Федеральное государственное образовательное бюджетное учреждение

высшего образования

«Финансовый УНИВЕРСИТЕТ

при Правительстве Российской Федерации»

(Финансовый университет)

Факультет «Информационных технологий и анализа больших данных»

Реферат

на тему

**«Нотация C4 (Context, Container, Component & Code)»**

Вариант 3

Выполнил студент

группы ПИ20-5

Борисов Н.А.

Москва

2023

Оглавление

[ВВЕДЕНИЕ 3](#_Toc130415932)

[ОСНОВНАЯ ЧАСТЬ 5](#_Toc130415933)

[1.1. Основные компоненты нотации 5](#_Toc130415934)

[1.2. Основные диаграммы нотации 8](#_Toc130415935)

[ЗАКЛЮЧЕНИЕ 14](#_Toc130415936)

[СПИСОК ЛИТЕРАТУРЫ 15](#_Toc130415937)

# ВВЕДЕНИЕ

Нотация C4 (Context, Container, Component & Code) — это нотация, предназначенная для описания архитектуры программного обеспечения (ПО). Нотация C4 состоит из четырех уровней абстракции, каждый из которых используется для представления различных аспектов архитектуры ПО.

Первый уровень абстракции - контексты. Контексты описывают внешние системы, пользователей и другие внешние факторы, которые взаимодействуют с системой. Для описания контекстов используются диаграммы контекста, которые показывают взаимодействие между системой и ее внешними сущностями.

Второй уровень абстракции - контейнеры. Контейнеры — это логические группировки элементов ПО, которые реализуют определенную функциональность. Контейнеры могут быть как физическими серверами, так и виртуальными машинами, но не путать с Docker-контейнерами. Для описания контейнеров используются диаграммы контейнеров, которые показывают взаимодействие между различными контейнерами и их зависимости.

Третий уровень абстракции - компоненты. Компоненты — это отдельные элементы ПО, которые реализуют конкретные функциональные требования. Компоненты могут быть как встроенными в контейнеры, так и независимыми. Для описания компонентов используются диаграммы компонентов, которые показывают взаимодействие между компонентами и их зависимости.

Четвертый уровень абстракции - код. Код — это фактический исходный код компонентов, которые реализуют конкретные функциональные требования. Для описания кода используются диаграммы кода, которые показывают внутреннюю структуру и организацию кода компонентов.

Описание нотации C4 представляет собой подробное объяснение каждого уровня абстракции, а также предоставляет примеры их использования в контексте реальных проектов. Описание включает в себя как теоретические аспекты, так и практические советы по использованию нотации C4 для разработки.

Модель C4 была создана как способ помочь командам разработчиков программного обеспечения описать и передать архитектуру программного обеспечения, как во время предварительных сессий проектирования, так и при ретроспективном документировании существующей кодовой базы. Это способ создания схем вашего кода с различными уровнями детализации.

Хотя модель C4 в первую очередь предназначена для архитекторов и разработчиков программного обеспечения, она предоставляет командам разработчиков программного обеспечения возможность эффективно и действенно описывать свою архитектуру программного обеспечения на разных уровнях детализации, показывая различного уровня модели при диалоге с разными типам аудитории при выполнении предварительного проектирования или ретроспективном документировании существующей кодовой базы.

