УДК 004.01
ББК 32.972

Арно Лоре
Проектирование веб-API / Пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 440 с.
ISBN 978-5-97060-861-6

Книга, написанная с учетом многолетнего опыта автора в разработке API, научит вас, как собирать требования, как найти баланс между техническими и бизнес-целями и как принимать во внимание запросы потребителя. Рассматриваются основные характеристики API, принципы его изменения, документирования и проверки.

Эффективные методы разработки проиллюстрированы множеством интересных примеров.

Издание предназначено для разработчиков, обладающих минимальным опытом в создании и использовании API-интерфейсов.

УДК 004.01

ISBN 978-5-97060-861-6 (рус.)
ISBN 978-1-61729-510-2 (анг.)

Оглавление

Содержание

Предисловие

На протяжении более десяти лет для большинства разработчиков проектирование API всегда подразумевало архитектуру REST. Такая ситуация сложилась в связи с регулярным выходом книг и блогов об API, которые продвигают мировоззрение RESTful – и это порождает некую зацикленность на данном направлении, если не сказать догматизм. Эта книга знаменует появление руководств по проектированию API следующего поколения, которые помогут нам преодолеть ограничения, тормозившие более десяти лет. Практический подход к разработке API по-прежнему основывается на REST, но автор очень старался сформировать представление о разработке API в практических ситуациях за вычетом имеющихся догм.

Арно Лоре знакомит нас с основами проектирования API, которые легко можно найти в других отраслевых изданиях, и раскрывает тему просто и доступно, рассматривая широкий круг вопросов в удобной для читателя форме. Я знаком с Арно уже несколько лет и считаю его одним из немногих высокопрофессиональных специалистов, которые не только знают, как создавать API технически, но и понимают проблемы, связанные с их доставкой потребителю, и знают, в каких случаях API могут оказывать положительное или отрицательное влияние на опыт разработчика. Арно фокусируется не только на проектировании API, но и на предоставлении правильно спроектированного API целевой аудитории продуманным образом.

Я лично наблюдал, как Арно принимал самое активное участие в дискуссиях, касающихся API, по всему миру, извлекая из них все самое полезное для себя. Достаточно просмотреть его страницу в Twitter или перейти по хештегу, посвященному какому-нибудь популярному мероприятию, связанному с API, чтобы понять, о чем я говорю. Арно использует уникальный подход, слушая доклады представителей из индустрии API и обрабатывая информацию, которой они делятся, и затем освеща-

ет в Twitter важные моменты беседы, представляя сведения об API в удобном для восприятия формате. Я рад, что Арно свел накопленные им знания воедино и изложил в книге, продолжая не только оттачивать собственные навыки, но и делиться с другими людьми своими знаниями и уникальным подходом к проектированию API. Арно принадлежит к редкой породе API-аналитиков, которые заботятся о развитии темы: впитывают знания об API, усваивают их и распространяют в доступной форме, так что их становится действительно легко применить в мире бизнеса.

После того как, начиная с 2012 года, проектирование API стало набирать обороты и стала очевидной доминирующая роль спецификации OpenAPI (ранее известной как Swagger), Арно был одним из немногих экспертов в области API, которые усердно старались раскрыть потенциал этой спецификации, а также разрабатывали инновационные инструменты и визуализации, связанные со стандартом спецификации API с открытым исходным кодом – чтобы понять не только спецификацию, но и то, как она может реализовывать, представлять и даже организовывать многие принципы проектирования API, необходимые для достижения успеха. Понадобится много работать, прежде чем вы поймете, что OpenAPI – не только документация; большинство разработчиков API так и не проходят этот путь до конца. Арно понимает, что OpenAPI – это к тому же и основа проектирования API для любой платформы, что помогает определить каждую остановку на протяжении всего жизненного цикла API. Эта книга – первое руководство по проектированию API из тех, что я видел, в котором OpenAPI и проектирование API объединены так вдумчиво и эффективно, что многие разработчики почувствуют несомненную пользу чтения.

Потратьте время, чтобы понять, чем Арно делится с вами. Это книга не предназначена для беглого пролистывания, и она уж точно не из серии «прочитал и забыл». Это справочник. Руководство по переходу проектирования ваших API на новый уровень. Оно дает вам набор концепций API, с которыми вы знакомы, и предоставляет вам чертежи для создания «Тысячелетнего сокола» или даже «Звезды Смерти» (если, конечно, пожелаете) из набора строительных блоков API.

Рекомендую вам прочитать эту книгу, а затем отложить ее на месяц. Затем приступайте к созданию API и переходу от проектирования к фактическому развертыванию и общедоступности – и поделитесь своими знаниями с разработчиками. А пока ждете отзывов, сядьте и снова возьмитесь за книгу. Вы начнете понимать глубину и ценность знаний, которыми Арно делится с вами. Возвращайтесь к чтению до тех пор, пока вы не научитесь в полной мере создавать API – не идеальные образцы,

а именно тот API, который вам нужен, чтобы достучаться до потребителей, на которых вы рассчитываете повлиять.

— Кин Лейн, апологет API

От автора

На протяжении своей карьеры я большей частью соединял программные блоки, используя различные технологии программных интерфейсов – начиная с простых файлов и баз данных и заканчивая удаленными программными интерфейсами на базе RPC, Corba, Java RMI, веб-сервисов SOAP и веб-API. В эти годы мне посчастливилось работать над разнородными распределенными системами, смешивая очень старую технологию мейнфреймов с современными облачными системами и всем, что находится посередине между ними. Мне также посчастливилось работать с обеими сторонами программных интерфейсов в различных контекстах. Я работал над IVR (Interactive Voice Response), веб-сайтами и мобильными приложениями, созданными поверх огромных систем сервис-ориентированной архитектуры. Я создавал закрытые и общедоступные веб-сервисы и веб-API для веб-приложений и приложений на стороне сервера. Все эти годы я много жаловался на ужасные программные интерфейсы – и попадал в ловушки, создавая точно такие же.

Шли годы, и технология развивалась, переходя от RPC к веб-сервисам SOAP и веб-API. Объединять программное обеспечение становилось все проще и проще с технической точки зрения. Но независимо от используемой технологии, я понял, что программный интерфейс – это гораздо больше, чем просто связующая система или побочный продукт программного проекта. После посещения первых конференций по API в 2014 году на API Days в Париже я понял, что есть множество других людей, которые, как и я, сражаются с API. Вот почему в 2015 году я начал вести блог API Handyman, а также стал участвовать в конференциях, посвященных API. Я хотел поделиться своим опытом с другими и помочь им избежать попадания в те же ловушки, в которые попал я. Когда я писал о веб-API и обсуждал их, это не только позволяло мне помогать другим, но и давало мне возможность узнать о них еще больше.

После двух лет ведения блогов и выступлений на конференциях я пришел к решению написать книгу. Я хотел адресовать ее прежнему себе – тому, кто испытывал столько затруднений. К счастью, издательство Manning Publications искало человека, готового написать книгу о спецификации OpenAPI, формате описания API (об этом мы поговорим в главе 4). Я воспользовался шансом, предложив свою книгу Design of Everyday APIs, и ее приняли. На это название меня вдохновила книга «Дизайн привычных вещей» Дона Нормана (MIT Press, 1998) – вам обязательно стоит прочитать ее. Позже первоначальный заголовок заменили более простым, The Design of Web APIs. Должен признать, что он мне нравится больше; теперь у меня нет ощущения, что я покушаюсь на лавры Дона Нормана.

