C4Builder

Overview

hello-C4

Introduction

This project was created using c4builder

Take a look at

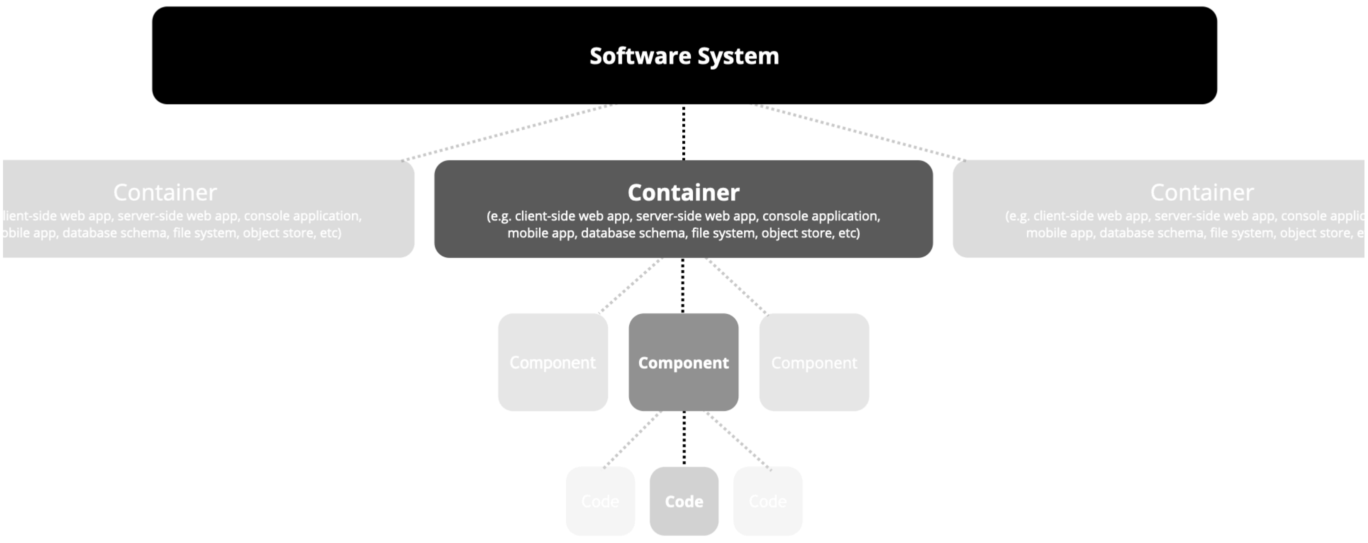
- PlantUml creates diagrams from plain text.
- Markdown creates rich text documents from plant text.
- C4Model the idea behind maps of your code
- C4-PlantUML C4 syntax support for generating plantuml diagrams
- vscode-plantuml plugin for visual studio code to view diagrams at design time

Open the terminal and run the following commands to start compiling the documentation

```
npm i -g c4builder
c4builder
```

> Note on using local images inside markdown files
>
> Images should be placed next to the markdown file using them.
>
> All of them will be copied over to the `docs` folder either in `/` (in the case of a single MD/PDF file) or following the same folder structure as in `src`, so make sure they have unique names.

Abstractions used


Software System
Container
(e.g. client-side web app, server-side web app, console application, mobile app, database schema, file system, object store, etc)
Component
Code

A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).

Person

However you think about your users (as actors, roles, personas, etc), people are the various human users of your software system.

Software System

A software system is the highest level of abstraction and describes something that delivers value to its users, whether they are human or not. This includes the software system you are modelling, and the other software systems upon which your software system depends (or vice versa).

Container

A container represents something that hosts code or data. A container is something that needs to be running in order for the overall software system to work. In real terms, a container is something like:

- Server-side web application: A Java EE web application running on Apache Tomcat, an ASP.NET MVC application running on Microsoft IIS, a Ruby on Rails application running on WEBrick, a Node.js application, etc.
- Client-side web application: A JavaScript application running in a web browser using Angular, Backbone.JS, jQuery, etc).
- Client-side desktop application: A Windows desktop application written using WPF, an OS X desktop application written using Objective-C, a cross-platform desktop application written using JavaFX, etc.
- Mobile app: An Apple iOS app, an Android app, a Microsoft Windows Phone app, etc.

