

ГЛАВА 5

Диаграмма компонентов

В этой главе мы поговорим о *диаграмме компонентов* модели С4. Эта диаграмма позволяет рассмотреть компоненты, находящиеся внутри отдельного контейнера.

Ключевые особенности диаграммы компонентов

Чтобы разобраться, зачем и как применять эту диаграмму, обратите внимание на ее назначение, охват, содержание, варианты использования и целевую аудиторию.

Назначение

Диаграмма компонентов помогает ответить на следующие вопросы, в основном касающиеся контейнеров-приложений, а не хранилищ данных.

- Как контейнер разбит на компоненты?
- За что отвечает каждый компонент?
- Как компоненты взаимодействуют между собой?
- Какие основные технологии (например, фреймворки, библиотеки) используются для реализации компонентов?
- Где разработчику нужно писать код для добавления новой функции?

При составлении диаграммы на этапе предварительного проектирования у вас может не быть некоторых технических деталей, например информации о фреймворках и библиотеках. Ничего страшного — укажите то, что знаете. Если же диаграмма документирует существующую систему, все необходимые детали вам уже доступны.

Охват

Область охвата диаграммы компонентов — *один контейнер*. Если программная система состоит, к примеру, из трех приложений, может возникнуть соблазн показать компоненты всех трех контейнеров на одной диаграмме. Но я рекомендую нарисовать три отдельные диаграммы — по одной на каждый контейнер. В результате получатся три простые и компактные диаграммы, и тем, кто их будет просматривать, не придется вникать во внутреннее устройство сразу трех приложений.

Содержание

Диаграмма компонентов обычно включает шесть типов элементов: пользователей-людей, программные системы и контейнеры (из диаграммы контейнеров), а также сами компоненты, границу контейнера и границу программной системы.

Люди, программные системы и контейнеры

Диаграмма компонентов детализирует один из контейнеров из диаграммы контейнеров, который окружен людьми, программными системами и другими контейнерами. Эти же элементы нужно повторить на диаграмме компонентов для сохранения преемственности. Вы продолжаете рассказывать все ту же историю, но более подробно.

Компоненты

Диаграмма компонентов показывает набор компонентов, существующих в границах контейнера, который мы рассматриваем. Для каждого компонента я указываю следующую информацию.

- *Название* — наименование компонента.
- *Технология* — основная технология реализации.
- *Описание* — краткая сводка обязанностей компонента.

В контексте приложения компонент — это группа связанного кода. Поэтому диаграмма компонентов должна отражать и сами компоненты (группы кода) внутри контейнера, и используемый архитектурный стиль. Если в основе приложения лежит многоуровневая архитектура, то и компоненты на диаграмме должны быть организованы в слои. Мы рассмотрим подобный пример в главе 11, когда перейдем от основ к более сложным темам. Если же используется паттерн «порты и адаптеры» (иначе говоря, «гексагональная архитектура»), диаграмма должна это отражать.

Граница контейнера

Во избежание путаницы советую заключать компоненты в рамку, обозначающую границу контейнера. Эта граница должна соответствовать блоку данного контейнера на диаграмме контейнеров.

Граница программной системы

Чтобы точнее отразить иерархию элементов, можно дополнительно обозначить границу родительской программной системы. Пример мы вскоре рассмотрим.

Связи

Как и для других диаграмм, советую использовать однонаправленные стрелки с указанием следующей информации.

- *Описание* — краткая характеристика связи (например, «считывает данные из», «проверяет токен с помощью»).
- *Технология* — если это применимо, основной протокол (например, JSON/HTTP, gRPC).

Стоит отметить, что большинство взаимодействий между компонентами — это вызовы методов или функций в пространстве одного процесса. В таком случае технологию можно не указывать. Однако, как и ранее, можно обозначить синхронность или асинхронность взаимодействия, например, сплошной и пунктирной линиями соответственно.