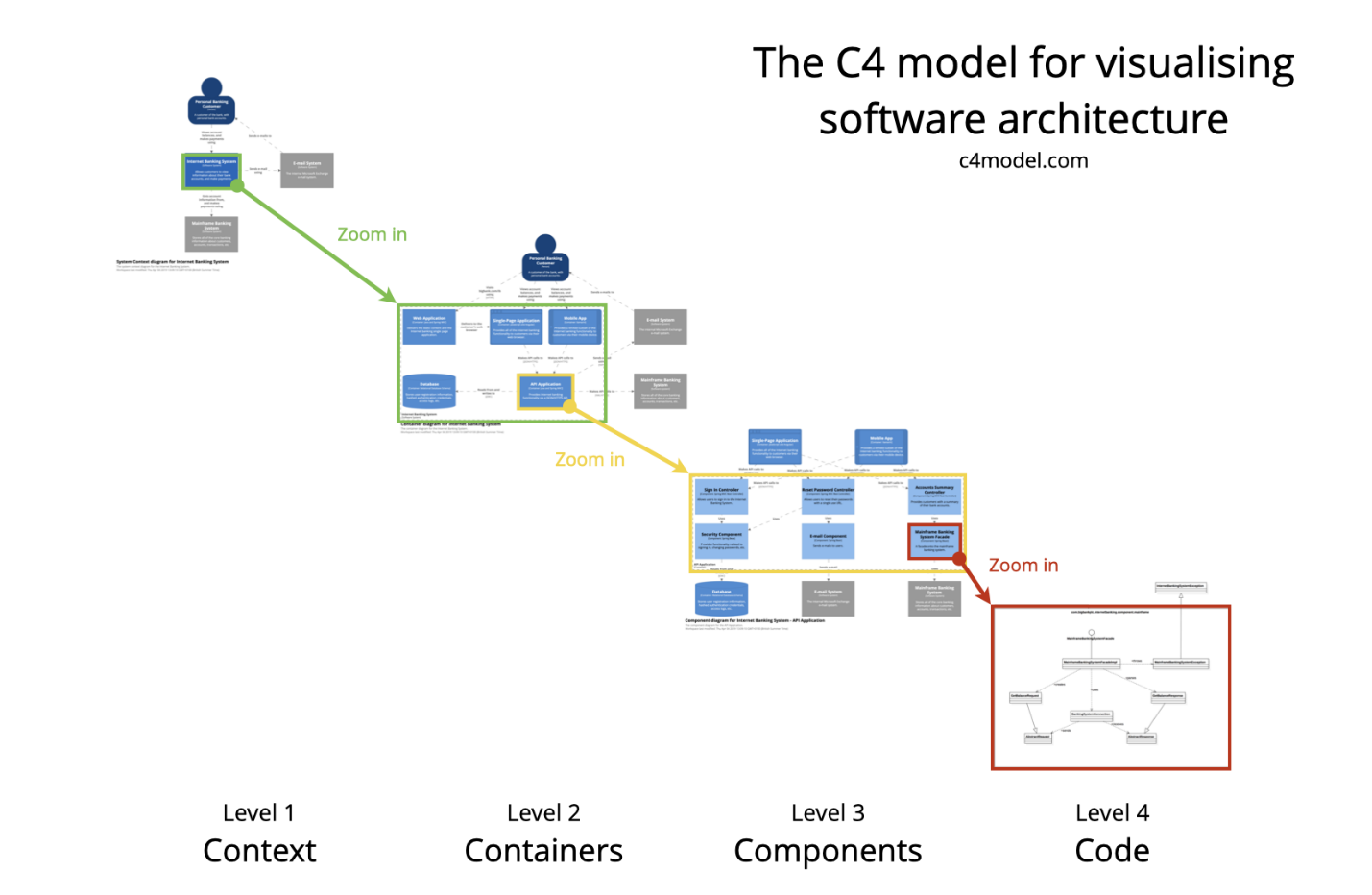


Рисунок 1 – пример модели C4 (нотация C4)

# ОСНОВНАЯ ЧАСТЬ

## Основные компоненты нотации

Модель C4 - это подход "сначала абстракция" к построению диаграмм архитектуры программного обеспечения, основанный на абстракциях, которые отражают то, как архитекторы и разработчики программного обеспечения представляют и строят программное обеспечение. Небольшой набор абстракций и типов диаграмм делает модель C4 простой в освоении и использовании. В частности, не нужно использовать все 4 уровня диаграммы; только те, которые повышают ценность - системный контекст и диаграммы контейнеров достаточны для многих групп разработчиков программного обеспечения.

Применение нотации C4 начинается с описания контекста системы, включая внешние системы и пользователей, которые взаимодействуют с системой.

Модель C4 рассматривает статические структуры программной системы (software system) с точки зрения контейнеров (containers), компонентов (components) и кода (code) и людей (people) использующих программные системы, которые мы создаем.

Программная система состоит из одного или нескольких контейнеров (веб-приложений, мобильных приложений, настольных приложений, баз данных, файловых систем и т.д.), Каждый из которых содержит один или несколько компонентов, которые, в свою очередь, реализуются одним или несколькими элементами кода (например, классы, интерфейсы, объекты, функции и т.д.):

Человек (Person) представляет одного из пользователей вашей программной системы (например, акторов, ролей, персонажей и т.д.).

Программная система (Software System) - это высший уровень абстракции и описывает то, что приносит пользу ее пользователям, независимо от того, являются они людьми или нет. Это включает в себя программную систему, которую вы моделируете, и другие программные системы, от которых зависит ваша программная система (или наоборот). Во многих случаях программная система "принадлежит" одной команде разработчиков программного обеспечения. (структура на рисунке 2)