Поначалу в книге освещалось проектирование повседневных вещей + API + REST в сравнении с gRPC в сравнении с GraphQL. Усвоить все это было бы довольно трудно, но я хотел, чтобы изложенные в книге принципы можно было использовать для любого типа API. Месяц за месяцем содержание совершенствовалось, в результате чего получилась книга, которая сейчас называется The Design of Web APIs («Проектирование веб-API»). Я решил сосредоточиться на REST API и использовать их в качестве примера для изучения принципов проектирования веб- и удаленных API, что выходит за рамки простого проектирования API. Если бы такая книга попалась в прежние времена, я был бы очень рад ее изучить; надеюсь, вам тоже понравится!

Благодарности

Два года. Мне потребовалось два года, чтобы написать книгу. Да, это немало, за-то в итоге получилась замечательная книга, которая, надеюсь, вам понравится. Я работал над ней не в одиночку. Мне многих нужно поблагодарить за непосредственную помощь в работе, но не меньше я благодарен и тем, кто эту работу сделал возможной. Прежде всего я благодарен своей жене Синзии и дочери Элизабетте. Большое вам спасибо за вашу поддержку и терпение, пока я проводил вечера и выходные за написанием книги. Я очень вас люблю.

Далее я хотел бы поблагодарить всех сотрудников издательства Manning Publications. Многие даже не представляют, сколько людей работает над книгой с самого начала ее создания! Каждый из этих людей проделал замечательную работу. Главным образом я бы хотел поблагодарить моего редактора Майка Стивенса, который поверил в этот проект. Особая благодарность двум редакторам – консультантам по аудитории, Кевину Харрелду и Дженнифер Стаут, и техническому редактору Майклу Ланду. Вы действительно очень помогли мне! Без вашего участия эта книга не стала бы такой, какой ее видит читатель. И отдельное merci beaucoup моему редактору текста из ESL Рэйчел Хэд, которая проделала поистине потрясающую работу, корректируя мой «французский английский». Спасибо производственному редактору Дейрдре Хайам, редактору Фрэнсис Буран, корректору Мелоди Долаб и техническому редактору Полу Гребенцу. Также я хотел бы поблагодарить своих рецензентов: Эндрю Гвоздзевич, Энди Кирша, Бриджера Хауэлла, Эдвина Квока, Энрико Маццареллу, Мохаммада Али Бацци, Нараянан Джаяратчаган, Питера Пола Селларса, Равиша Шарму, Санджива Кумара Джайсвала, Серхио Пачеко, Шона Хиксона, Шона Смита, Винсента Терона и Уильяма Руденмальма.

Особая благодарность Ивану Гончарову, переславшему мне 15 марта 2017 года электронное письмо от издательства Manning Publications. Оно искало человека, который написал бы книгу об API, в результате чего

и появилась на свет «Проектирование веб-API». Я рад, что по счастливой случайности мы встретились на REST Fest 2015.

Спасибо всем, кто нашел время, чтобы прочитать рукопись на разных этапах ее подготовки и предоставил неоценимую поддержку и отзывы. Отдельная благодарность Изабель Реуса и Мехди Меджауи за «полевое тестирование» содержания книги и за обратную связь. И спасибо всем тем, кто занимается проектированием API, – людям, с которыми я встречался и работал на протяжении многих лет. Я вложил в эту книгу все, чему научился у вас!

И еще один раз поблагодарю Майка Амундсена, Кина Лэйна и Мехди Меджауи, теперь уже за поддержку и помощь в то время, когда я начал вести блог API Handyman (в 2015 году). Без вас этой книги бы не было.

Об этой книге

Эта книга была написана, для того чтобы помочь вам проектировать веб-API, которые не просто удовлетворяют выраженные потребности. Книга поможет вам проектировать выдающиеся веб-API, которые могут использоваться любым человеком в самых разных контекстах и которые также являются безопасными, долговечными, расширяемыми, эффективными и реализуемыми. В ней раскрываются все аспекты проектирования веб-API и дается полный обзор экосистемы веб-API и того, как проектировщики API могут внести в нее свой вклад.

Кому адресована эта книга?

Данная книга, очевидно, предназначена для всех тех, кому необходимо проектировать веб-API. Это могут быть разработчики, работающие над серверной частью для мобильных приложений или веб-сайтов, или те, кому нужно соединять микросервисы воедино, или же это могут быть владельцы продуктов, работающие над API в качестве продукта и всем, что находится посередине. На самом деле эту книгу могут прочитать все те, кто работает над проектом, связанным с созданием API.

Как устроена эта книга: дорожная карта

Книга состоит из трех частей, которые охватывают 13 глав.

В первой части представлены основные понятия и навыки, необходимые для проектирования API.

В главе 1 обсуждается, что такое API, чем важно его проектирование, и что из себя представляют элементы проектирования API.

В главе 2 объясняется, как точно определить назначение API – его реальные цели, – сосредоточив внимание на точке зрения пользователей API и программного обеспечения, использующего API, и избегая точки зрения организации и программного обеспечения, предоставляющего API.

Глава 3 знакомит вас с протоколом HTTP, REST API и архитектурным стилем REST. Она учит, как проектировать интерфейс веб-программирования (содержащий ресурсы, действия над ресурсами, данные, параметры и ответы) на основе определенных целей.

Глава 4 знакомит вас со спецификацией OpenAPI и демонстрирует, как описывать API структурированным и стандартным способом, используя такой формат описания API.

Вторая часть посвящена тому, как проектировать API типа «не заставляйте меня думать», которые будут просты для понимания и использования.

В главе 5 объясняется, как проектировать простые представления данных, сообщения об ошибках и успешных результатах, а также потоки вызовов API, которые сразу же будут понятны, и их легко будет использовать.

В главе 6 рассказывается, как создавать еще более простые для понимания и удобные в использовании API-интерфейсы, пользователи которых (люди или машины) смогут угадать, как работают API, делая их согласованными, адаптируемыми и обнаруживаемыми.

В главе 7 показано, как организовать и оценить все аспекты API, чтобы их было легко понять и использовать.

В третьей части показано, что проектировщики API должны учитывать контекст, окружающий API, и весь контекст, окружающий сам процесс проектирования API.

В главе 8 описывается безопасность API и то, как проектировать безопасные API.

В главе 9 рассказывается, как изменить API, не оказывая чрезмерного влияния на его пользователей, и как и когда использовать управление версиями. В ней также демонстрируется, как проектировать API, которые будет легко расширять с нуля.

Глава 10 посвящена тому, как создавать веб-API для эффективной работы в сети.

Глава 11 раскрывает весь контекст, который должны учитывать проектировщики при проектировании API. Она включает в себя адаптацию механизмов обмена данными (запрос/ответы, асинхронность, события и пакетную обработку), оценку и адаптацию к ограничениям потребителей или поставщиков и выбор адекватного стиля API (на базе ресурсов, функций или данных).

В главе 12 объясняется, как проектировщики API участвуют в создании различных типов документации API, используя преимущества такого формата описания API как спецификацию OpenAPI.

В главе 13 показано, как проектировщики могут участвовать в развитии множества API, принимая участие в жизненном цикле API. Особое внимание уделяется руководству по проектированию API и обзору.

Эту книгу следует читать от корки до корки, каждую главу по порядку. Каждая новая глава расширяет то, что было изучено в предыдущих. При этом, как только вы закончите главы 1, 2 и 3, вы можете перейти к любой главе, посвященной теме, которую вам необходимо срочно исследовать.

О коде

В данной книге содержится множество примеров исходного кода как в пронумерованных списках, так и в обычном тексте. В обоих случаях исходный код форматируется `шрифтом фиксированной ширины` наподобие этого, чтобы отделить его от обычного текста. Иногда код также печатается **жирным шрифтом**, чтобы выделить код, который изменился по сравнению с предыдущим вариантом, например когда в существующую строку кода добавляется новая функция.

