1DreamUnited Desktop Development Guide

Platform Overview

The 1DreamUnited desktop application is built using Electron with React and TypeScript, providing native desktop experiences across Windows, macOS, and Linux platforms.

Development Environment Setup

Prerequisites

- **Node.js** 18.0.0 or higher
- **npm** or **yarn** package manager
- **Git** for version control

Platform-Specific Requirements

Windows Development

- **Visual Studio Build Tools** or **Visual Studio Community**
- **Windows SDK** (latest version)
- **Python** 3.x (for native modules)

macOS Development

- **Xcode Command Line Tools**
- **macOS SDK** (included with Xcode)

Linux Development

- **Build essentials**: `sudo apt-get install build-essential`
- **Development libraries**: `sudo apt-get install libnss3-dev libatk-bridge2.0-dev libdrm2 libxcomposite1 libxdamage1 libxrandr2 libgbm1 libxss1 libasound2-dev`

Getting Started

1. Project Setup

```bash
# Navigate to desktop project
cd desktop/electron/

# Install dependencies
npm install

# Start development server
npm run dev
# or
npm start
```

2. Development Mode

The application will start in development mode with:
- Hot reload for renderer process

- DevTools enabled
- Source maps for debugging

Build Process

Development Build

```
# Package the application (no installer)
npm run package

# Create platform-specific distributables
npm run make
```

Production Builds

```
# Build for current platform
npm run dist

# Build for specific platforms
npm run dist:win     # Windows
npm run dist:mac     # macOS
npm run dist:linux   # Linux
```

Cross-Platform Building

```
# Build for all platforms (requires platform-specific tools)
npm run build:all
```

Project Structure

```
desktop/electron/
├── src/
│   ├── main.tsx              # React app entry point
│   ├── App.tsx               # Main React component
│   ├── index.css             # Global styles
│   ├── components/           # React components
│   │   ├── NavBar.tsx
│   │   ├── Footer.tsx
│   │   └── ...
│   ├── sections/             # Page sections
│   ├── contexts/             # React contexts
│   └── pages/                # Page components
├── main.ts                  # Electron main process
├── preload.ts              # Preload script
├── forge.config.ts         # Electron Forge configuration
├── vite.config.ts          # Vite configuration
├── package.json            # Dependencies and scripts
└── assets/                 # Application assets
    ├── icon.png
    ├── icon.ico
    └── icon.icns
```

⚡ Electron Architecture

Main Process (main.ts)

```
import { app, BrowserWindow } from 'electron';
import path from 'path';

const createWindow = (): void => {
  const mainWindow = new BrowserWindow({
    height: 800,
    width: 1200,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: false,
      contextIsolation: true
    }
  });

  // Load the app
  if (MAIN_WINDOW_VITE_DEV_SERVER_URL) {
    mainWindow.loadURL(MAIN_WINDOW_VITE_DEV_SERVER_URL);
  } else {
    mainWindow.loadFile(path.join(__dirname, `../renderer/${MAIN_WINDOW_VITE_NAME}/in-
dex.html`));
  }
};
```

Preload Script (preload.ts)

```
import { contextBridge, ipcRenderer } from 'electron';

// Expose protected methods that allow the renderer process to use
// the ipcRenderer without exposing the entire object
contextBridge.exposeInMainWorld('electronAPI', {
  openFile: () => ipcRenderer.invoke('dialog:openFile'),
  saveFile: (data: string) => ipcRenderer.invoke('dialog:saveFile', data),
  // Add more secure APIs as needed
});
```

Renderer Process (React App)

```
// Using the exposed API from preload script
declare global {
  interface Window {
    electronAPI: {
      openFile: () => Promise<string>;
      saveFile: (data: string) => Promise<void>;
    };
  }
}

const Component = () => {
  const handleOpenFile = async () => {
    const filePath = await window.electronAPI.openFile();
    console.log('Selected file:', filePath);
  };

  return (
    <button onClick={handleOpenFile}>
      Open File
    </button>
  );
};
```

Key Features Implementation

AI Ecosystem Integration

```
// Desktop-specific AI features
import { ipcRenderer } from 'electron';

const AIDesktopService = {
  async processAudioFile(filePath: string) {
    return await ipcRenderer.invoke('ai:processAudio', filePath);
  },

  async generateRecommendations(userData: any) {
    return await ipcRenderer.invoke('ai:generateRecommendations', userData);
  }
};
```

Native File System Access

```
// File operations with native dialogs
const FileManager = {
  async openMusicFile() {
    const { dialog } = require('@electron/remote');
    const result = await dialog.showOpenDialog({
      properties: ['openFile'],
      filters: [
        { name: 'Audio Files', extensions: ['mp3', 'wav', 'flac', 'm4a'] },
        { name: 'All Files', extensions: ['*'] }
      ]
    });

    return result.filePaths[0];
  },

  async saveProject(projectData: any) {
    const { dialog } = require('@electron/remote');
    const result = await dialog.showSaveDialog({
      filters: [
        { name: 'Project Files', extensions: ['1du'] },
        { name: 'JSON Files', extensions: ['json'] }
      ]
    });

    if (!result.canceled) {
      // Save project data to file
      await fs.writeFile(result.filePath, JSON.stringify(projectData));
    }
  }
};
```