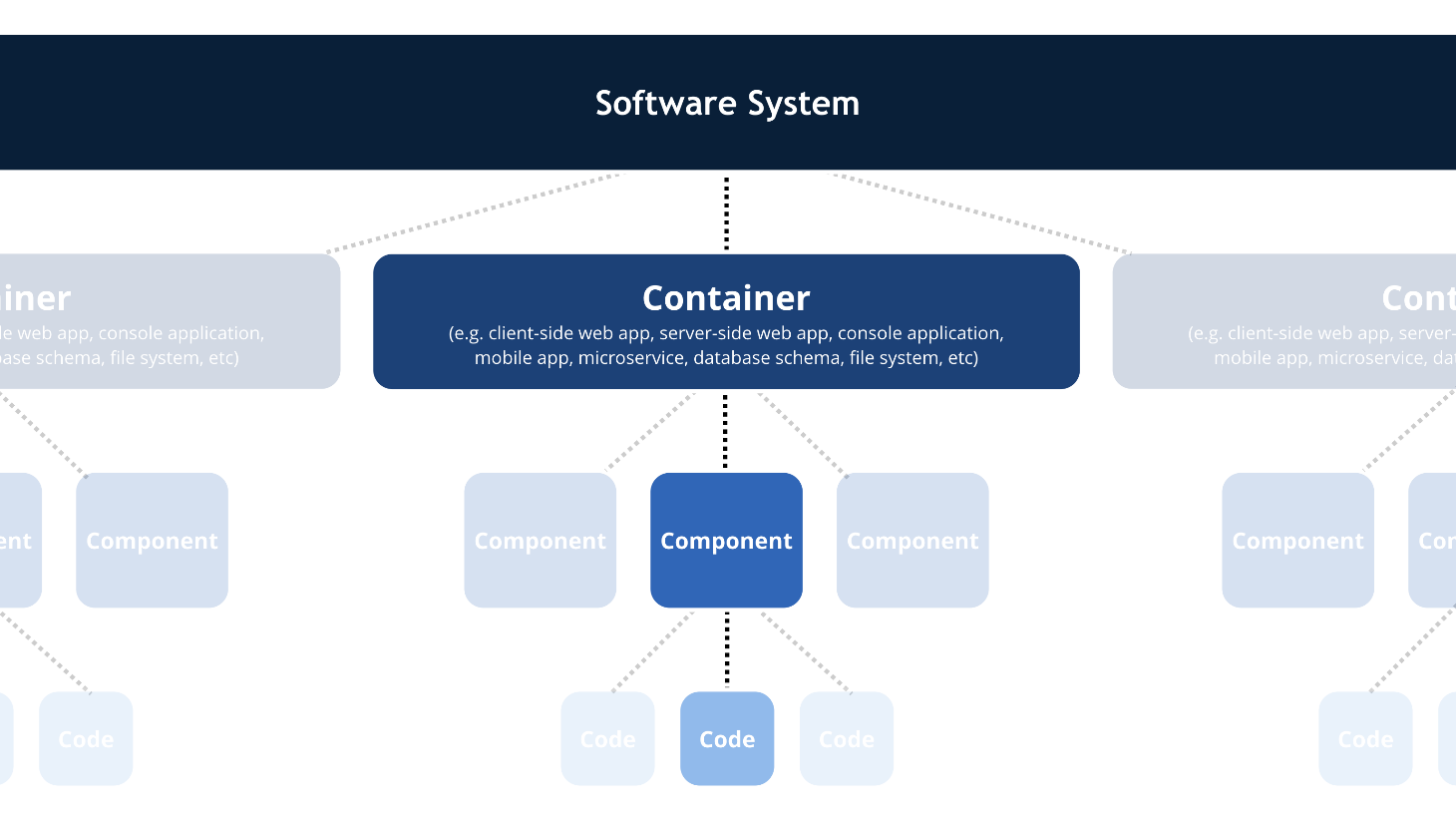


Рисунок 2 – Структура нотации

Контейнер (Container) (приложения и хранилища данных) это не Docker. В модели C4 контейнер представляет приложение или хранилище данных. Контейнер - это то, что должно быть запущено для работы всей программной системы. В реальном выражении контейнер - это что-то вроде:

* Серверное веб-приложение: Веб-приложение Java EE, работающее на Apache Tomcat, ASP.NET Приложение MVC, работающее на Microsoft IIS, приложение Ruby on Rails, работающее на WEBrick, Node.js применение и т.д.
* Веб-приложение на стороне клиента: Приложение JavaScript, работающее в веб-браузере с использованием Angular, Backbone.JS, jQuery и т.д.
* Клиентское настольное приложение: Настольное приложение Windows, написанное с использованием WPF, настольное приложение OS X, написанное с использованием Objective-C, кроссплатформенное настольное приложение, написанное с использованием JavaFX, и т.д.
* Мобильное приложение: Приложение Apple iOS, приложение для Android, приложение Microsoft Windows Phone и т.д.
* Консольное приложение на стороне сервера: Автономное (например, "public static void main") приложение, пакетный процесс и т.д.
* Бессерверная функция: Одна бессерверная функция (например, Amazon Lambda, функция Azure и т.д.).
* База данных: Схема или база данных в системе управления реляционными базами данных, хранилище документов, графическая база данных и т.д., Такие как MySQL, Microsoft SQL Server, база данных Oracle, MongoDB, Riak, Cassandra, Neo4j и т.д.
* Хранилище больших двоичных объектов или контента: Хранилище больших двоичных объектов (например, Amazon S3, хранилище больших двоичных объектов Microsoft Azure и т.д.) Или сеть доставки контента (например, Akamai, Amazon CloudFront и т.д.).
* Файловая система: Полная локальная файловая система или часть более крупной сетевой файловой системы (например, SAN, NAS и т. Д.).
* Сценарий оболочки: Один сценарий оболочки, написанный на Bash и т.д.

Контейнер - это, по сути, контекст или граница, внутри которой выполняется некоторый код или хранятся некоторые данные. И каждый контейнер - это отдельно развертываемая/запускаемая вещь или среда выполнения, обычно (но не всегда) работающая в своем собственном пространстве процессов. Из-за этого связь между контейнерами обычно принимает форму связи между процессами.

Компонент (Component) - является чрезвычайно перегруженным термином в индустрии разработки программного обеспечения, но в данном контексте компонент представляет собой группу связанных функций, инкапсулированных за четко определенным интерфейсом. Если используется такой язык, как Java или C#, самый простой способ представить компонент как набор классов реализации за интерфейсом. Такие аспекты, как упаковка этих компонентов (например, один компонент против множества компонентов в файле JAR, DLL, общей библиотеке и т.д.), Являются отдельной и ортогональной проблемой.

Здесь важно отметить, что все компоненты внутри контейнера обычно выполняются в одном и том же пространстве процессов. В модели C4 компоненты не являются отдельными развертываемыми единицами.

## Основные диаграммы нотации

Визуализация иерархии абстракций выполняется путем создания коллекции диаграмм контекста (Context), контейнеров (Container), компонент (Component) и (необязательно) кода (Code) (например, класса UML). Именно отсюда модель C4 получила свое название.

Уровень 1: Схема системного контекста

Схема системного контекста является хорошей отправной точкой для построения диаграмм и документирования программной системы, позволяя посмотреть со стороны и увидеть общую картину. Изображается схема, показывающая систему в виде прямоугольника в центре, окруженного её пользователями и другими системами, с которыми она взаимодействует.

Детали здесь не важны, так как это вид, показывающий общую картину системного ландшафта. Основное внимание уделяется людям (актерам, ролям, персонажам и т.д.) и программным системам, а не технологиям, протоколам и другим деталям низкого уровня. Это своего рода схема, которую могли бы показать нетехническим людям.

* Область применения: Единая программная система.
* Основные элементы: Программная система в объеме.
* Вспомогательные элементы: Люди (например, пользователи, действующие лица, роли или персонажи) и программные системы (внешние зависимости), которые непосредственно связаны с программной системой в области действия. Как правило, эти другие программные системы выходят за рамки или границы собственной программной системы, и собственник (разработчик) модели не имеет никакой за них ответственности или не является их владельцем.
* Целевая аудитория: все, как технические, так и нетехнические люди, внутри и за пределами команды разработчиков программного обеспечения.
* Рекомендуется для большинства команд.

Пример диаграммы на рисунке 3.