Во многих случаях оригинальный исходный код был переформатирован; мы добавили разрывы строк и переработали отступы, чтобы разместить доступное пространство страницы в книге. В редких случаях даже этого было недостаточно, и листинги содержат маркеры продолжения строки (➥). Кроме того, комментарии в исходном коде часто удаляются из листингов, когда описание кода приводится в тексте. Многие листинги сопровождаются кодовыми аннотациями для выделения важных понятий.

Исходный код примеров из этой книги доступен для скачивания с веб-сайта издателя по адресу https://www.manning.com/books/the-design-of-web-apis.

Дискуссионный форум liveBook

Покупка книги *The Design of Web APIs* дает право свободного доступа к закрытому веб-форуму издательства Manning Publication, где можно высказать свои замечания о книге, задать технические вопросы и получить помощь от авторов и других пользователей. Чтобы получить доступ к форуму, перейдите по ссылке https://livebook.manning.com/book/the-design-of-everyday-apis/welcome/v-11/discussion. Вы также можете узнать больше о форумах издательства и правилах поведения на них по адресу https://livebook.manning.com/discussion.

Издательство Manning обязуется предоставить читателям площадку для содержательного диалога не только между читателями, но и между читателями и авторами. Но авторы не обязаны выделять определенное количество времени для участия, т. к. их вклад в работу форума является добровольным (и неоплачиваемым). Мы предлагаем читателям задавать авторам действительно непростые вопросы, чтобы их интерес не угасал! Форум и архив предыдущих обсуждений будут доступны на веб-сайте издательства, пока книга находится в печати.

Другие интернет-ресурсы

Существует множество онлайн-ресурсов об API. Вот два моих любимых:

- *API Developer Weekly Newsletter* (https://apideveloperweekly.com/) – лучший способ узнать, что происходит в мире API и открыть для себя новые источники информации об API;
- сайт Web API Events (https://webapi.events/), он будет информировать вас о предстоящих конференциях, посвященных API.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Об авторе

Арно Лоре – архитектор программного обеспечения с 17-летним опытом из Франции. Большую часть этого периода он провел в финансовом секторе, работая над объединением систем различными способами, особенно с использованием веб-сервисов и API. Он ведет блог *API Handyman* и веб-сайт API Stylebook. Его приглашают в качестве лектора на конференции по API по всему миру. Арно увлекается -откой программного обеспечения, ориентированного на человека, и любит создавать и помогать создавать системы, обеспечивающие замечательный опыт для всех пользователей – от разработчиков и рабочих групп до конечных пользователей.

Часть I

Основы проектирования API

Каждое путешествие начинается с первого шага, и проектирование API не является исключением. Проектировщикам API необходимо обладать множеством навыков и нужно учитывать множество тем при создании API, но без прочной основы все передовые навыки и темы ничего не стоят. Это то, о чем вы узнаете в первой части.

Сначала мы рассмотрим ситуацию, объяснив, что такое API, почему его нужно проектировать и что на самом деле означает *обучение проектированию API*. Вы узнаете, что, будучи программными интерфейсами, API – это больше, чем просто «связующее звено», и что для проектирования любого типа API необходимо изучить фундаментальные принципы.

Еще до того, как задуматься о программировании, вы увидите, что API нужно рассматривать с точки зрения его пользователей. Предполагается, что API позволяет вашим пользователям легко достигать своих целей, а не системам, представляющим API. Только после того, как эти цели станут известны и точно описаны, можно спроектировать фактический интерфейс программирования, такой как REST API. И, как и любое программирование, описание интерфейса программирования должно выполняться с помощью адаптированного инструмента, такого как спецификация OpenAPI для REST API.

1 Что такое проектирование API

В этой главе вы узнаете:

- что такое API;
- почему важно проектирование API;
- что означает проектирование API.

Веб-API (программные интерфейсы приложения) являются неотъемлемой частью нашего взаимосвязанного мира. Программное обеспечение использует эти интерфейсы для обмена данными – от приложений на смартфонах до глубоко скрытых серверов баз данных, API-интерфейсы присутствуют абсолютно везде. Считаются ли они простыми техническими интерфейсами или продуктами сами по себе, целые системы, независимо от размера и назначения, зависят от них. То же самое делают целые компании и организации, начиная с технологических стартапов и интернет-гигантов, заканчивая нетехническими малыми и средними предприятиями, крупными корпорациями и государственными структурами.

Если API-интерфейсы являются неотъемлемой частью нашего взаимосвязанного мира, их проектирование – это его основа. При создании и развитии системы на базе API, независимо от того, является ли она видимой для кого-либо или глубоко скрытой, создает ли она один или несколько API-интерфейсов, проектирование всегда должно быть основной задачей. Успех или неудача такой системы напрямую зависит от качества проектирования всех ее API. Но что на самом деле означает проектирование API? И что нужно изучать, чтобы проектировать

их? Чтобы ответить на эти вопросы, нужно рассмотреть, что такое API и для кого они предназначены, а также понять, что проектирование API – это больше, чем просто проектирование программного интерфейса для приложений.

1.1. Что такое API?

Миллиарды людей владеют смартфонами и используют их для обмена фотографиями в социальных сетях. Без API это было бы невозможно. Обмен фотографиями с помощью мобильного приложения для социальных сетей предполагает использование различных типов API, как показано на рис. 1.1.

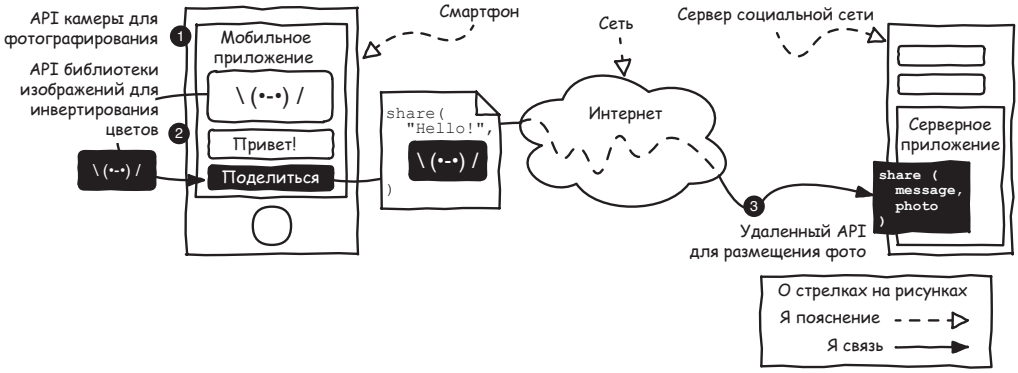

Рис. 1.1. Три разных типа API

Во-первых, чтобы сделать фото, мобильное приложение социальной сети использует камеру смартфона с помощью API. Затем через API оно может использовать некую библиотеку изображений, встроенную в приложение, для инвертирования цветов фотографии. И в итоге оно делится измененной фотографией, отправляя ее на серверное приложение, размещенное на сервере социальной сети, используя удаленный API, доступ к которому можно получить через сеть, обычно через интернет. Таким образом, в этом сценарии задействованы три различных типа API: аппаратный API, библиотека и удаленный API, соответственно. Эта книга посвящена последнему типу.

API-интерфейсы, независимо от своего типа, упрощают создание программного обеспечения, но удаленные API, особенно веб-API, изменили способ создания программного обеспечения. В настоящее время любой может легко создать что-либо, собрав удаленные части программного обеспечения. Но, прежде чем говорить о безграничных возможностях, предоставляемых такими API, давайте разберемся, что на самом деле подразумевается под термином *API* в этой книге.