- Server-side console application: A standalone (e.g. "public static void main")
- etc

Component

Component The word "component" is a hugely overloaded term in the software development industry, but in this context a component is simply a grouping of related functionality encapsulated behind a well-defined interface. If you're using a language like Java or C#, the simplest way to think of a component is that it's a collection of implementation classes behind an interface. Aspects such as how those components are packaged (e.g. one component vs many components per JAR file, DLL, shared library, etc) is a separate and orthogonal concern.

An important point to note here is that all components inside a container typically execute in the same process space.

Introduction

This project was created using c4builder

Take a look at

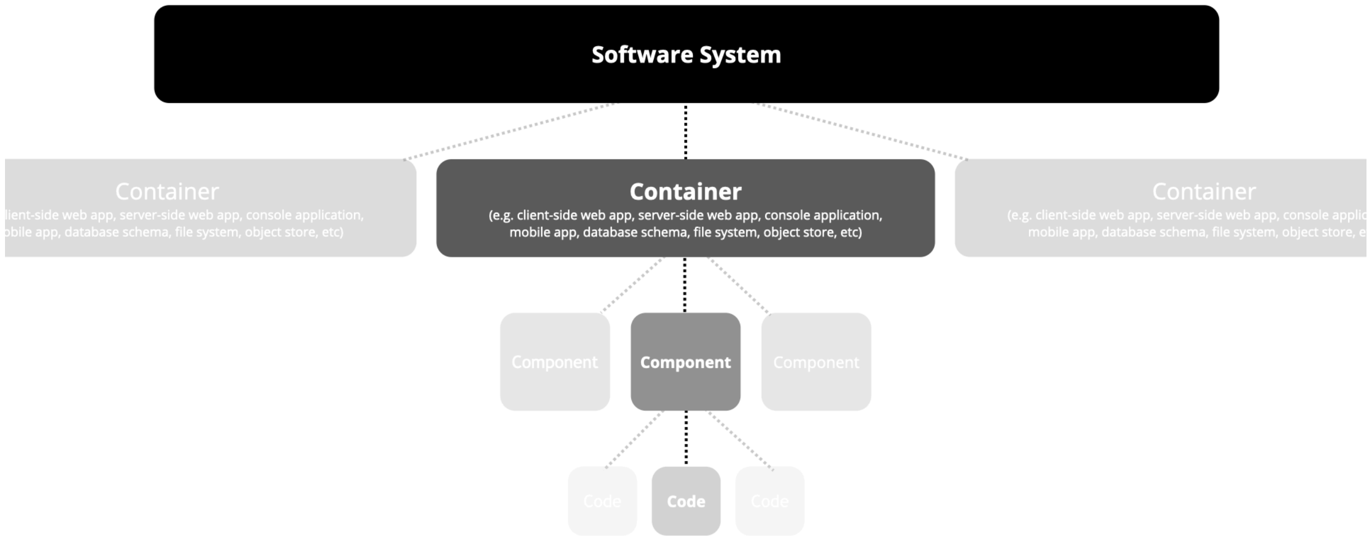
- PlantUml creates diagrams from plain text.
- Markdown creates rich text documents from plant text.
- C4Model the idea behind maps of your code
- C4-PlantUML C4 syntax support for generating plantuml diagrams
- vscode-plantuml plugin for visual studio code to view diagrams at design time

Open the terminal and run the following commands to start compiling the documentation

```
npm i -g c4builder
c4builder
```

> Note on using local images inside markdown files
>
> Images should be placed next to the markdown file using them.
>
> All of them will be copied over to the `docs` folder either in `/` (in the case of a single MD/PDF file) or following the same folder structure as in `src`, so make sure they have unique names.

Abstractions used


Software System
Container
(e.g. client-side web app, server-side web app, console application, mobile app, database schema, file system, object store, etc)
Component
Code

A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).

Person

However you think about your users (as actors, roles, personas, etc), people are the various human users of your software system.

Software System

A software system is the highest level of abstraction and describes something that delivers value to its users, whether they are human or not. This includes the software system you are modelling, and the other software systems upon which your software system depends (or vice versa).

