

Георгий Димитриади



**Практическое руководство по
созданию приложений на
основе фреймворка Electron**

Георгий Димитриади

Практическое руководство

по созданию приложений на

основе фреймворка Electron

<https://litres.ru/74611928>

SelfPub; 2026

Аннотация

Это пошаговое пособие для начинающих разработчиков, посвященное созданию кроссплатформенных настольных программ. Книга охватывает полный цикл разработки: от инициализации проекта и настройки среды до сборки дистрибутивов для Windows, macOS и Linux, а также реализации механизмов автоматического обновления.

В основе руководства лежит архитектурная парадигма Electron, объединяющая движок Chromium для рендеринга интерфейса и среду Node.js для доступа к нативным API операционной системы. Особое внимание уделяется вопросам безопасности, изоляции процессов и организации межпроцессного взаимодействия (IPC).

Издание рассчитано на читателей, уже знакомых с основами синтаксиса JavaScript, работой с командной строкой и менеджером пакетов npm. Знание HTML и CSS является

необходимым условием для понимания примеров верстки. Теоретические концепции подкреплены большим количеством практических примеров кода, а финалом обучения становится создание полноценного приложения «Менеджер заметок».

Содержание

Введение	5
Глава 1. Инициализация проекта и базовая конфигурация	11
Конец ознакомительного фрагмента.	16

Георгий Димитриади

Практическое

руководство по созданию

приложений на основе

фреймворка Electron

Введение

1.1. Определение кроссплатформенного программного обеспечения

В современной индустрии разработки программного обеспечения существует фундаментальная проблема фрагментации целевых платформ. Операционные системы, такие как Microsoft Windows, Apple macOS и различные дистрибутивы на базе ядра Linux, обладают уникальными наборами системных библиотек, графических подсистем и прикладных интерфейсов (API). Традиционный подход к созданию настольных приложений (desktop applications) требует написания и поддержки отдельных кодовых баз для каждой целевой платформы, что экспоненциально увеличивает трудозатраты, стоимость разработки и время вывода продукта на

рынок.

Решением данной проблемы является разработка кросс-платформенного программного обеспечения. Под кросс-платформенным программным обеспечением понимается совокупность программных модулей, исполняемый код которых способен функционировать на нескольких аппаратно-программных архитектурах без необходимости перекомпиляции исходного текста под каждую конкретную систему. Данный подход позволяет абстрагироваться от специфики операционной системы и сосредоточить ресурсы на реализации бизнес-логики приложения.

1.2. Фреймворк Electron: определение и архитектурная парадигма

Фреймворк Electron представляет собой программный каркас с открытым исходным кодом, первоначально разработанный компанией GitHub для редактора исходного кода Atom. Electron позволяет создавать графические пользовательские интерфейсы (Graphical User Interface, GUI) с использованием стандартных веб-технологий.

Для начинающего разработчика необходимо пояснить состав этих технологий:

HTML (HyperText Markup Language) - язык разметки гипертекста. Определяет семантическую структуру и содержимое пользовательского интерфейса (кнопки, текстовые поля, блоки).

CSS (Cascading Style Sheets) - каскадные таблицы сти-

лей. Описывают визуальное представление компонентов: цвета, шрифты, отступы и позиционирование элементов.

JavaScript - высокоуровневый язык программирования. Обеспечивает интерактивность интерфейса, обработку событий (например, нажатие мыши) и манипуляцию данными.

Использование данных технологий позволяет привлекать к разработке широкий круг специалистов в области веб-фронтенда, минимизируя порог входа в сферу нативной (родной для операционной системы) разработки.

1.3. Компоненты архитектуры: Chromium и Node.js

Архитектура Electron базируется на интеграции двух фундаментальных программных компонентов с открытым исходным кодом.

Chromium. Это мультиплатформный проект веб-браузера с открытым исходным кодом, инициированный компанией Google. В контексте Electron движок Chromium выполняет роль подсистемы рендеринга. Он отвечает за парсинг (чтение) HTML-разметки, применение CSS-стилей и отрисовку финального изображения окна приложения на экране монитора. По сути, каждое окно Electron-приложения является изолированным экземпляром браузера.

Node.js. Это среда выполнения JavaScript-кода, построенная на движке V8 (том же, что используется в Chromium), но функционирующая вне контекста браузера. Ключевым отличием Node.js является предоставление доступа к файловой системе, сетевым сокетам, процессам операционной систе-

мы и другим низкоуровневым ресурсам. В Electron Node.js используется для реализации серверной (backend) логики, взаимодействия с API операционной системы и управления жизненным циклом приложения.

Объединение этих компонентов позволяет исполнять один и тот же язык программирования (JavaScript) как для отрисовки интерфейса, так и для работы с системными ресурсами.

1.4. Процессный подход: основной (Main) и рендерный (Renderer) процессы

Для обеспечения стабильности и безопасности Electron реализует многопроцессную архитектуру, заимствованную из модели современных веб-браузеров. Исполнение кода приложения разделено на два фундаментальных типа процессов.

Основной процесс (Main Process). Это управляющий процесс приложения. В каждом запущенном приложении Electron существует ровно один основной процесс. Он исполняется в окружении Node.js и несет ответственность за создание окон, взаимодействие с системным треем (областью уведомлений), управление глобальными горячими клавишами и обработку жизненного цикла приложения (старт, завершение, сворачивание). Точкой входа в основной процесс является файл, путь к которому указан в поле `main` конфигурационного файла `package.json`.