1.1.1. API – это веб-интерфейс для программного обеспечения

В этой книге API – это удаленный API, а точнее, веб-API – веб-интерфейс для программного обеспечения. API, независимо от типа, прежде всего

является интерфейсом: точкой, где две системы, субъекта, организации и т. д. встречаются и взаимодействуют. Поначалу концепция API может быть непростой для понимания, но рис. 1.2 делает ее более осязаемой, сравнивая ее с пользовательским интерфейсом приложения.

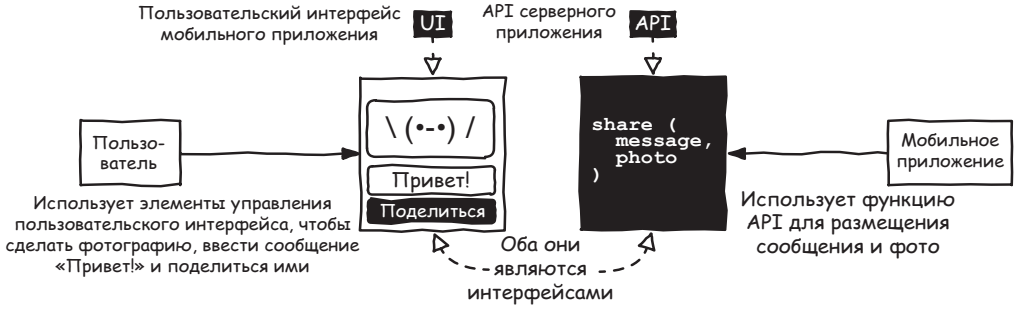

Рис. 1.2. Сравнение пользовательского интерфейса приложения с программным интерфейсом приложения (API)

Будучи пользователем мобильного приложения, вы взаимодействуете с ним, касаясь экрана вашего смартфона, на котором отображается пользовательский интерфейс приложения. Пользовательский интерфейс мобильного приложения может предоставлять такие элементы, как кнопки, текстовые поля или метки на экране. Эти элементы позволяют пользователям взаимодействовать с приложением для просмотра или предоставления информации, а также для запуска таких действий, как обмен сообщениями и фотографиями.

Так же как мы (люди) используем пользовательский интерфейс приложения для взаимодействия с ним, это приложение может использовать еще одно приложение с помощью своего программного интерфейса. Принимая во внимание, что пользовательский интерфейс предоставляет поля ввода данных, метки и кнопки, чтобы обеспечить обратную связь, которая может развиваться по мере их использования, API предоставляет функции, которые могут нуждаться во входных данных или которые могут возвращать выходные данные в качестве ответа. Эти функции позволяют другим приложениям взаимодействовать с приложением, предоставляющим API, для извлечения или отправки информации или для запуска действий.

Строго говоря, API – это *только* интерфейс, предоставляемый неким программным обеспечением. Это абстракция базовой *реализации* (базовый код – это то, что на самом деле происходит внутри программного продукта при использовании API). Но обратите внимание, что термин *API* часто используется для обозначения всего программного продукта, включая API и его реализацию.

Таким образом, API – это интерфейсы для программного обеспечения, но API, о которых мы говорим в этой книге, – это больше, чем просто API: это веб-API, как показано на рис. 1.3.

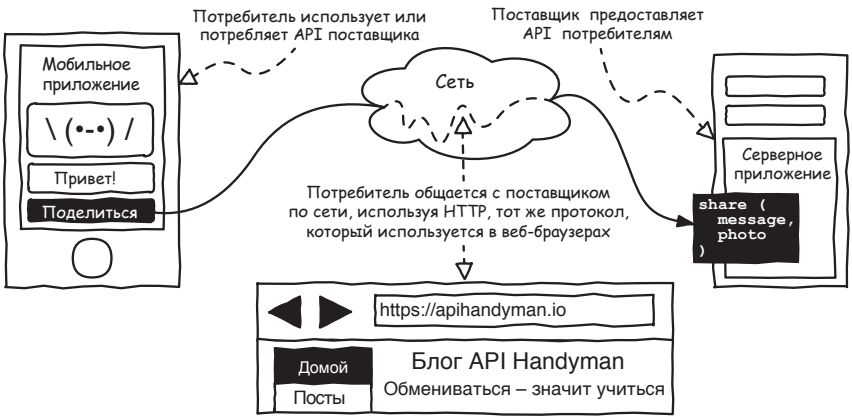

Рис. 1.3. Веб-API – это удаленные API, которые можно использовать с протоколом HTTP

Мобильное приложение, работающее в смартфоне, использует API, предоставляемый серверным приложением (часто именуемым бэкенд-приложением или просто *серверной частью*), которое размещено на удаленном сервере. Поэтому мобильное приложение называется *потребителем*, а серверная часть называется *поставщиком*. Данные термины также применимы к компаниям и людям, создающим приложения, или использующим или предоставляющим API. Здесь это означает, что разработчики мобильного приложения являются потребителями, а разработчики бэкенд-приложения – поставщиками.

Для обмена данными со своим бэкенд-приложением мобильное приложение обычно использует известную сеть: интернет. Здесь интересен не сам интернет – такой обмен данными также может осуществляться через локальную сеть, – а то, *как* эти два приложения обмениваются данными по сети. Когда мобильное приложение отправляет фотографию и сообщение бэкенд-приложению, оно использует Протокол передачи гипертекста (HTTP). Если вы открываете веб-браузер на компьютере или смартфоне, то используете HTTP (косвенно). Это протокол, который применяется любым веб-сайтом. Когда вы вводите адрес веб-сайта, например http://apihandyman.io или его защищенную версию, https://apihandyman.io, в адресной строке и нажимаете **Enter** или щелкаете по ссылке в браузере, браузер использует HTTP для связи с удаленным сервером, на котором размещен веб-сайт, чтобы показать вам содержимое сайта. Удаленные API или, по крайней мере, те, о которых мы говорим в этой книге, используют этот протокол так же, как веб-сайты; вот почему они называются *веб-API*.

Итак, в этой книге API – это веб-API. Это веб-интерфейсы для программного обеспечения. Но чем они так интересны?

1.1.2. API превращают программное обеспечение в детали конструктора LEGO®

Тысячи, даже миллионы мобильных приложений и их бэкендов были созданы благодаря веб-API, но это еще не все. Веб-API развивают твор-

ческий потенциал и инновации, превращая программное обеспечение в многократно используемые блоки, которые можно легко собрать. Давайте вернемся к нашему примеру и посмотрим, что может произойти, если разместить фото в социальной сети.

Когда бэкенд социальной сети получает фотографию и сообщение, он может сохранить фотографию в файловой системе сервера, а сообщение и идентификатор фотографии (для последующего извлечения фактического файла) – в базе данных. Он также может обработать фотографию, используя некий самодельный алгоритм распознавания лиц, чтобы определить, есть ли ней ваши друзья, перед тем как сохранить фотографию. Это одна из возможностей – одно приложение, обрабатывающее все для другого отдельного приложения. Давайте рассмотрим что-нибудь другое, как показано на рис. 1.4.

Внутренний API может использоваться как мобильным приложением социальной сети, так и веб-сайтом, и его реализация может быть совершенно иной. Когда серверная часть получает фотографию и сообщение для совместного использования (какое бы приложение его ни отправляло), она может делегировать хранение фотографии в качестве сервисной компании через свой API. Он также может делегировать хранение идентификатора сообщения и фотографии внутреннему программному модулю хроники через его API. Как обрабатывать распознавание лиц? Что же, это можно было бы делегировать какому-нибудь специалисту по распознаванию лиц, предлагающему свои услуги через... как вы уже догадались... API.