Container

A container represents something that hosts code or data. A container is something that needs to be running in order for the overall software system to work. In real terms, a container is something like:

- Server-side web application: A Java EE web application running on Apache Tomcat, an ASP.NET MVC application running on Microsoft IIS, a Ruby on Rails application running on WEBrick, a Node.js application, etc.
- Client-side web application: A JavaScript application running in a web browser using Angular, Backbone.JS, jQuery, etc).
- Client-side desktop application: A Windows desktop application written using WPF, an OS X desktop application written using Objective-C, a cross-platform desktop application written using JavaFX, etc.
- Mobile app: An Apple iOS app, an Android app, a Microsoft Windows Phone app, etc.
- Server-side console application: A standalone (e.g. "public static void main")
- etc

Component

Component The word "component" is a hugely overloaded term in the software development industry, but in this context a component is simply a grouping of related functionality encapsulated behind a well-defined interface. If you're using a language like Java or C#, the simplest way to think of a component is that it's a collection of implementation classes behind an interface. Aspects such as how those components are packaged (e.g. one component vs many components per JAR file, DLL, shared library, etc) is a separate and orthogonal concern.

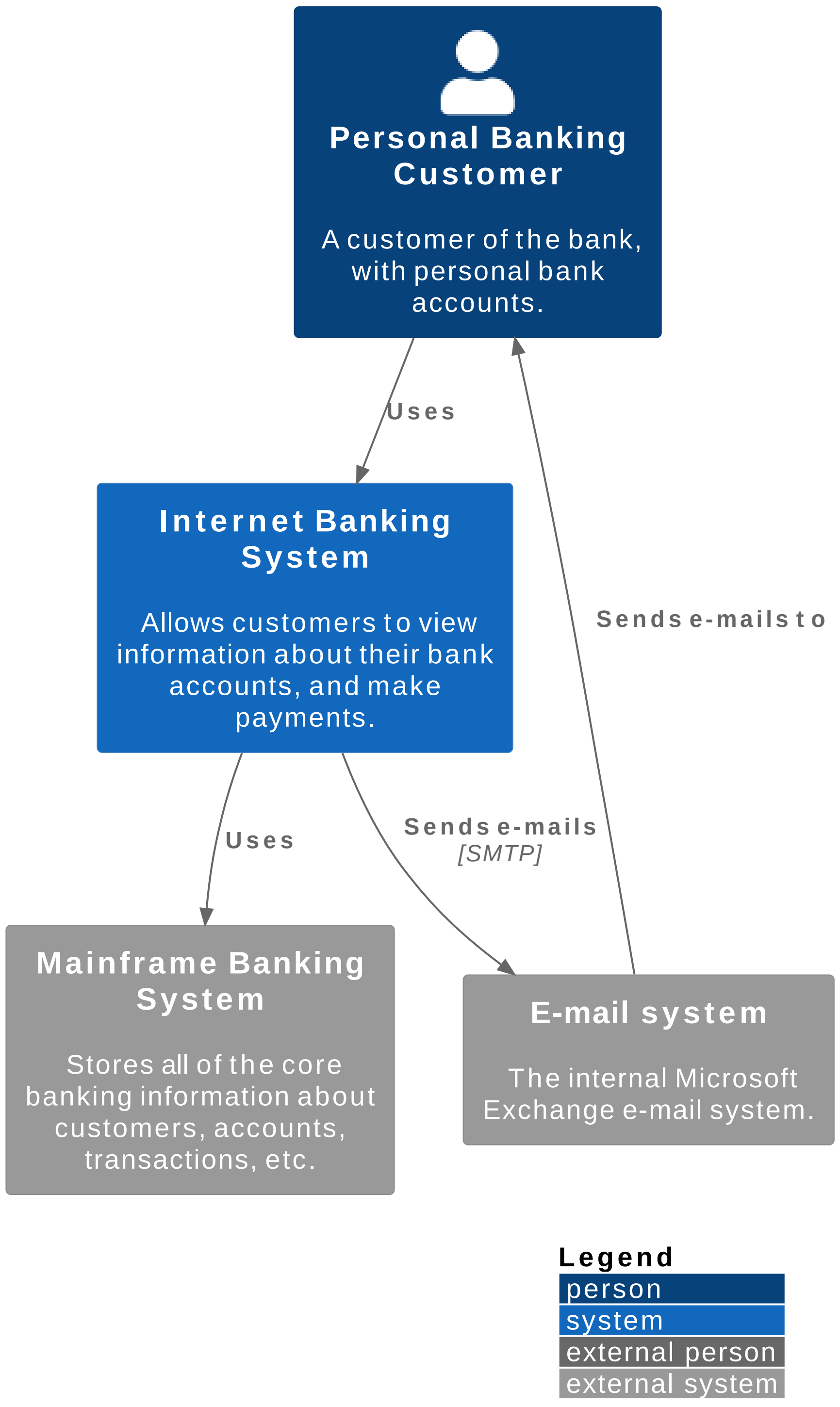
An important point to note here is that all components inside a container typically execute in the same process space.