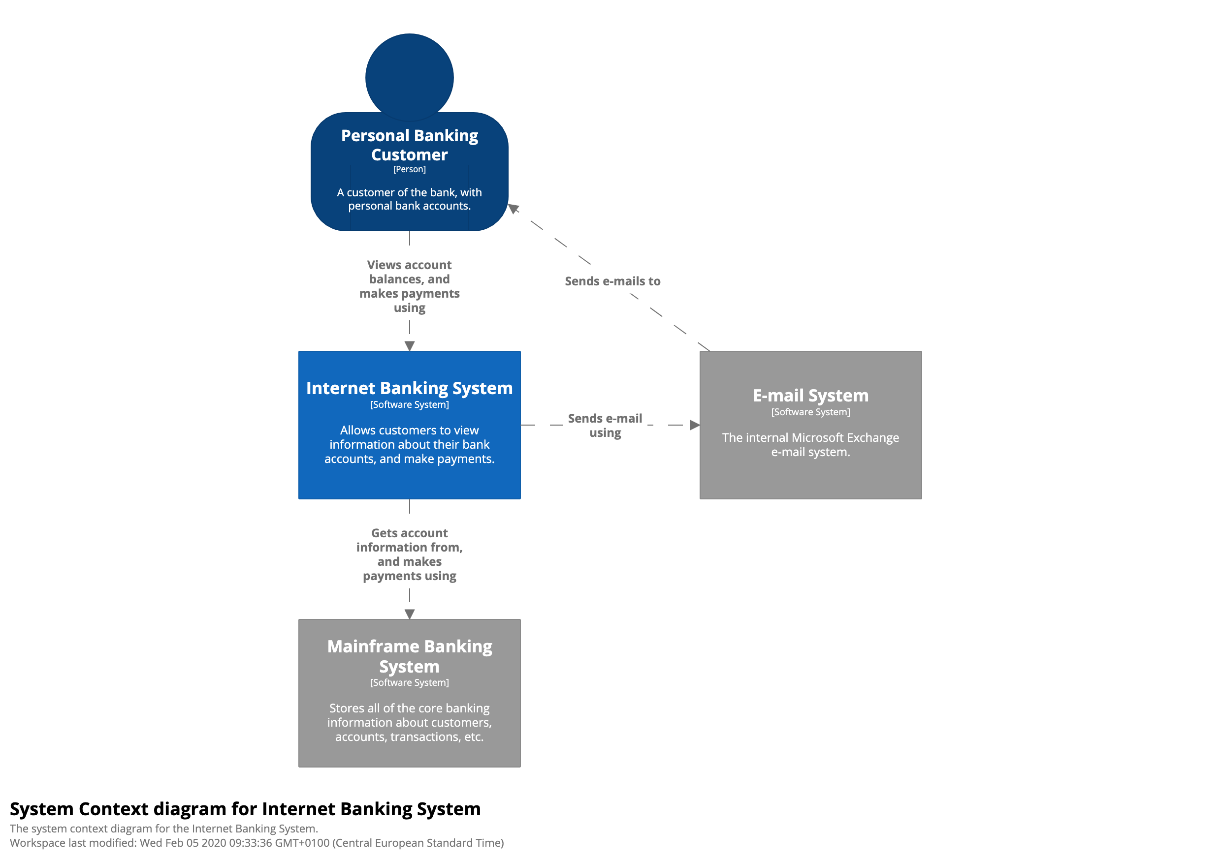


Рисунок 3 – Диаграмма уровня контекста

Уровень 2: Схема контейнеров

Как только описывается представление того, как система вписывается в общую ИТ-среду, действительно полезным следующим шагом будет уточнение границ системы с помощью диаграммы контейнеров.

Схема контейнера показывает высокоуровневую форму архитектуры программного обеспечения и то, как распределяются обязанности между её узлами. Она также показывает основные технологические решения и то, как контейнеры взаимодействуют друг с другом. Это простая, ориентированная на технологии диаграмма высокого уровня, которая полезна как разработчикам программного обеспечения, так и персоналу службы поддержки/эксплуатации. Пример диаграммы на рисунке 4.