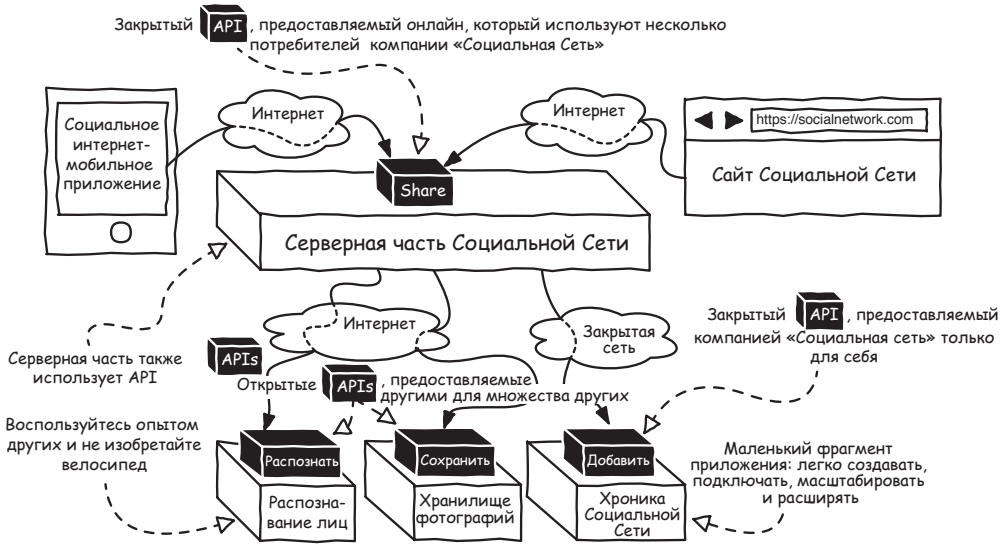

Рис. 1.4. Система, состоящая из открытых и закрытых программных деталей LEGO®, подключенных через API

Обратите внимание, что на рис. 1.4 каждый API предоставляет только одну функцию. Это делает рисунок максимально простым: один API мо-

жет предоставлять множество функций. Серверная часть может, например, предоставлять такие функции, как **Добавить друга**, **Перечислить друзей** или **Получить хронику**.

Это похоже на программную версию системы деталей конструктора LEGO; это те пластиковые блоки, которые можно собирать, чтобы создавать новые вещи. (Аристотель, вероятно, играл с некоторыми из них, когда понял, что «целое больше, чем сумма его частей».) Возможности тут бесконечны – единственным ограничением является ваше воображение.

Когда я был ребенком, я целыми часами играл в LEGO, собирая здания, машинки, самолеты, космические корабли или все что угодно. Когда мне надоедало одно из моих творений, я мог полностью уничтожить его и начать что-то новое с нуля или мог переделать его, заменив некоторые детали. Я мог бы даже собрать существующие структуры вместе, например чтобы создать массивный космический корабль. То же самое происходит и в мире API: вы можете разложить огромные системы программных блоков, которые можно легко собрать и даже заменить благодаря API, но есть небольшие отличия.

Каждый программный блок может одновременно использоваться и многими другими. В нашем примере внутренний API может использоваться мобильным приложением и веб-сайтом. Обычно API предназначен для использования не одним потребителем, а множеством. Таким образом, вам не нужно все время приступать к повторной разработке.

Каждый программный блок может работать где угодно самостоятельно, если он подключен к сети, чтобы быть доступным через API. Это хороший способ управления производительностью и масштабируемостью; программный блок, такой как блок распознавания лиц на рис. 1.4, вероятно, потребует гораздо больше ресурсов обработки по сравнению с хроникой социальной сети. Если первым управляет компания Social Network, его можно установить на другом, выделенном и более мощном сервере, тогда как хроника Social Network работает на сервере поменьше. А доступность через простое сетевое соединение позволяет любому API, предоставленному кем-либо, использоваться кем-либо.

В мире API существует два типа блоков, представляющих два типа API: *открытые API* и *закрытые API*. Программные блоки распознавания лиц и хранения фотографий создаются и даже управляются не компанией Social Network, а третьими лицами. Предоставляемые ими API являются открытыми.

Открытые API-интерфейсы предлагаются *в качестве сервиса* или *продукта* другими; вы не создаете их, не устанавливаете и не запускаете – вы только используете их. Открытые API предоставляются всем, кто в них нуждается и готов принять условия стороннего поставщика. В зависимости от бизнес-модели провайдеров API такие API могут быть бесплатными или платными, как и любой другой программный продукт. Такие открытые API-интерфейсы раскрывают творческий потенциал и инновации, а также могут значительно ускорить создание всего того, о чем вы можете мечтать. Перефразируя Стива Джобса, можно сказать,

что для этого есть API. Зачем терять время, пытаясь изобрести колесо, которое все равно будет недостаточно круглым? В нашем примере компания Social Network решила сосредоточиться на своем основном опыте, соединяя людей, и делегировала распознавание лиц третьей стороне.

Но открытые API – это только вершина айсберга. Серверная часть и временная шкала были созданы компанией Social Network для собственного использования. API хроники используются только приложениями, созданными этой компанией, как показано на рис. 1.4. То же самое касается мобильного бэкенда, который используется мобильным приложением Social Network. Эти API являются *закрытыми*, и их – миллиарды. Закрытый API – это API, который вы создаете для себя: его используют только приложения, созданные вами или сотрудниками вашей команды, отдела или компании. В данном случае вы являетесь поставщиком и потребителем собственного API.

ПРИМЕЧАНИЕ. Вопрос открытости/закрытости не в том, *как* API предоставляется, а *кому*. Даже будучи размещенным в интернете, API мобильного бэкенда по-прежнему является закрытым.

Между *настоящими* закрытыми и открытыми API могут быть различные виды взаимодействия. Например, вы можете установить коммерческое готовое программное обеспечение, такое как система управления контентом (CMS) или система управления взаимоотношениями с клиентами (CRM), на собственные серверы (такая установка часто называется *on-premise*), и эти приложения могут (и даже должны!) предоставлять API. Эти API являются закрытыми, но вы не создаете их сами. Тем не менее вы можете использовать их по своему усмотрению, в особенности для подключения большего количества блоков к своим блокам. Возьмем другой пример. Вы можете предоставить некоторые из ваших API-интерфейсов клиентам или выбранным партнерам. Такие *почти открытые* API часто называют *партнерскими*.

Но независимо от ситуации API в основном превращают программное обеспечение в программные блоки многократного использования, которые вы или другие пользователи могут собрать, чтобы создать модульные системы, способные выполнять абсолютно все. Вот почему API так интересны. Но почему их разработка так важно?

1.2. Чем важно проектирование API

Даже если это полезно, API кажется лишь техническим интерфейсом для программного обеспечения. Так в чем состоит важность проектирования такого интерфейса?

API-интерфейсы используются программным обеспечением, это правда. Но кто создает программное обеспечение, которое их использует? Разработчики. Люди. Эти люди ожидают, что эти программные интерфейсы будут полезными и простыми, как и любой другой (хорошо спроектированный) интерфейс. Подумайте о своей реакции, когда вы сталкиваетесь с плохо спроектированным веб-сайтом или пользователь-

ским интерфейсом мобильного приложения. Что вы чувствуете, когда сталкиваетесь с плохо продуманными повседневными вещами, такими как пульт дистанционного управления или даже дверь? Это может вызвать у вас раздражение, вероятно, вы даже рассердитесь или разразитесь тирадой и вряд ли захотите когда-либо пользоваться ими. А в некоторых случаях плохо спроектированный интерфейс может быть даже опасным. Вот почему проектирование любого интерфейса имеет значение, и API не являются исключением.