.vscode

`/.vscode`

src

`/src`


Personal Banking Customer
A customer of the bank, with personal bank accounts.
Uses
Sends e-mails to
Internet Banking System
Allows customers to view information about their bank accounts, and make payments.
Uses
Sends e-mails
[SMTP]
Mainframe Banking System
Stores all of the core banking information about customers, accounts, transactions, etc.
E-mail system
The internal Microsoft Exchange e-mail system.
Legend
person
system
external person
external system

Level 1: System Context diagram

A System Context diagram is a good starting point for diagramming and documenting a software system, allowing you to step back and see the big picture. Draw a diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with.

Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.

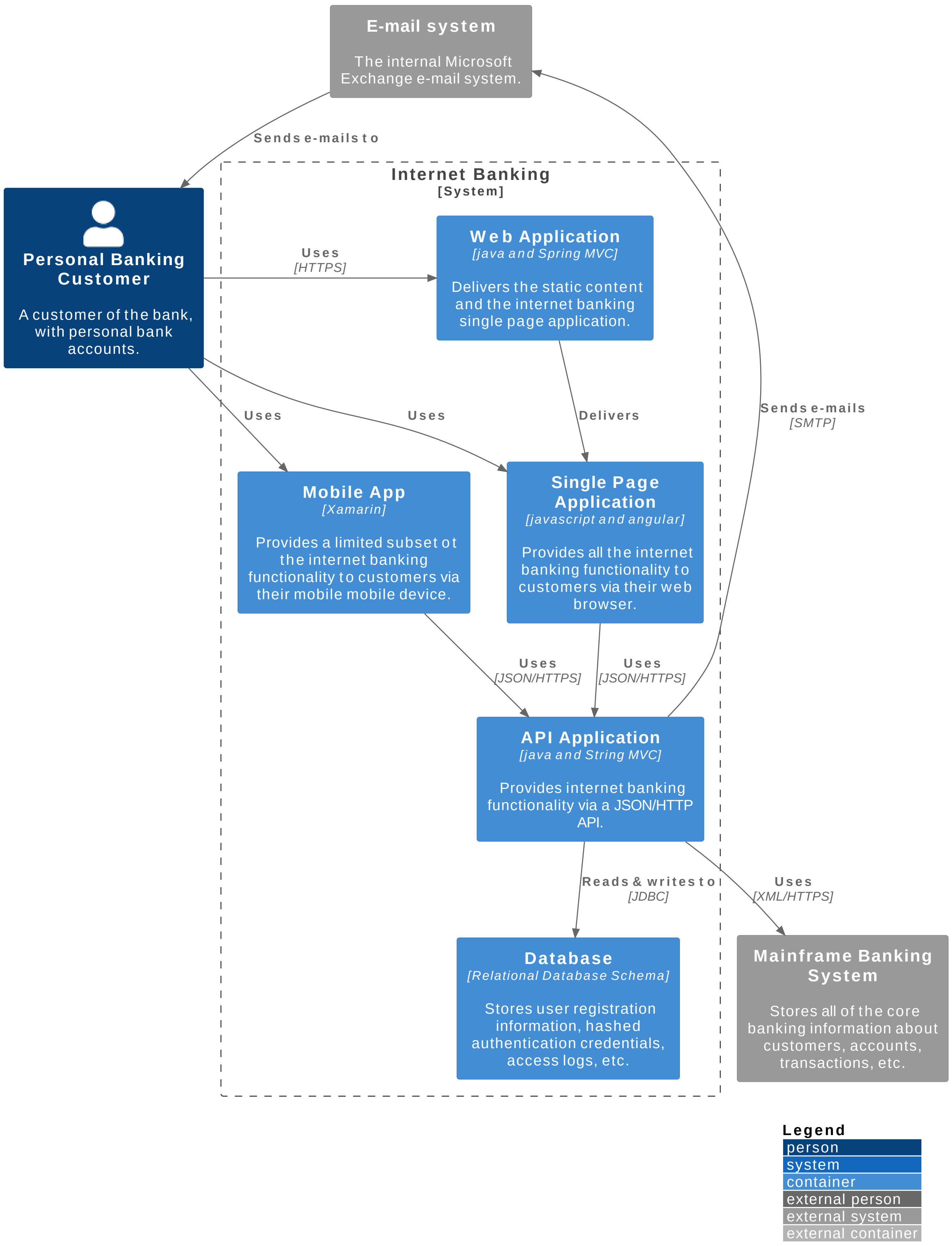
Scope: A single software system.

