SafePlay™ Developer Setup Guide

Prerequisites

Before setting up SafePlay for development, ensure you have the following installed and configured:

System Requirements

- **Node.js**: Version 18.0 or higher
- **npm** or **yarn**: Latest stable version
- **Git**: For version control
- **PostgreSQL**: Version 13 or higher (local or cloud)
- **Code Editor**: VS Code recommended with TypeScript support

Required Accounts and Services

- **AWS Account**: For Rekognition and S3 services
- **Stripe Account**: For payment processing
- **Geoapify Account**: For address autocomplete services
- **Neon/Supabase Account**: For cloud PostgreSQL (production)

Local Development Setup

1. Repository Setup

```
# Clone the repository
git clone <repository-url>
cd safeplay-staging

# Install dependencies
yarn install

# Verify installation
yarn --version
node --version
```

2. Environment Configuration

Create a `.env.local` file in the root directory:

```
# Core Application Configuration
NODE_ENV=development
NEXT_PUBLIC_APP_URL=http://localhost:3000
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=your-super-secret-nextauth-key-min-32-chars

# Database Configuration
DATABASE_URL="postgresql://username:password@localhost:5432/safeplay_dev"

# Authentication Configuration
NEXTAUTH_JWT_SECRET=your-jwt-secret-key
NEXTAUTH_ENCRYPTION_KEY=your-encryption-key

# AWS Services
AWS_REGION=us-west-2
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=your-aws-secret-key
AWS_S3_BUCKET_NAME=safeplay-uploads-dev

# Stripe Payment Processing
STRIPE_SECRET_KEY=sk_test_...
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...

# Geoapify Address Services
GEOAPIFY_API_KEY=your-geoapify-api-key

# Email Configuration (Optional)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASSWORD=your-app-password

# Development Features
NEXT_PUBLIC_ENABLE_DEBUG=true
NEXT_PUBLIC_SHOW_VERSION_INFO=true
```

3. Database Setup

Local PostgreSQL Setup

```
# Install PostgreSQL (macOS)
brew install postgresql
brew services start postgresql

# Create development database
createdb safeplay_dev

# Create test database
createdb safeplay_test
```

Database Migration and Seeding

```
# Generate Prisma client
npx prisma generate

# Apply database migrations
npx prisma db push

# Seed the database with demo data
npx prisma db seed
```

Verify Database Setup

```
# Open Prisma Studio to view data
npx prisma studio
```

4. Development Server

```
# Start development server
yarn dev

# Alternative with debug logging
DEBUG=safeplay:* yarn dev

# Start on different port
PORT=3001 yarn dev
```

The application will be available at:
- **Main App**: http://localhost:3000
- **Admin Dashboard**: http://localhost:3000/admin
- **Venue Admin**: http://localhost:3000/venue-admin
- **Parent Dashboard**: http://localhost:3000/parent
- **Documentation**: http://localhost:3000/docs

Development Workflow

File Structure Overview

```
safeplay-staging/
├── app/                      # Next.js 14 App Router
│   ├── api/                  # API routes
│   ├── admin/                # Admin dashboard pages
│   ├── venue-admin/          # Venue admin pages
│   ├── parent/               # Parent dashboard pages
│   ├── docs/                 # User documentation
│   └── globals.css           # Global styles
├── components/               # Reusable React components
│   ├── ui/                   # Base UI components
│   ├── layouts/              # Layout components
│   └── [feature-folders]/    # Feature-specific components
├── lib/                      # Utility functions and services
│   ├── services/             # External service integrations
│   ├── auth.ts               # Authentication configuration
│   └── db.ts                 # Database connection
├── prisma/                   # Database schema and migrations
│   └── schema.prisma         # Database schema
├── public/                   # Static assets
└── types/                    # TypeScript type definitions
```

Code Standards and Conventions

TypeScript Configuration

- **Strict Mode**: Enabled for type safety
- **No Implicit Any**: All types must be explicit
- **Path Mapping**: Use `@/` for absolute imports

Component Guidelines

```tsx
// Use functional components with TypeScript
interface ComponentProps {
  title: string;
  children?: React.ReactNode;
  onAction?: () => void;
}

export function MyComponent({ title, children, onAction }: ComponentProps) {
  const [state, setState] = useState<string>('');

  // Always handle errors gracefully
  const handleAction = useCallback(() => {
    try {
      onAction?.();
    } catch (error) {
      console.error('Action failed:', error);
    }
  }, [onAction]);

  return (
    <div className="component-container">
      <h2>{title}</h2>
      {children}
      <button onClick={handleAction}>Action</button>
    </div>
  );
}
```

API Route Pattern

```ts
// app/api/example/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';

export async function GET(request: NextRequest) {
  try {
    // Check authentication
    const session = await getServerSession(authOptions);
    if (!session) {
      return NextResponse.json(
        { error: 'Authentication required' },
        { status: 401 }
      );
    }

    // Process request
    const data = await processRequest();

    return NextResponse.json(data);
  } catch (error) {
    console.error('API Error:', error);
    return NextResponse.json(
      { error: 'Internal server error' },
      { status: 500 }
    );
  }
}
```