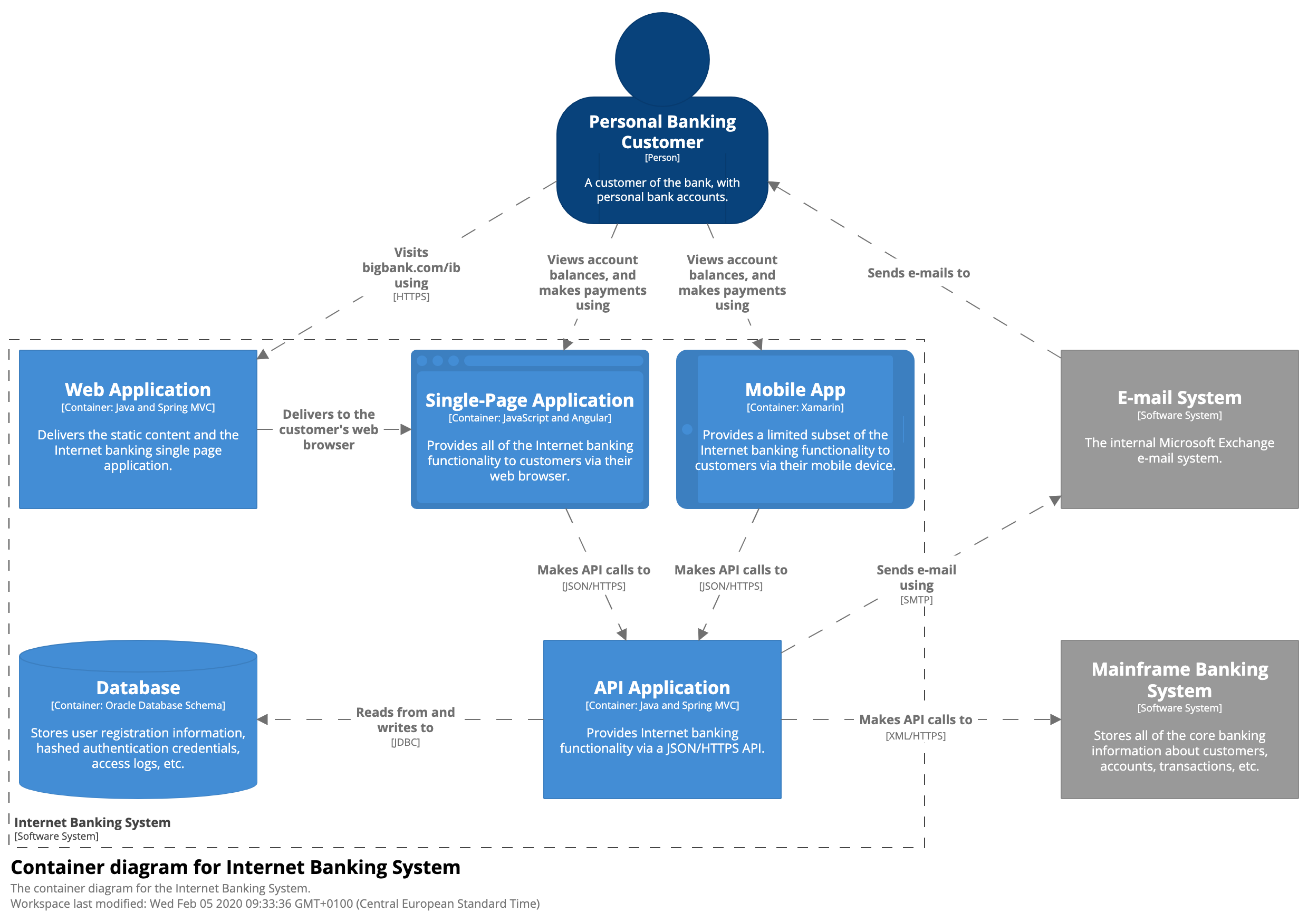


Рисунок 4 – Диаграмма уровня контейнеров

* Область применения: Единая программная система.
* Основные элементы: Контейнеры в пределах области действия программной системы.
* Вспомогательные элементы: Люди и программные системы, непосредственно подключенные к контейнерам.
* Целевая аудитория: Технические специалисты внутри и за пределами команды разработчиков программного обеспечения; в том числе архитекторы программного обеспечения, разработчики и операционный/вспомогательный персонал.
* Рекомендуется для большинства команд.
* Примечания: На этой диаграмме ничего не говорится о сценариях развертывания, кластеризации, репликации, отработке отказа и т.д.

Уровень 3: Схема компонентов

Затем можно увеличить и декомпозировать каждый контейнер дальше, чтобы определить основные структурные строительные блоки и их взаимодействие.

Диаграмма компонентов показывает, как контейнер состоит из нескольких "компонентов", каковы каждый из этих компонентов, их обязанности и детали технологии/реализации. Пример на рисунке 5.