Primary elements: The software system in scope. Supporting elements: People (e.g. users, actors, roles, or personas) and software systems (external dependencies) that are directly connected to the software system in scope. Typically these other software systems sit outside the scope or boundary of your own software system, and you don’t have responsibility or ownership of them.

Intended audience: Everybody, both technical and non-technical people, inside and outside of the software development team.

1 Internet Banking System

`/src/1 Internet Banking System`


E-mail system
The internal Microsoft Exchange e-mail system.
Sends e-mails to
Internet Banking
[System]
Personal Banking Customer
A customer of the bank, with personal bank accounts.
Uses
[HTTPS]
Web Application
[java and Spring MVC]
Delivers the static content and the internet banking single page application.
Uses
Uses
Delivers
Sends e-mails
[SMTP]
Mobile App
[Xamarin]
Provides a limited subset ot the internet banking functionality to customers via their mobile mobile device.
Single Page Application
[javascript and angular]
Provides all the internet banking functionality to customers via their web browser.
Uses
[JSON/HTTPS]
Uses
[JSON/HTTPS]
API Application
[java and String MVC]
Provides internet banking functionality via a JSON/HTTP API.
Reads & writes to
[JDBC]
Uses
[XML/HTTPS]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Mainframe Banking System
Stores all of the core banking information about customers, accounts, transactions, etc.
Legend
person
system
container
external person
external system
external container

Level 2: Container diagram

Once you understand how your system fits in to the overall IT environment, a really useful next step is to zoom-in to the system boundary with a Container diagram. A "container" is something like a server-side web application, single-page application, desktop application, mobile app, database schema, file system, etc.

Essentially, a container is a separately runnable/deployable unit (e.g. a separate process space) that executes code or stores data.

The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focussed diagram that is useful for software developers and support/operations staff alike.

Scope: A single software system.

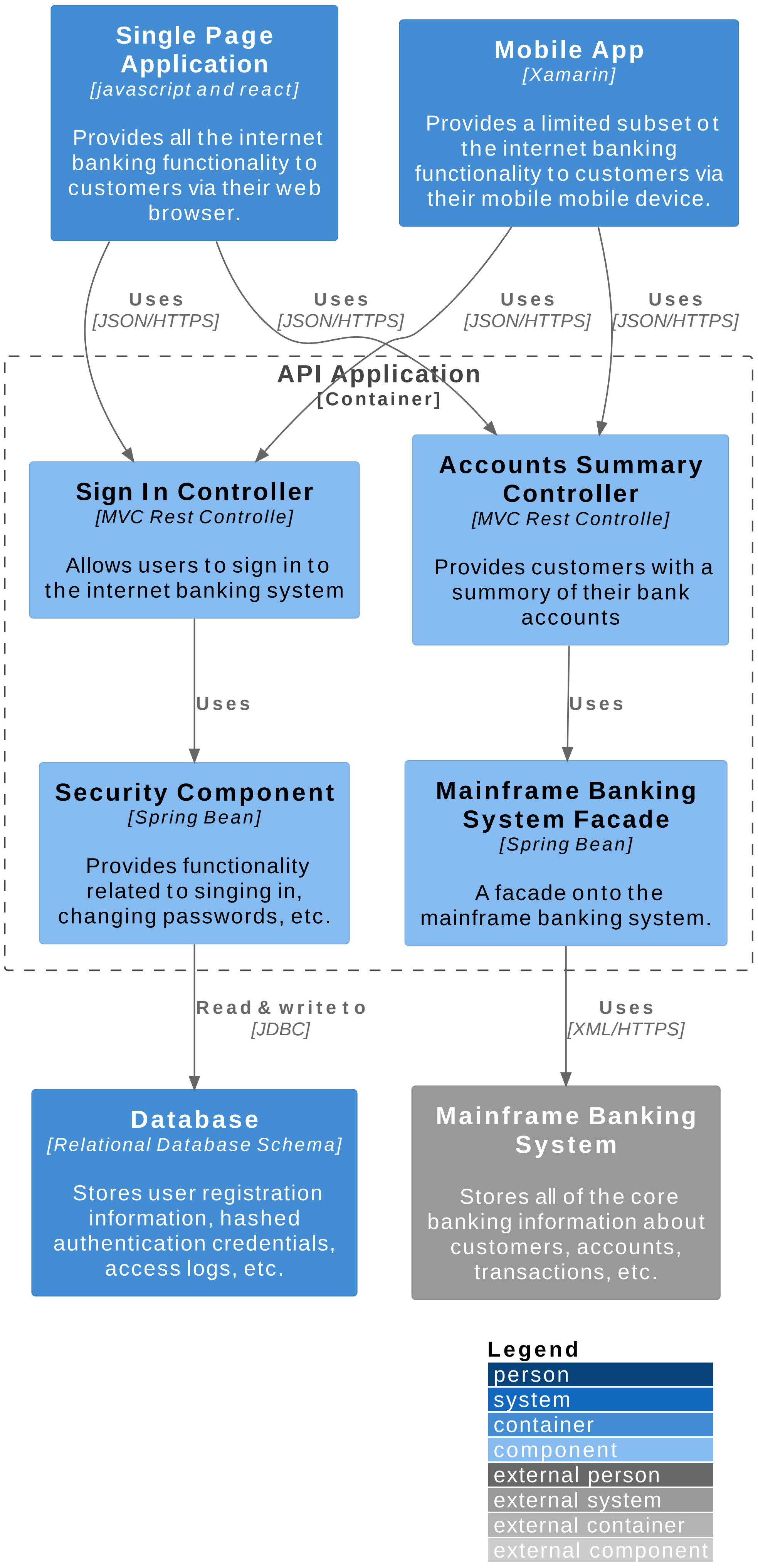
Primary elements: Containers within the software system in scope. Supporting elements: People and software systems directly connected to the containers.