Почему такая диаграмма полезна

Эта диаграмма показывает компоненты внутри отдельного контейнера. Она полезна тем, что:

- демонстрирует высокоуровневую декомпозицию контейнера на компоненты, что дает представление о структуре кодовой базы данного приложения;
- показывает связи и зависимости между компонентами;
- дает общее представление о деталях реализации, включая фреймворки и библиотеки.

Аудитория

Диаграмма компонентов предназначена для технических специалистов из команды разработки (архитекторов, разработчиков) и для всех, кому важна структура кода (например, специалисты третьей линии поддержки, инженеры по обслуживанию). Для нетехнических специалистов диаграмма может оказаться полезной, но целевой аудиторией они не являются.

Рекомендуется ли строить эту диаграмму?

В общем случае — нет. Я не делаю такие диаграммы для контейнеров-хранилищ данных (таких как базы данных, файловые системы, хранилища контента), потому что их лучше документировать с помощью таких подходов, как ER-диаграммы. Кроме того, я бы не стал создавать диаграммы компонентов для очень простых контейнеров-приложений — например, для приложения с единственной конечной точкой API. Такое встречается в микросервисной архитектуре, где каждый сервис выполняет одну конкретную функцию.

Диаграммы компонентов лучше подходят для документирования внутренней структуры крупного приложения, но я все же рекомендую их большинству команд разработчиков в качестве *опционального уровня детализации* — на этапе проектирования и для документации, которая создается на длительный срок.

Если я проектирую небольшое приложение, бывает полезно набросать диаграмму компонентов, чтобы проиллюстрировать первоначальный замысел предполагаемой высокоуровневой структуры кода. Однако для крупных приложений этот процесс будет слишком трудоемким.

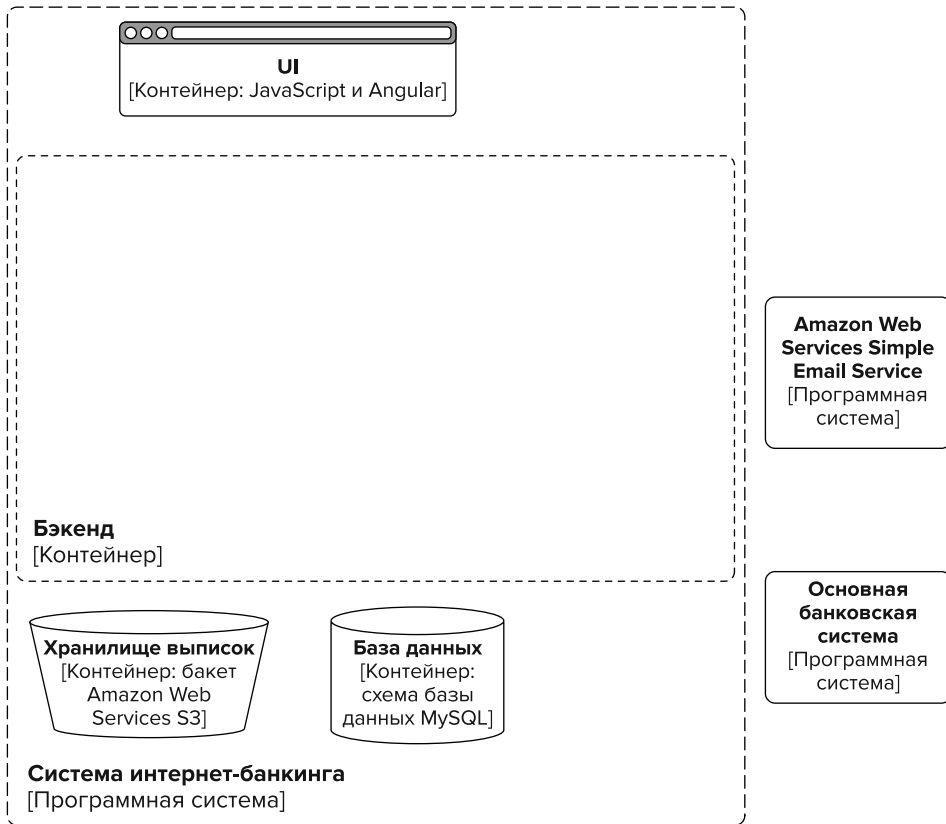
То же справедливо и для долгосрочной документации существующего приложения: добавляются затраты на ее обновление при каждом изменении кода. В отличие от редко меняющихся диаграмм контекста и контейнеров диаграммы компонентов очень изменчивы и быстро устаревают. Снизить влияние этой проблемы поможет инструмент, автоматически генерирующий диаграмму компонентов из кодовой базы. Однако для большинства языков программирования это нетривиальная задача, и затраты на ее решение могут превысить пользу от диаграмм, особенно при хорошо структурированной кодовой базе. Инструменты на основе ИИ могли бы с этим справиться. Однако стоит учитывать, что из-за своей недетерминированной природы при повторном запуске они нередко генерируют разные диаграммы для одной и той же задачи.