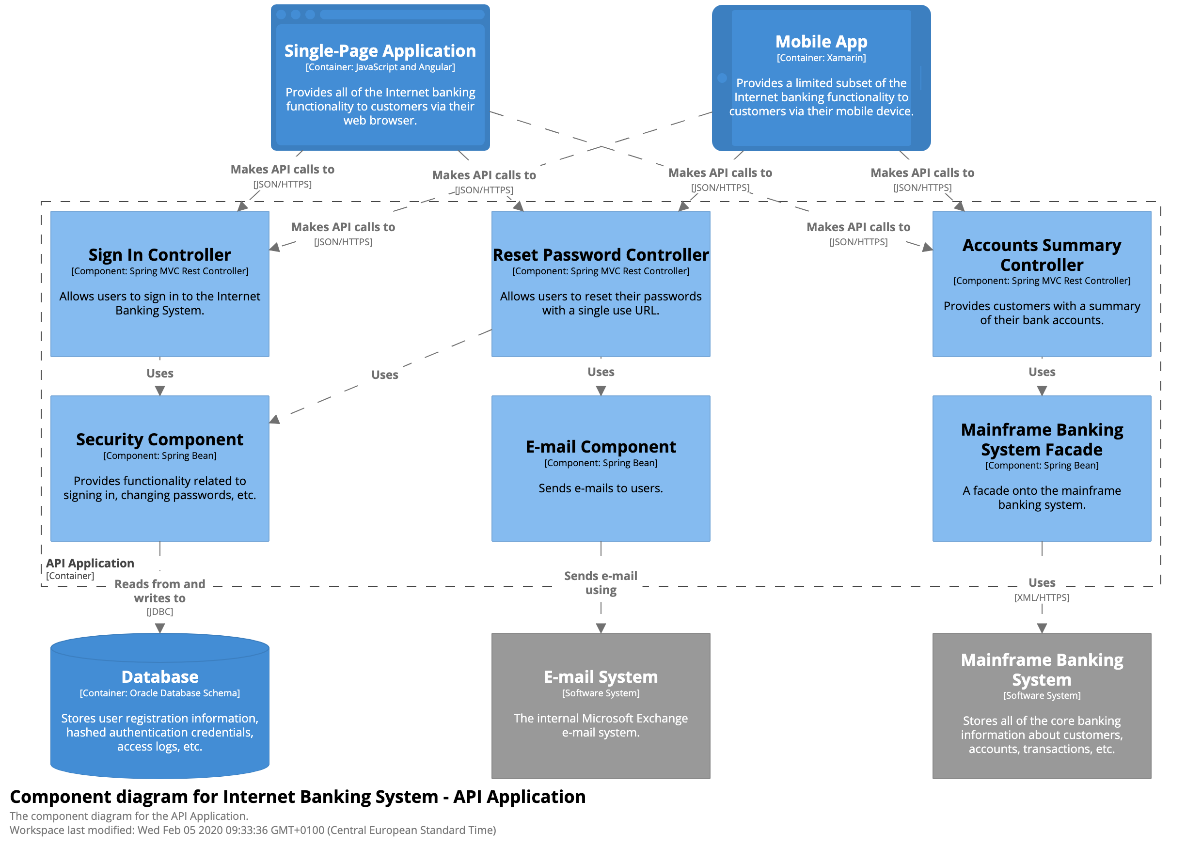


Рисунок 5 – Диаграмма компонентов

Область применения: Один контейнер.

Основные элементы: Компоненты внутри контейнера в области действия.

Вспомогательные элементы: Контейнеры (в пределах области применения программной системы) плюс люди и программные системы, непосредственно подключенные к компонентам.

Целевая аудитория: Архитекторы и разработчики программного обеспечения.

Не рекомендуется для большинства команд. Рекомендуется создавать диаграммы компонентов только в том случае, если они повышают ценность, и необходимо рассмотреть возможность автоматизации их создания для долговременной документации.

Уровень 4: Код

Наконец, возможно увеличить масштаб каждого компонента, чтобы показать, как он реализован в виде кода; используя диаграммы классов UML, диаграммы отношений сущностей или аналогичные.

Это необязательный уровень детализации, который часто доступен по требованию с помощью таких инструментов, как IDE. В идеале эта диаграмма должна быть автоматически сгенерирована с помощью инструментов (например, IDE или инструмента моделирования UML), и следует рассмотреть возможность отображения только тех атрибутов и методов, которые позволяют рассказать то, что необходимо рассказать целевой аудитории. Этот уровень детализации не рекомендуется ни для чего, кроме наиболее важных или сложных компонентов.