Intended audience: Technical people inside and outside of the software development team; including software architects, developers and operations/support staff.

Notes: This diagram says nothing about deployment scenarios, clustering, replication, failover, etc.

API Application

`/src/1 Internet Banking System/API Application`


Single Page Application
[javascript and react]
Provides all the internet banking functionality to customers via their web browser.
Mobile App
[Xamarin]
Provides a limited subset ot the internet banking functionality to customers via their mobile mobile device.
Uses
[JSON/HTTPS]
Uses
[JSON/HTTPS]
Uses
[JSON/HTTPS]
Uses
[JSON/HTTPS]
API Application
[Container]
Sign In Controller
[MVC Rest Controlle]
Allows users to sign in to the internet banking system
Accounts Summary Controller
[MVC Rest Controlle]
Provides customers with a summory of their bank accounts
Uses
Uses
Security Component
[Spring Bean]
Provides functionality related to singing in, changing passwords, etc.
Mainframe Banking System Facade
[Spring Bean]
A facade onto the mainframe banking system.
Read & write to
[JDBC]
Uses
[XML/HTTPS]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Mainframe Banking System
Stores all of the core banking information about customers, accounts, transactions, etc.
Legend
person
system
container
component
external person
external system
external container
external component

1. Record architecture decisions

Date: 2020-06-05

Status

Accepted

Context

We need to record the architectural decisions made on this project.

Decision

We will use Architecture Decision Records, as described by Michael Nygard in this article: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

Consequences

See Michael Nygard's article, linked above.

Level 3: Component diagram

Next you can zoom in and decompose each container further to identify the major structural building blocks and their interactions.

The Component diagram shows how a container is made up of a number of "components", what each of those components are, their responsibilities and the technology/implementation details.