Рендерный процесс (Renderer Process). Данный

процесс создается основным процессом для каждого отдельного окна (объекта `BrowserWindow`). Код рендерерного процесса выполняется в изолированном окружении, аналогичном вкладке браузера. Он имеет доступ к объектной модели документа (DOM) и отвечает исключительно за отображение пользовательского интерфейса и обработку действий пользователя внутри конкретного окна. По умолчанию рендерерный процесс не имеет прямого доступа к Node.js API для предотвращения выполнения вредоносного кода в контексте веб-страницы.

Взаимодействие между этими процессами осуществляется посредством асинхронного механизма межпроцессного взаимодействия (Inter-Process Communication, IPC), который будет детально рассмотрен в Главе 2.

1.5. Предварительные требования к рабочей станции

Для выполнения практических примеров, представленных в данном руководстве, рабочая станция должна быть сконфигурирована следующим образом:

Операционная система: Windows 10/11 (64-bit), macOS 12 или новее, либо современный дистрибутив Linux (например, Ubuntu 20.04 LTS и выше).

Node.js и npm. Node.js - это среда выполнения, npm (Node Package Manager) - входящий в ее состав менеджер пакетов. Рекомендуется установка актуальной версии LTS (Long-Term Support). Проверка корректности установки осу-

осуществляется выполнением в терминале следующих команд:

```
node -v
```

```
npm -v
```

Терминал должен вернуть номера версий установленного программного обеспечения.

Интегрированная среда разработки (IDE). Рекомендуется использование Visual Studio Code. Данная среда разработана на базе Electron, обладает встроенным терминалом, системой контроля версий Git и расширенной поддержкой синтаксиса JavaScript, что делает ее оптимальным инструментом для обучения.

Глава 1. Инициализация проекта и базовая конфигурация

Данная глава посвящена практическим шагам по созданию минимальной структуры приложения на базе фреймворка Electron. Рассматриваемые процедуры являются стандартом де-факто в экосистеме Node.js и служат фундаментом для дальнейшей разработки.

1.1. Создание директории проекта и инициализация через npm

Первоначальным этапом является создание изолированной директории для хранения исходных кодов и зависимостей проекта. В рамках данного руководства проект будет именоваться `electron-guide-app`.

В терминале операционной системы необходимо выполнить следующую последовательность команд:

```
mkdir electron-guide-app  
cd electron-guide-app
```

Пояснение: Утилита `mkdir` (make directory) создает новую папку, команда `cd` (change directory) осуществляет переход в созданную директорию, делая ее рабочей.

Следующим шагом является инициализация менеджера пакетов npm. Данная процедура генерирует файл манифеста `package.json`.

`npm init -y`

Пояснение термина: npm (Node Package Manager) - инструмент для управления зависимостями проекта. Файл `package.json` представляет собой документ в формате JSON (JavaScript Object Notation), который содержит метаданные приложения: имя, версию, автора, а также список необходимых библиотек. Флаг `-y` (yes) автоматически подтверждает установку значений по умолчанию, избавляя от необходимости интерактивного ввода данных.

Пример сгенерированного файла `package.json`:

```
{  
  "name": "electron-guide-app",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

1.2. Установка фреймворка Electron как зависимости для разработки

Для интеграции Electron в проект необходимо загрузить его из центрального репозитория npm и сохранить в качестве

зависимости.

```
npm install --save-dev electron
```

Пояснение: Команда `npm install` (допускается сокращение `npm i`) инициирует загрузку. Флаг `--save-dev` (допускается `-D`) указывает менеджеру пакетов, что `Electron` является *dev зависимостью* (dependency for development). Это означает, что фреймворк требуется для процесса создания и тестирования приложения, но не будет включен в минимальную сборку готового продукта, что оптимизирует итоговый размер дистрибутива. После выполнения команды в директории появится поддиректория `node_modules`, содержащая код `Electron`, а в `package.json` в блоке `devDependencies` будет добавлена соответствующая запись.

1.3. Создание точки входа: файл `main.js`

В соответствии с архитектурной парадигмой, описанной в разделе 1.4 Введения, необходимо создать файл, исполняющийся в основном (Main) процессе. Создадим файл с именем `main.js` в корне проекта.

Содержимое файла `main.js`:

```
// Импорт необходимых модулей из API Electron
const { app, BrowserWindow } = require('electron');
const path = require('node:path');

// Функция создания окна браузера
function createWindow () {
  // Создание экземпляра окна
  const win = new BrowserWindow({
```

```
width: 800,  
height: 600  
});  
// Загрузка локального HTML-файла в созданное окно  
win.loadFile('index.html');  
}  
// Жизненный цикл приложения  
// Метод app.whenReady() возвращает Promise, который  
разрешается,  
// когда Electron завершит инициализацию и инициализи-  
рует BrowserWindow  
app.whenReady().then(() => {  
  createWindow();  
  // На macOS приложение не завершается при закрытии  
всех окон  
  app.on('activate', () => {  
    if (BrowserWindow.getAllWindows().length === 0) {  
      createWindow();  
    }  
  });  
});  
// Выход из приложения, когда все окна закрыты (для  
Windows и Linux)  
app.on('window-all-closed', () => {  
  if (process.platform !== 'darwin') {  
    app.quit();
```

```
}  
});
```

Разбор кода:

`require('electron')` - подключение встроенного модуля.

Оператор деструктуризации `{ app, BrowserWindow }` позволяет извлечь конкретные классы из модуля.

`new BrowserWindow()` - инстанцирование окна. Объект конфигурации задает начальные размеры.

`win.loadFile()` - метод, указывающий окну отрендерить конкретный HTML-документ.

`app.whenReady()` - асинхронный метод, гарантирующий, что API Electron готово к созданию окон.

Обработчики событий `app.on('window-all-closed')` и `app.on('activate')` обеспечивают нативное поведение приложения на разных операционных системах.

1.4. Модификация файла `package.json`

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.