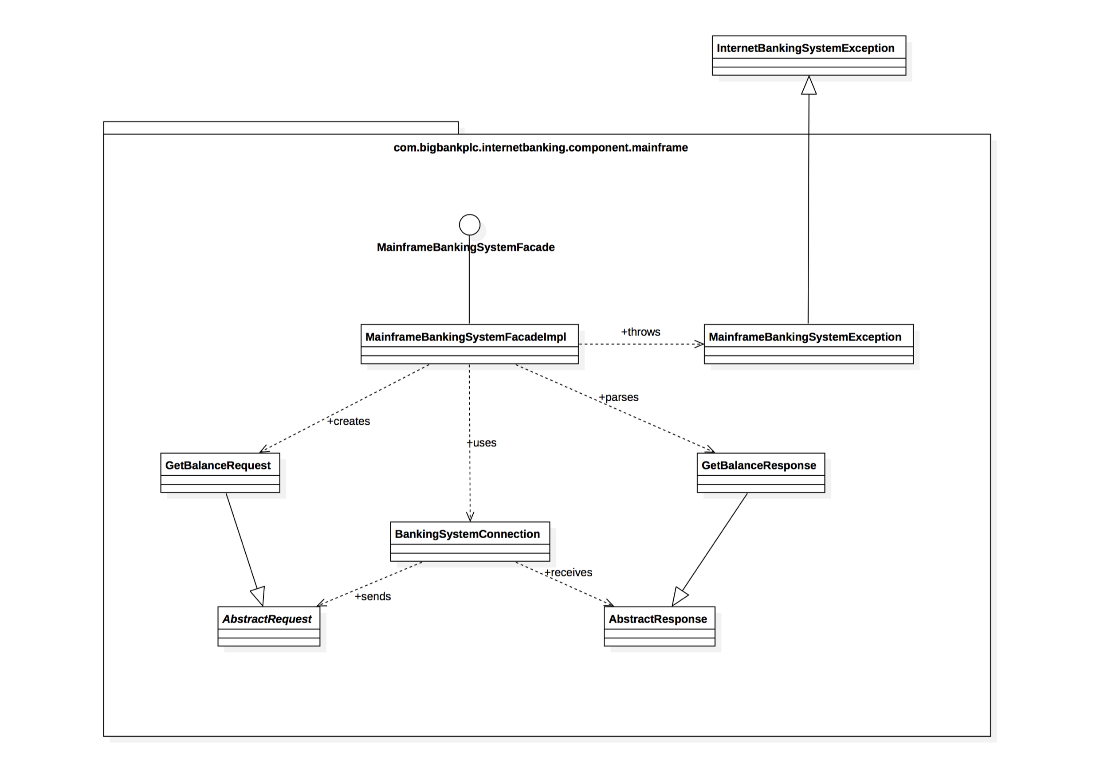


Рисунок 6 – Диаграмма кода

Сфера применения: Предприятие.

Основные элементы: Люди и программные системы, относящиеся к предприятию по сфере деятельности.

Целевая аудитория: Технические и нетехнические специалисты, как внутри, так и за пределами команды разработчиков программного обеспечения.

Существуют также и другие диаграммы в рамках данной нотации.

Динамическая диаграмма может быть полезна, если необходимо показать, как элементы статической модели взаимодействуют во время выполнения для реализации истории пользователя, варианта использования, функции и т.д. Эта динамическая диаграмма основана на диаграмме связей UML (communication diagram), ранее известной как "диаграмма совместной работы UML (collaboration diagram). Это похоже на диаграмму последовательности UML (sequence diagram), хотя она позволяет расположить элементы диаграммы в свободной форме с пронумерованными взаимодействиями для указания порядка.

Схема развертывания позволяет проиллюстрировать, как программные системы и/или контейнеры в статической модели сопоставляются с инфраструктурой. Эта схема развертывания основана на схеме развертывания UML, хотя и немного упрощена, чтобы показать сопоставление между контейнерами и узлами развертывания. Узел развертывания - это что-то вроде физической инфраструктуры (например, физический сервер или устройство), виртуализированной инфраструктуры (например, IaaS, PaaS, виртуальная машина), контейнерной инфраструктуры (например, контейнер Docker), среды выполнения (например, сервер базы данных, веб-сервер/сервер приложений Java EE, Microsoft IIS) и т.д. Узлы развертывания могут быть вложенными.

Возможно также можете включить узлы инфраструктуры, такие как службы DNS, балансировщики нагрузки, брандмауэры и т.д.

# ЗАКЛЮЧЕНИЕ

Можно сделать вывод, что нотация C4(Context, Container, Component & Code) является полезным инструментом для описания архитектуры программного обеспечения. Нотация C4 позволяет описывать архитектуру программного обеспечения на различных уровнях абстракции, начиная от контекста и заканчивая кодом компонентов.

Основным преимуществом нотации C4 является ее простота использования. Нотация C4 является понятной для понимания и легкой в использовании даже для неопытных пользователей. Кроме того, использование нотации C4 обеспечивает улучшение коммуникации между различными участниками проекта, так как нотация C4 предоставляет структурированный и понятный подход для описания архитектуры. Нотация C4 также помогает быстро обнаруживать ошибки и несоответствия в архитектуре, что может привести к более высокому качеству и эффективности разработки ПО.

Однако, следует учитывать, что нотация C4 не может быть использована для описания всех типов архитектур, так как она ориентирована на описание контекста, контейнеров, компонентов и кода. В больших проектах, где имеется много компонентов, описание архитектуры с помощью нотации C4 может стать сложным и трудоемким процессом. Кроме того, нотация C4 не имеет формальной стандартизации, что может привести к различным интерпретациям и затруднить понимание архитектуры. Нотация C4 также не является удобной для описания динамических аспектов архитектуры, таких как последовательность вызовов или изменения состояния.

# СПИСОК ЛИТЕРАТУРЫ

1. Simon Brown. "The C4 model for visualising software architecture"
2. Simon Brown. "Software Architecture for Developers"
3. Eoin Woods, Nick Rozanski. "Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives"
4. Gernot Starke, Klaus Schmid. "Documenting Software Architectures: Views and Beyond"
5. Len Bass, Paul Clements, Rick Kazman. "Software Architecture in Practice"
6. Mark Richards. "Fundamentals of Software Architecture"