Database Development

Schema Changes

```
# Make changes to prisma/schema.prisma
# Then apply changes:
npx prisma db push

# For production migrations:
npx prisma migrate dev --name descriptive-migration-name
```

Database Queries

```
// Use Prisma client with proper error handling
import { prisma } from '@/lib/db';

export async function getChildById(id: string) {
  try {
    const child = await prisma.child.findUnique({
      where: { id },
      include: {
        parent: true,
        emergencyContacts: true,
        currentLocation: true
      }
    });

    return child;
  } catch (error) {
    console.error('Database query failed:', error);
    throw new Error('Failed to fetch child data');
  }
}
```

Testing

Test Setup

```
# Install test dependencies (if not already installed)
yarn add -D jest @testing-library/react @testing-library/jest-dom

# Run unit tests
yarn test

# Run tests in watch mode
yarn test:watch

# Run tests with coverage
yarn test:coverage
```

Test Examples

```
// __tests__/components/MyComponent.test.tsx
import { render, screen } from '@testing-library/react';
import { MyComponent } from '@/components/MyComponent';

describe('MyComponent', () => {
  it('renders title correctly', () => {
    render(<MyComponent title="Test Title" />);

    expect(screen.getByText('Test Title')).toBeInTheDocument();
  });

  it('handles action click', () => {
    const mockAction = jest.fn();
    render(<MyComponent title="Test" onAction={mockAction} />);

    const button = screen.getByText('Action');
    button.click();

    expect(mockAction).toHaveBeenCalled();
  });
});
```

Integration Testing

```
# Test API endpoints
yarn test:api

# Test database operations
yarn test:db

# End-to-end testing with Playwright
yarn test:e2e
```

Debugging

Debug Configuration

```
// .vscode/launch.json
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Next.js: debug server-side",
      "type": "node",
      "request": "attach",
      "port": 9229,
      "skipFiles": ["<node_internals>/**"]
    },
    {
      "name": "Next.js: debug client-side",
      "type": "chrome",
      "request": "launch",
      "url": "http://localhost:3000"
    }
  ]
}
```

Logging and Monitoring

```
// Use structured logging
import { logger } from '@/lib/logger';

logger.info('User action', {
  userId: session.user.id,
  action: 'child_checkin',
  metadata: { childId, venueId }
});

logger.error('Database error', {
  error: error.message,
  query: 'getChildLocation',
  userId: session.user.id
});
```

Performance Optimization

Build Optimization

```
# Analyze bundle size
yarn build
yarn analyze

# Check for type errors
yarn type-check

# Lint code
yarn lint
yarn lint:fix
```

Performance Monitoring

```
// Add performance monitoring
import { performance } from 'perf_hooks';

export function withPerformanceMonitoring<T>(
  fn: () => Promise<T>,
  operationName: string
): Promise<T> {
  const start = performance.now();

  return fn().finally(() => {
    const duration = performance.now() - start;
    console.log(`${operationName} took ${duration.toFixed(2)}ms`);
  });
}
```

Deployment Preparation

Pre-deployment Checklist

- [ ] All environment variables configured
- [ ] Database migrations applied
- [ ] Tests passing
- [ ] No TypeScript errors
- [ ] Security audit passed
- [ ] Performance optimized

Build Commands

```
# Production build
yarn build

# Start production server locally
yarn start

# Docker build (if using Docker)
docker build -t safeplay .
docker run -p 3000:3000 safeplay
```

Troubleshooting

Common Issues

Database Connection Issues

```
# Check database status
npx prisma db pull

# Reset database (development only)
npx prisma migrate reset
```

Build Errors

```
# Clear Next.js cache
rm -rf .next

# Clear node modules and reinstall
rm -rf node_modules yarn.lock
yarn install
```

TypeScript Errors

```
# Restart TypeScript server in VS Code
# Cmd/Ctrl + Shift + P -> "TypeScript: Restart TS Server"

# Check types manually
yarn type-check
```

Environment-Specific Issues

Development

- Ensure all environment variables are set in `.env.local`
- Check database connectivity
- Verify service API keys are valid for development

Production

- Use environment-specific configuration
- Ensure all secrets are properly configured
- Monitor application logs for errors

Additional Resources

Documentation Links

- Next.js Documentation (https://nextjs.org/docs)
- Prisma Documentation (https://www.prisma.io/docs)
- NextAuth.js Documentation (https://next-auth.js.org)
- Tailwind CSS Documentation (https://tailwindcss.com/docs)

Internal Documentation

- User Manual (/docs) - Complete user documentation
- API Documentation (./API_DOCUMENTATION.md) - API endpoint specifications
- Deployment Guide (./DEPLOYMENT.md) - Production deployment instructions

Support

- **Development Team**: Internal support for setup issues
- **Documentation**: Complete guides at `/docs`
- **Code Examples**: Reference implementations in `/examples`

Last Updated: January 13, 2025
Version: 1.4.1
Maintainer: SafePlay Development Team