1.2.1. Открытый или закрытый API – это интерфейс для других разработчиков

В разделе 1.1.1 вы узнали, что потребителем API может быть либо программное обеспечение, использующее API, либо компания, либо отдельные лица, разрабатывающие это программное обеспечение. Все эти потребители важны, но первый, кого нужно принимать во внимание, – это разработчик.

Как вы уже видели, в этом примере с социальными сетями участвуют разные API. Существует модуль хроники, который управляет хранением данных и предоставляет закрытый API. И есть открытые (предоставляемые другими компаниями) API распознавания лиц и хранения фотографий. Бэкенд, который вызывает эти три API, не появляется сам по себе; он разработан компанией Social Network.

Например, чтобы использовать API распознавания лиц, разработчики пишут код в программном обеспечении Social Network, чтобы отправлять фотографии для распознавания лиц и обрабатывать результат обработки фотографий, как при использовании программной библиотеки. Эти разработчики не являются теми, кто создал API распознавания лиц, и они, вероятно, не знают друг друга, потому что они из разных компаний. Мы также можем представить, что мобильное приложение, веб-сайт, серверная часть и модуль хранения данных разрабатываются разными группами внутри компании. В зависимости от организации компании эти команды могут хорошо знать друг друга или не знать вообще. И даже если каждый разработчик в компании знает все маленькие секреты каждого API, который она разработала, неизбежно появятся новые разработчики.

Таким образом, будь то открытый или закрытый API независимо от того, по какой причине он создан и какой бы ни была организация компании, рано или поздно он будет использоваться другими разработчиками – людьми, которые не участвовали в создании программного обеспечения, предоставляющего API. Вот почему нужно сделать все, для того чтобы при написании кода этим новичкам было легко использовать API. Разработчики ожидают, что API будут полезны и просты, как и любой интерфейс, с которым им приходится взаимодействовать. Вот почему так важно проектирование API.

Опыт разработчика

API (DX) – это опыт, который получают разработчики при использовании API. Он охватывает множество различных тем, таких как регистрация (для использования API), документирование (чтобы узнать, что делает API и как его использовать) или поддержка (чтобы получить помощь при возникновении проблем). Но все усилия ничего не стоят, если не решен самый важный вопрос – проектирование API.

1.2.2 API создается, для того чтобы скрыть реализацию

Проектирование API имеет значение, потому что, когда люди используют API, они хотят использовать его, не беспокоясь о мелочах, которые не имеют к ним никакого отношения. А для этого разработка должна скрывать детали реализации (что на самом деле происходит). Позвольте мне использовать реальную аналогию в качестве объяснения.

Скажем, вы решили пойти в ресторан. Как насчет французского, например? Когда вы идете в ресторан, то становитесь клиентом. Как клиент ресторана, вы читаете его меню, чтобы узнать, какие блюда можно заказать. Вы решаете попробовать миногу по-бордосски (знаменитое французское рыбное блюдо из области Гасконь). Чтобы заказать выбранную еду, вы говорите с (обычно очень приятным и дружелюбным) человеком, которого называют официантом или официанткой. Спустя некоторое время официант возвращается и приносит заказанное вами блюдо – миногу по-бордосски, – которое приготовили на кухне. Пока вы едите свой вкусный обед, можно задать вам два вопроса?

Первое: вы знаете, как приготовить миногу по-бордосски? Наверное, нет, и, возможно, по этой причине вы и идете в ресторан. И даже зная рецепт приготовления, вы, вероятно, не захотите этого делать, потому что это сложно и для этого требуются труднодоступные ингредиенты. Вы отправляетесь в ресторан за блюдом, которое не умеете или не хотите готовить.

Второе: знаете ли вы, что произошло между моментом, когда официант принял ваш заказ и когда принес его вам? Вы можете догадаться, что официант был на кухне, чтобы отдать заказ повару, который работает один. Этот повар очень старается и уведомляет официанта, когда блюдо будет готово, звоня в маленький колокольчик и крича: «Заказ для столика № 2 готов!» Но сценарий может немного отличаться.

Официант может использовать смартфон, чтобы принять ваш заказ, который мгновенно отображается на сенсорном экране на кухне. А там не одинокий повар, а целая бригада. Как только блюдо готово, один из членов бригады помечает ваш заказ как готовый на сенсорном экране, а официант получает уведомление на свой смартфон. Независимо от количества поваров, вы не знаете рецепт и ингредиенты, используемые для приготовления еды. Независимо от сценария, еда, которую вы заказали, поговорив с официантом, была приготовлена на кухне, и официант принес ее вам. В ресторане вы говорите только с официантом (или офи-

цианткой), и вам не нужно знать, что происходит на кухне. Какое отношение все это имеет к API? Самое прямое, как показано на рис. 1.5.

С точки зрения группы разработчиков мобильных приложений для социальных сетей, *размещение фото* с использованием бэкенд-API абсолютно одинаково, только бэкенд уже реализован. Разработчикам не нужно знать рецепт и его ингредиенты; они только предоставляют фото и сообщение. Им все равно, будет ли фотография проходить через распознавание изображений до или после добавления в хронику. Они также не заботятся о том, является ли серверная часть единственным приложением, написанным на языке Go или Java, которое обрабатывает все или использует другие API, написанные на NodeJS, Python или любом другом языке. Когда разработчик создает *потребительское приложение* (клиент), которое использует *приложение провайдера* (ресторан) через свой API (официант или официантка), чтобы сделать что-то (например, заказать еду), разработчик и приложение знают только об API; им не нужно знать, как сделать что-то самостоятельно или как программное обеспечение провайдера (кухня) будет это делать. Но скрыть реализацию недостаточно. Это важно, но это не то, что на самом деле нужно тем, кто использует API.

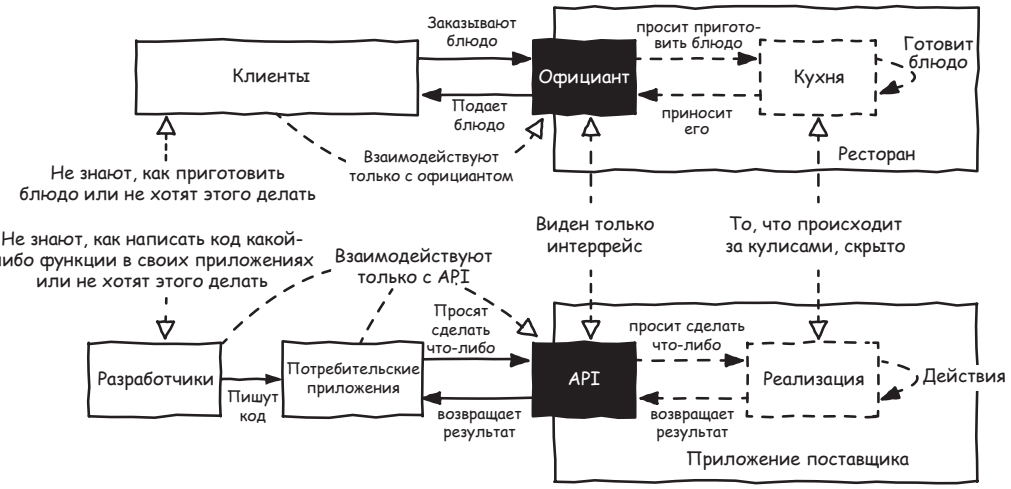

Рис. 1.5. Параллели между обедом в ресторане и использованием API

1.2.3. Страшные последствия плохо спроектированных API

Что вы делаете, когда впервые используете какую-либо повседневную вещь? Вы внимательно смотрите на ее интерфейс, чтобы определить ее назначение и то, как ее использовать, основываясь на том, что вы видите, и на своем прошлом опыте. И здесь важен дизайн.