System Integration

```
// Desktop notifications
const NotificationService = {
  show(title: string, body: string) {
    new Notification(title, {
      body,
      icon: path.join(__dirname, 'assets/icon.png')
    });
  }
};

// System tray integration
const TrayService = {
  create() {
    const tray = new Tray(path.join(__dirname, 'assets/tray-icon.png'));
    tray.setContextMenu(Menu.buildFromTemplate([
      { label: 'Show App', click: () => mainWindow.show() },
      { label: 'Quit', click: () => app.quit() }
    ]));
  }
};
```

Testing

Unit Testing

```
# Run unit tests
npm test

# Run tests with coverage
npm run test:coverage

# Run tests in watch mode
npm run test:watch
```

E2E Testing with Spectron

```
# Install Spectron for E2E testing
npm install --save-dev spectron

# Run E2E tests
npm run test:e2e
```

Testing Configuration

```
// jest.config.js
module.exports = {
  testEnvironment: 'jsdom',
  setupFilesAfterEnv: ['<rootDir>/src/setupTests.ts'],
  moduleNameMapping: {
    '\\.(css|less|scss|sass)$': 'identity-obj-proxy'
  },
  transform: {
    '^.+\\.(ts|tsx)$': 'ts-jest'
  }
};
```

Configuration

Electron Forge Configuration

```typescript
// forge.config.ts
import type { ForgeConfig } from '@electron-forge/shared-types';
import { MakerSquirrel } from '@electron-forge/maker-squirrel';
import { MakerZIP } from '@electron-forge/maker-zip';
import { MakerDeb } from '@electron-forge/maker-deb';
import { MakerRpm } from '@electron-forge/maker-rpm';
import { MakerDMG } from '@electron-forge/maker-dmg';

const config: ForgeConfig = {
  packagerConfig: {
    icon: './assets/icon',
    executableName: '1dreamunited'
  },
  rebuildConfig: {},
  makers: [
    new MakerSquirrel({}),
    new MakerZIP({}, ['darwin']),
    new MakerRpm({}),
    new MakerDeb({}),
    new MakerDMG({
      background: './assets/dmg-background.png',
      format: 'ULFO'
    })
  ]
};

export default config;
```

Vite Configuration

```typescript
// vite.config.ts
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  base: './',
  build: {
    outDir: 'dist',
    assetsDir: 'assets',
    rollupOptions: {
      external: ['electron']
    }
  },
  server: {
    port: 3000
  }
});
```

Deployment

Windows Deployment

```
# Create Windows installer
npm run dist:win

# Output: dist/1DreamUnited Setup 1.0.0.exe
```

macOS Deployment

```
# Create macOS DMG
npm run dist:mac

# Output: dist/1DreamUnited-1.0.0.dmg
# Code signing required for distribution
```

Linux Deployment

```
# Create Linux packages
npm run dist:linux

# Output:
# - dist/1DreamUnited-1.0.0.AppImage
# - dist/1dreamunited_1.0.0_amd64.deb
# - dist/1dreamunited-1.0.0.x86_64.rpm
```

Code Signing

macOS Code Signing

```
# Set environment variables
export APPLE_ID="your-apple-id@example.com"
export APPLE_ID_PASSWORD="app-specific-password"
export CSC_LINK="path/to/certificate.p12"
export CSC_KEY_PASSWORD="certificate-password"

# Build with code signing
npm run dist:mac
```

Windows Code Signing

```
# Set environment variables
export CSC_LINK="path/to/certificate.p12"
export CSC_KEY_PASSWORD="certificate-password"

# Build with code signing
npm run dist:win
```

Troubleshooting

Common Issues

Native Dependencies

```
# Rebuild native modules for Electron
npm run electron-rebuild

# Or use electron-builder
npx electron-builder install-app-deps
```

Permission Issues (macOS)

```
# Fix quarantine issues
xattr -cr /Applications/1DreamUnited.app
```

Linux Dependencies

```
# Install missing libraries
sudo apt-get install libgconf-2-4 libxss1 libasound2-dev
```

Performance Optimization

- Use `webSecurity: false` only in development
- Implement proper memory management
- Use `nodeIntegration: false` for security
- Optimize bundle size with tree shaking

Security Best Practices

- Always use `contextIsolation: true`
- Never set `nodeIntegration: true` in production
- Validate all IPC communications
- Use CSP (Content Security Policy)

Analytics and Monitoring

Crash Reporting

```
import { crashReporter } from 'electron';

crashReporter.start({
  productName: '1DreamUnited',
  companyName: '1DreamUnited',
  submitURL: 'https://your-crash-server.com/submit',
  uploadToServer: true
});
```

Usage Analytics

```
// Track application usage
const AnalyticsService = {
  track(event: string, properties: any) {
    // Send to analytics service
    ipcRenderer.send('analytics:track', { event, properties });
  }
};
```

Auto-Updates

Electron Updater Setup

```
import { autoUpdater } from 'electron-updater';

// Configure auto-updater
autoUpdater.checkForUpdatesAndNotify();

autoUpdater.on('update-available', () => {
  // Notify user of available update
});

autoUpdater.on('update-downloaded', () => {
  // Prompt user to restart and install
  autoUpdater.quitAndInstall();
});
```

Additional Resources

- Electron Documentation (https://www.electronjs.org/docs)
- Electron Forge Documentation (https://www.electronforge.io/)
- Vite Documentation (https://vitejs.dev/)
- React Documentation (https://reactjs.org/docs)
- TypeScript Documentation (https://www.typescriptlang.org/docs)

For technical support or questions, refer to the main project documentation or contact the development team.