Кроме прочего, важна и визуальная составляющая. Уже при небольшом увеличении количества компонентов диаграмма быстро становится перегруженной и менее полезной. Стратегии для решения этой проблемы мы рассмотрим в главе 11.

Итак, диаграмма компонентов — полезный инструмент, но я бы не рекомендовал его всем без исключения командам разработчиков.

Пример

Продолжим разбирать пример с системой интернет-банкинга и детализируем контейнер бэкенда, чтобы показать его компоненты. Исходная диаграмма компонентов показана на рис. 5.1. Она включает те же элементы, что и диаграмма контейнеров из главы 4: UI, базу данных, хранилище выписок, основную банковскую систему и Amazon Simple Email Service. Блок «Бэкенд» (ограниченный точечной рамкой) — это граница контейнера, внутри которой разместятся компоненты. Граница всей программной системы охватывает всю зону этого контейнера наряду с остальными контейнерами (UI, база данных, хранилище выписок). Для удобства восприятия представленных в этой главе диаграмм и из-за ограничений печатного формата из всех элементов удалено описание.

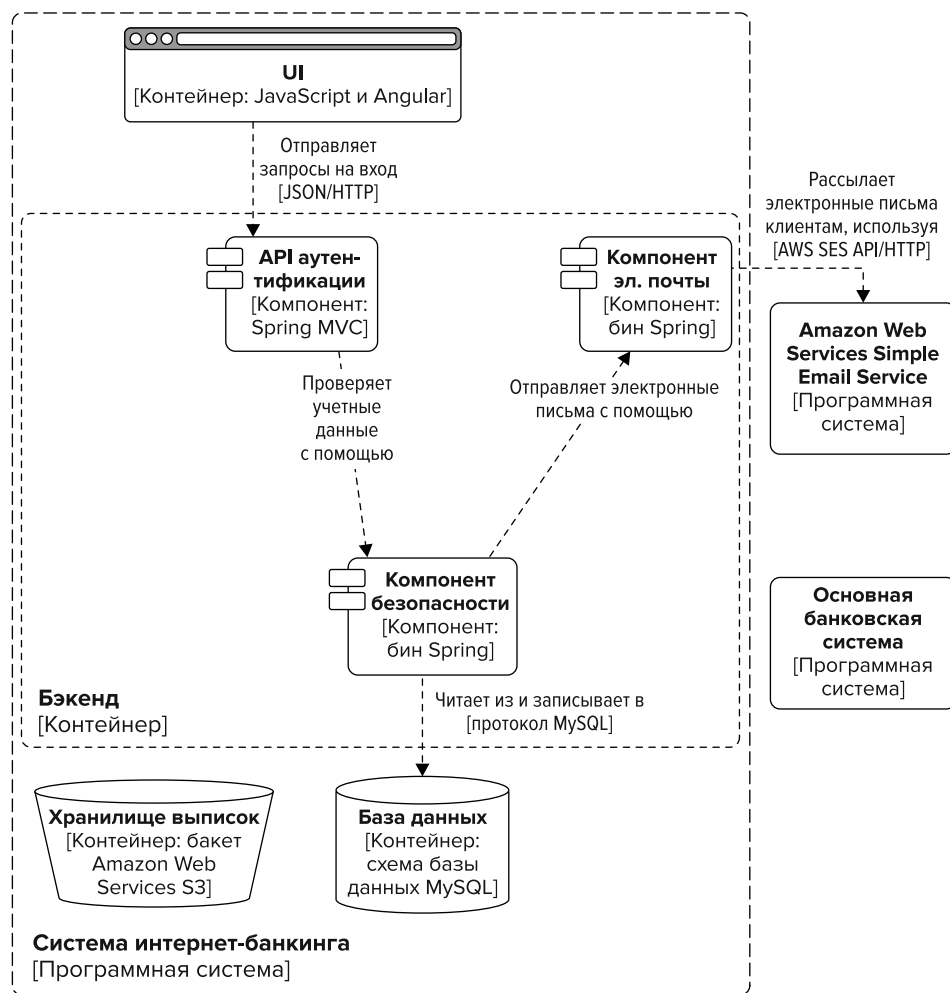


Представление компонентов: система интернет-банкинга, бэкенд

Рис. 5.1. Пример диаграммы компонентов для бэкенда вымышленной системы интернет-банкинга (шаг 1)

Для доступа к функциям системы интернет-банкинга требуется авторизация, поэтому начнем с процесса входа в систему (рис. 5.2). В этом примере UI через JSON/HTTP API отправляет запрос к API аутентификации, построенному с помощью фреймворка Spring Web MVC.

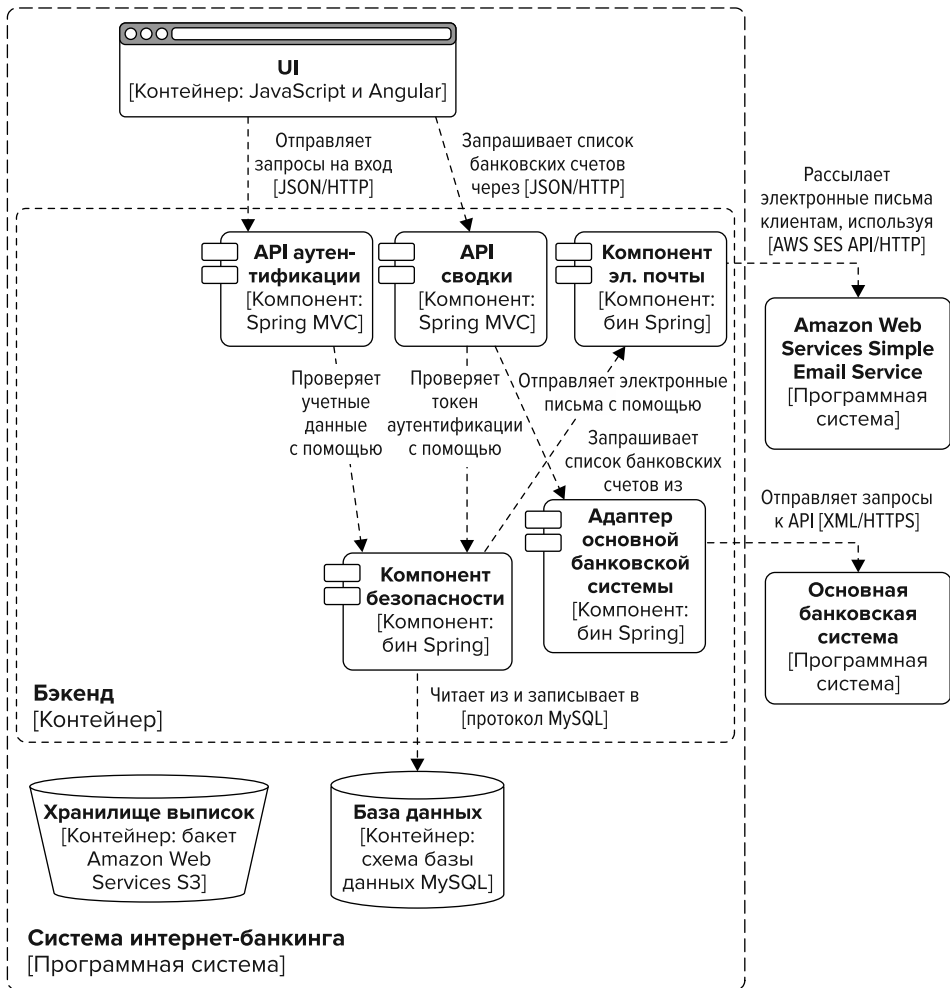
Этот API, в свою очередь, использует компонент безопасности — бин Spring (Java-объект, которым управляет фреймворк Spring) — для проверки учетных данных по базе данных и выпуска токена аутентификации. Компонент безопасности также может рассылать письма через почтовый компонент (еще один бин Spring), например, при многофакторной аутентификации или при входе из новой локации.



