => {
    socket.join(workspaceId);
    socket.emit('workspace-joined', { workspaceId });
  });

  socket.on('theme-change', (data) => {
    socket.to(data.workspaceId).emit('theme-updated', data.theme);
  });

  socket.on('collaboration-event', (data) => {
    socket.to(data.workspaceId).emit('collaboration-update', data);
  });
});

// Static file serving
app.use('/assets', express.static('packages/themes/assets'));
app.use('/fonts', express.static('packages/themes/fonts'));

// Error handling
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    error: 'Internal Server Error',
    message: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
});

// Start server
const PORT = process.env.PORT || 3001;
httpServer.listen(PORT, () => {
  console.log(`Development server running on http://localhost:${PORT}`);
  console.log(`WebSocket server ready for connections`);
});
```

Documentation Required

- Development server architecture
- API endpoint documentation
- WebSocket event documentation
- Middleware configuration guide
- Troubleshooting server issues

Testing Requirements

- Server startup and health tests
- WebSocket connection tests
- API endpoint validation tests
- Workspace context tests
- Performance load tests

Integration Points

- Integration with existing real-time system
- Support for workspace context providers
- Compatibility with theme system
- Storybook development integration
- Build system integration

Deliverables

- Complete development server setup
- WebSocket and API mocking system
- Middleware for workspace features
- Performance monitoring tools
- Server configuration documentation

Performance Requirements

- Server startup under 10 seconds

- WebSocket connection establishment under 1 second
- API response time under 100ms
- Static file serving under 50ms
- Memory usage under 512MB

Developer Scripts and Commands

Package Scripts

json

```json
{
  "scripts": {
    "dev": "concurrently \"npm run dev:server\" \"npm run dev:storybook\"",
    "dev:server": "nodemon dev-server.js",
    "dev:storybook": "storybook dev -p 3000",
    "dev:packages": "nx run-many --target=dev --all",
    "build": "nx run-many --target=build --all",
    "test": "nx run-many --target=test --all",
    "lint": "nx run-many --target=lint --all",
    "format": "prettier --write \"packages/**/*.{ts,tsx,js,jsx,json,css,md}\"",
    "validate": "npm run lint && npm run test && npm run build",
    "clean": "nx run-many --target=clean --all && rm -rf dist",
    "reset": "npm run clean && rm -rf node_modules && npm install"
  }
}
```

Timeline and Dependencies

Timeline

- **Days 1-4**: Story 1.3.1 - Local Development Configuration
- **Days 5-7**: Story 1.3.2 - Hot Module Replacement Setup
- **Days 8-10**: Story 1.3.3 - Development Server Configuration

Dependencies

- Requires Epic 1.1 and 1.2 completion
- Docker and Node.js prerequisites
- Development tool installations

Success Metrics

- Development environment setup time under 10 minutes
- HMR working across all packages
- Server handling all workspace contexts
- Zero development environment blockers
- High developer satisfaction scores

Risk Mitigation

- Provide multiple environment options (Docker, local)
- Create comprehensive troubleshooting guides
- Test on multiple operating systems
- Have fallback configurations ready
- Regular developer feedback sessions