Давайте рассмотрим гипотетический пример: устройство под названием UDRC 1138. Что это может быть за устройство? Какова его цель? Ну, его название не очень помогает. Возможно, его интерфейс может дать нам больше подсказок. Посмотрите на рис. 1.6.

Рис. 1.6. Загадочный интерфейс устройства UDRC 1138

Справа есть шесть немаркированных треугольных и прямоугольных кнопок. Может, это что-то вроде запуска и остановки? Ну, как кнопки медиаплеера. Четыре другие кнопки имеют незнакомую форму без малейшего намека на их назначение. На ЖК-дисплее отображаются номера без меток с такими единицами измерения, как ft, NM, rad и km/h. Можно предположить, что *ft* – это расстояние в футах, а *km/h* – скорость в километрах в час, но что могут означать *rad* и *NM*? Кроме того, в нижней части ЖК-экрана также появляется тревожное сообщение, предупреждающее о том, что мы можем вводить значения вне допустимого диапазона без контроля безопасности.

Этот интерфейс, безусловно, трудно расшифровать, и он не очень интуитивно понятен. Давайте посмотрим на документацию, изображенную на рис. 1.7, чтобы увидеть, что в ней говорится об этом устройстве.

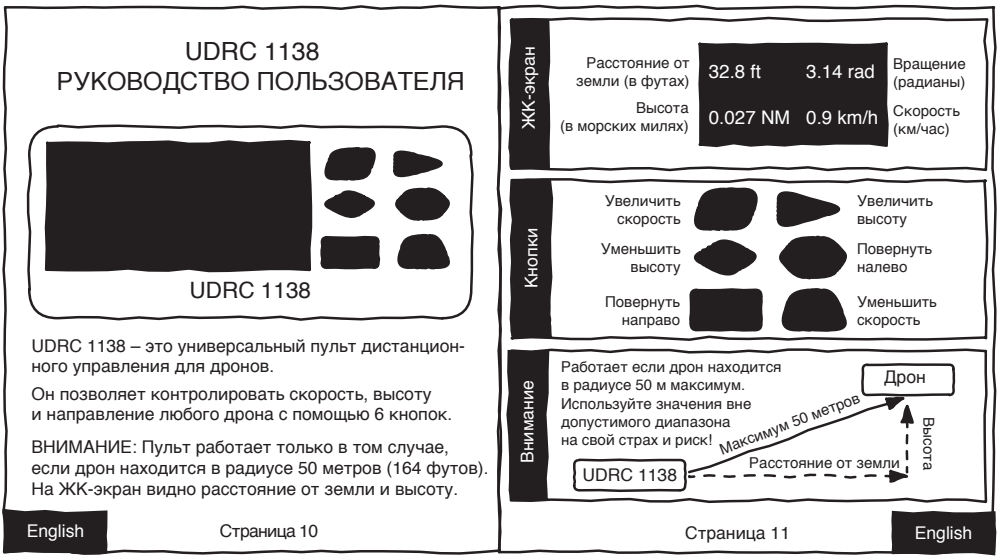

Рис. 1.7. Документация по UDRC 1138

Согласно описанию, это устройство представляет собой универсальный пульт дистанционного управления дроном – отсюда и название UDRC 1138, я полагаю, – который можно использовать для управления любым дроном. Ну что же, звучит интересно. На странице справа приве-

дено несколько пояснений по ЖК-экрану, кнопкам и сообщение с предупреждением.

Описание ЖК-экрана довольно загадочное. Расстояние от земли указано в футах, высота – в морских милях, ориентация дронов обеспечивается радианами, а скорость указана в километрах в час. Я не знаком с авиационными единицами измерения, но чувствую, что с выбранными единицами что-то не так. Они кажутся противоречивыми. Смесь футов и метров? И во всех фильмах о самолетах, которые я видел, футы используются для измерения высоты, а морские мили – для измерения расстояния, а не наоборот. Этот интерфейс определенно не интуитивно понятен.

Глядя на описания кнопки, видно, что для увеличения высоты используется треугольная кнопка в правом верхнем углу, а для ее уменьшения – ромбовидная кнопка во втором ряду. Связь неочевидна – почему эти элементы управления не идут друг за другом? Другие элементы управления также используют формы, которые абсолютно ничего не значат и кажутся расположенными случайным образом. Это безумие! Как использовать это устройство? Почему бы не взять старые добрые джойстики или геймпады вместо кнопок?

И последнее, но не менее важное объяснение предупреждающего сообщения. Похоже, что беспилотник может работать только в диапазоне 50 м – диапазон рассчитывается на основе расстояния до земли и высоты, обеспечиваемой ЖК-экраном. Подождите, что? Мы должны рассчитать расстояние между пультом дистанционного управления и дроном, используя теорему Пифагора?! Это полная чушь – устройство должно делать это за нас.

Вы бы купили или хотя бы попробовали в деле такой гаджет? Вероятно, нет. Но что, если у вас нет другого выбора, кроме как использовать это ужасное устройство? Что же, желаю удачи; вам понадобится что-то сделать с таким плохо спроектированным интерфейсом!

Вы, вероятно, думаете, что такой катастрофический интерфейс на самом деле не может существовать в реальной жизни. Конечно, проектировщики не могли создать такой ужасный прибор. И даже если бы они это сделали, конечно, отдел обеспечения качества никогда бы не пустил его в производство! Но давайте посмотрим правде в глаза. Плохо спроектированные устройства – вещи с плохо спроектированными интерфейсами для повседневного использования – все время *поступают* в производство.

Подумайте, сколько раз вы были озадачены или ворчали при работе с устройством, веб-сайтом или приложением, потому что в его дизайне были дефекты. Сколько раз вы решали не покупать или не использовать что-то из-за его дизайна? Сколько раз вы не могли применить что-то, или применить это правильно, или как вы хотели, потому что его интерфейс был непостижим?

Плохо спроектированный продукт может быть использован не по назначению, недоиспользован или не использован вообще. Это может даже быть опасно для его покупателей и для компании, которая созда-

ла его, и репутация такой компании может быть под угрозой. И если это физическое устройство, после того как оно будет запущено в производство, исправлять его будет слишком поздно.

Страшные недостатки дизайна не ограничиваются интерфейсами повседневных вещей. К сожалению, API также могут страдать от этого. Напомню, что API – это интерфейс, который разработчики должны использовать в своем программном обеспечении. Проектирование имеет значение независимо от типа интерфейса, и API-интерфейсы не являются исключением. Плохо спроектированные API-интерфейсы могут стать таким же разочарованием, что и устройства, подобные UDRC 1138; они могут быть очень трудным для понимания и использования, а это может иметь ужасные последствия.

Как люди выбирают открытый API, API в качестве продукта? Как и в повседневной жизни, они смотрят на его интерфейс и документацию. Они заходят на портал разработчиков API, читают документацию и анализируют API, чтобы понять, что он позволяет делать и как его использовать. Они оценивают, позволит ли он им эффективно и просто достичь своей цели. Даже самая совершенная документация не сможет скрыть недостатки проектирования, которые делают API трудным или даже опасным для использования. И если такие недостатки будут обнаружены, возможные пользователи (эти потенциальные клиенты) не станут выбирать этот API. Нет клиентов – нет дохода. Это может привести к банкротству компании.