Представление компонентов: система интернет-банкинга, бэкенд

Рис. 5.2. Пример диаграммы компонентов для бэкенда вымышленной системы интернет-банкинга (шаг 2)

Как видно из рис. 5.3, UI запрашивает список счетов клиента посредством вызова API JSON/HTTP, который обрабатывает API сводки по счетам (также построенному с помощью Spring Web MVC). Этот API, в свою очередь, проверяет токен аутентификации через компонент безопасности и запрашивает список счетов у основной банковской системы с помощью ее компонента-адаптера (еще одного бина Spring).

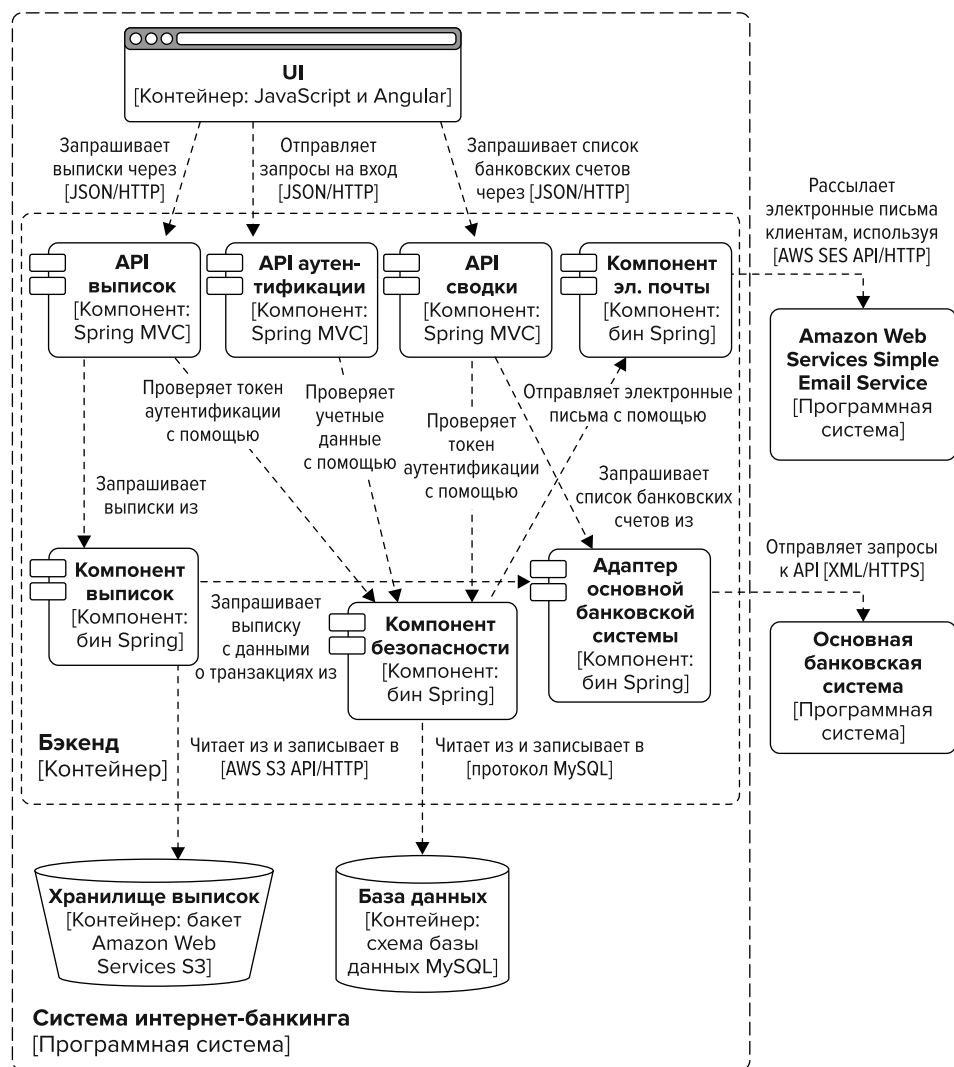


Представление компонентов: система интернет-банкинга, бэкенд

Рис. 5.3. Пример диаграммы компонентов для бэкенда вымышленной системы интернет-банкинга (шаг 3)

Чтобы получить выписку по счету, клиент может воспользоваться UI, в результате чего выполняется вызов API JSON/HTTP к API выписок, также построенному при помощи Spring Web MVC. Когда компонент безопасности подтверждает валидность токена аутентификации, API выписок обращается к компоненту для работы с выписками (бин Spring), который выполняет одно из двух действий. Если выписка есть в кэше,

компонент возвращает ее из хранилища, в противном случае — генерирует новую, запросив данные у основной банковской системы через ее адаптер. Этот дополнительный сценарий показан на рис. 5.4.



Представление компонентов: система интернет-банкинга, бэкенд

Рис. 5.4. Пример диаграммы компонентов для бэкенда вымышленной системы интернет-банкинга (шаг 4)

Вы наверняка заметили, что с ростом числа элементов и связей диаграмма загромождается. Связи начинают пересекаться друг с другом, и чем больше элементов, тем сильнее они переплетаются. Возникает вопрос: можно ли масштабировать этот подход? Если нет, то как изображать на диаграммах системы, которые значительно крупнее и сложнее нашего примера? Этой теме посвящена глава 12, в которой мы рассмотрим применение модели C4 на практике.

На рис. 5.5 показана легенда для диаграммы компонентов. Визуальный стиль наследует стиль диаграмм контекста и контейнеров. Каждый компонент отображается в форме компонента UML.

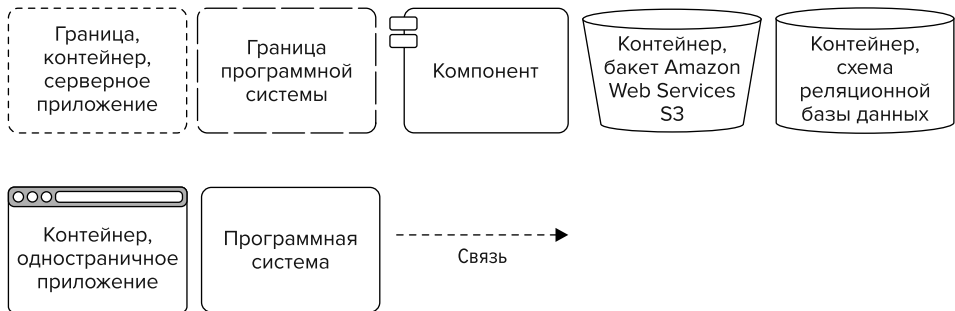


Рис. 5.5. Легенда к примеру диаграммы компонентов

Резюме

В этой главе мы рассмотрели диаграмму компонентов модели C4. Диаграмма показывает компоненты внутри отдельного контейнера, которые, в свою очередь, должны отражать высокоуровневые структуры кода в кодовой базе. Следующая глава станет завершающей в нашем разговоре о статической структуре: в ней мы с помощью диаграммы кода рассмотрим в деталях отдельный компонент.