Что, если кто-то решит воспользоваться некорректным API в любом случае? Иногда пользователи могут просто не обнаружить недостатки с первого взгляда. А иногда у них просто может не быть альтернативы. Такое может произойти, например, внутри какой-то компании. Чаще всего у людей нет другого выбора, кроме как использовать ужасные закрытые API или API, предоставляемые коммерческими готовыми приложениями.

Независимо от контекста недостатки проектирования увеличивают время, усилия и траты, необходимые для создания программного обеспечения с использованием API. API может быть использован неправильно или недоиспользован. Клиентам может потребоваться обширная поддержка со стороны поставщика API, что увеличивает затраты на стороне поставщика. Это потенциально серьезные последствия как для закрытых, так и для открытых API. Более того, благодаря открытым API-интерфейсам пользователи могут открыто пожаловаться или просто прекратить их применение, что приведет к уменьшению количества клиентов и снижению доходов для поставщиков API.

Ошибочное проектирование API также может привести к уязвимостям безопасности в API, таким как непреднамеренное раскрытие конфиденциальных данных, пренебрежение правами доступа и привилегиями группы или слишком большое доверие к потребителям. Что, если некоторые решат воспользоваться такими уязвимостями? Последствия для потребителей и поставщиков API могут быть катастрофическими.

В мире API плохое проектирование часто можно исправить, после того как API будет запущен в производство. Но издержки неизбежны:

поставщику понадобятся время и деньги, чтобы исправить ситуацию, и это может также серьезно беспокоить потребителей API.

Это лишь несколько примеров вредного воздействия. Плохо спроектированные API обречены на провал. Что можно сделать, чтобы избежать подобной участи? Все просто: *научиться правильно проектировать API*.

1.3. Элементы проектирования API

Научиться проектировать API – это гораздо больше, чем просто научиться проектировать программные интерфейсы. Обучение проектированию API требует изучения принципов, и не только технологий, но и знания всех аспектов проектирования. Для проектирования API важно не просто сосредоточиться на самих интерфейсах, но также знать весь контекст, окружающий их, и проявлять эмпатию ко всем пользователям и программному обеспечению. Проектирование API без принципов, полностью вне контекста и без учета обеих сторон интерфейса – как потребителя, так и поставщика, – лучший способ гарантировать полный провал.

1.3.1 Изучение принципов, выходящих за рамки проектирования программного интерфейса

Когда (хорошие) проектировщики помещают кнопку в определенное место, выбирают конкретную форму или решают добавить красный светодиод на объект, на то есть причина. Знание этих причин помогает проектировщикам создавать хорошие интерфейсы для повседневных вещей, которые позволят людям достичь своей цели как можно проще – будь то двери, стиральные машины, мобильные приложения или что-либо еще. То же самое происходит и в случае с API.

Цель API состоит в том, чтобы позволить людям достичь своих целей настолько просто, насколько это возможно, независимо от части, касающейся *программирования*. В программном обеспечении мода приходит и уходит. Было и будет много разных способов раскрытия данных и возможностей с помощью программного обеспечения. Было и будет много разных методов обеспечения обмена данными по сети. Возможно, вы слышали о RPC, SOAP, REST, gRPC или GraphQL. Вы можете создавать API-интерфейсы с помощью всех этих технологий: некоторые из них – это архитектурные стили, другие – протоколы или языки запросов. Чтобы было проще, будем называть их *стилями API*.

Каждый стиль API может сопровождаться некоторыми (более или менее) общепринятыми практиками, которым вы можете следовать, но они не защитят вас от ошибок. Не зная основополагающих принципов, вы можете растеряться при выборе так называемой общепринятой практики; вы можете изо всех сил пытаться найти решения, сталкиваясь с необычными случаями использования или контекстами, которые подобного рода практики не охватывают. И если вы переключитесь на новый стиль API, вам придется все изучать заново.

Знание основополагающих принципов проектирования API дает вам прочную основу для проектирования API любого стиля и решения лю-

бых задач проектирования. Но знание таких принципов – лишь один из аспектов проектирования.

1.3.2 Изучение всех аспектов проектирования API

Проектирование интерфейса – это гораздо больше, чем просто размещение кнопок на поверхности объекта. Создание такого объекта, как пульт дистанционного управления дроном, требует, чтобы проектировщики знали, какова его цель и чего хотят добиться люди, использующие его. Предполагается, что такое устройство должно контролировать скорость, высоту и направление летящего объекта. Это то, что нужно пользователям, и им все равно, будет ли это сделано с использованием радиоволн или любой другой технологии.

Все эти действия должны быть представлены в пользовательском интерфейсе с помощью кнопок, джойстиков, ползунков или других элементов управления. Назначение этих элементов управления и их представлений должно иметь смысл, чтобы у пользователей не возникало проблем при их применении, и, что самое важное, они должны быть полностью безопасными. Интерфейс UDRC 1138 является прекрасным примером абсолютно непригодного и небезопасного интерфейса с его ЖК-дисплеем, который сбивает с толку, или с кнопками и отсутствием управления безопасностью.

Конструкция такого пульта дистанционного управления также должна учитывать весь контекст. Как его станут использовать? Например, если он будет использоваться в условиях сильного холода, было бы разумно применять элементы управления, с которыми можно работать в громоздких перчатках. Кроме того, базовая технология может добавлять ограничения для интерфейса. Например, нельзя выполнять передачу приказа дрону более X раз в секунду.

Наконец, как такое можно запускать в производство? Возможно, принимавшие в этом участие проектировщики не были достаточно обучены и не получили должного руководства. Или этот дизайн так и не был утвержден. Если бы только кто-то – возможно, потенциальные пользователи – рассмотрел его, эти явные недостатки, вероятно, можно было бы исправить. Возможно, проектировщики создали хорошую модель, но в конце концов их план так и не был соблюден.

Независимо от качества, как только люди привыкнут к предмету и его интерфейсу, изменения нужно проводить с особой осторожностью. Если новая версия этого пульта дистанционного управления будет иметь совершенно иную организацию кнопок, пользователи, возможно, не захотят покупать новую версию, потому что им придется заново учиться работать с ним.

Как видите, проектирование интерфейса объекта требует сосредоточиться не только на кнопках. То же самое касается и проектирования API.

Проектирование API – это гораздо больше, чем просто проектирование простого и понятного интерфейса. Мы должны создать полностью безопасный интерфейс, ограничивая потребителям доступ к конфиденциальным данным или не давая им совершать лишние действия. Необ-

ходимо принимать во внимание весь контекст – каковы ограничения, как и кем будет использоваться API, как он создается и как может развиваться. Мы должны участвовать во всем жизненном цикле API – от первых обсуждений до проектирования, документирования, развития или вывода из эксплуатации. И поскольку компании обычно создают множество API, мы должны работать вместе с остальными проектировщиками, чтобы гарантировать, что все API организации имеют одинаковый внешний вид, чтобы создавать отдельные интерфейсы, которые были бы максимально согласованными, гарантируя таким образом, что все эти API так же просты для понимания и легки в использовании, как и каждый в отдельности.

Резюме

- Веб-API превращают программное обеспечение в блоки многократного использования, которые можно использовать по сети с помощью протокола HTTP.
- API – это интерфейсы для разработчиков, которые создают приложения, использующие их.
- Дизайн API важен для всех API – открытых или закрытых.
- Плохо спроектированные API-интерфейсы могут быть использованы недостаточно, неправильно или вообще не использоваться, и даже небезопасны.
- Проектирование хорошего API требует, чтобы вы учитывали весь контекст приложения, а не только сам интерфейс.