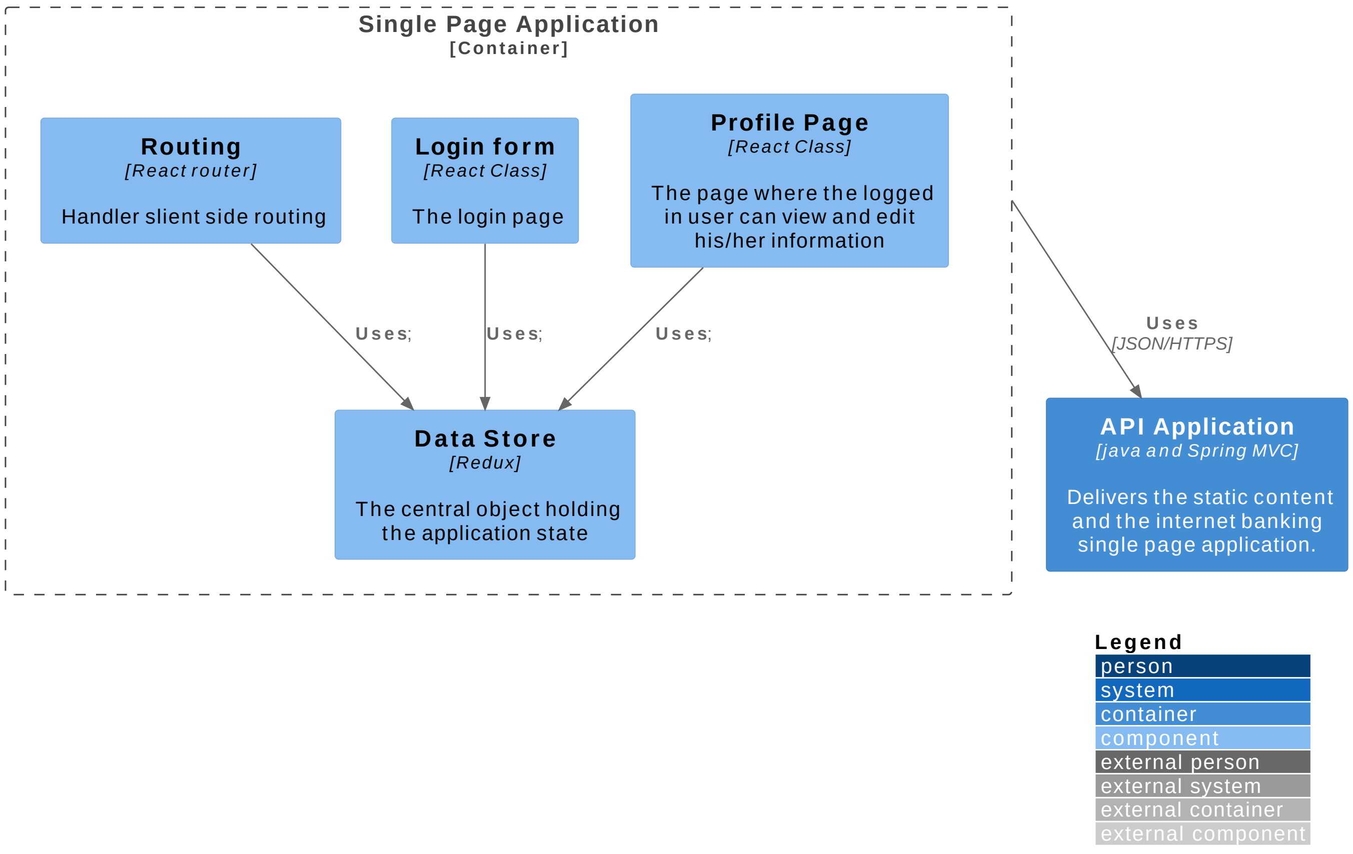
Scope: A single container.

Primary elements: Components within the container in scope. Supporting elements: Containers (within the software system in scope) plus people and software systems directly connected to the components.

Intended audience: Software architects and developers.

Single Page Application

`/src/1 Internet Banking System/Single Page Application`



Level 3: Component diagram

Next you can zoom in and decompose each container further to identify the major structural building blocks and their interactions.

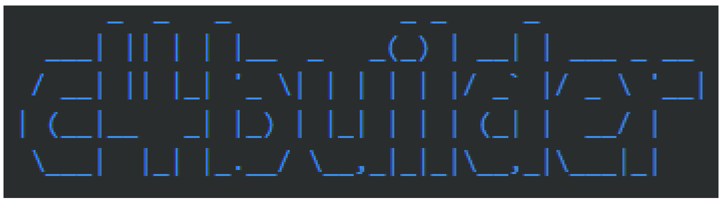
The Component diagram shows how a container is made up of a number of "components", what each of those components are, their responsibilities and the technology/implementation details.

Scope: A single container.

Primary elements: Components within the container in scope. Supporting elements: Containers (within the software system in scope) plus people and software systems directly connected to the components.

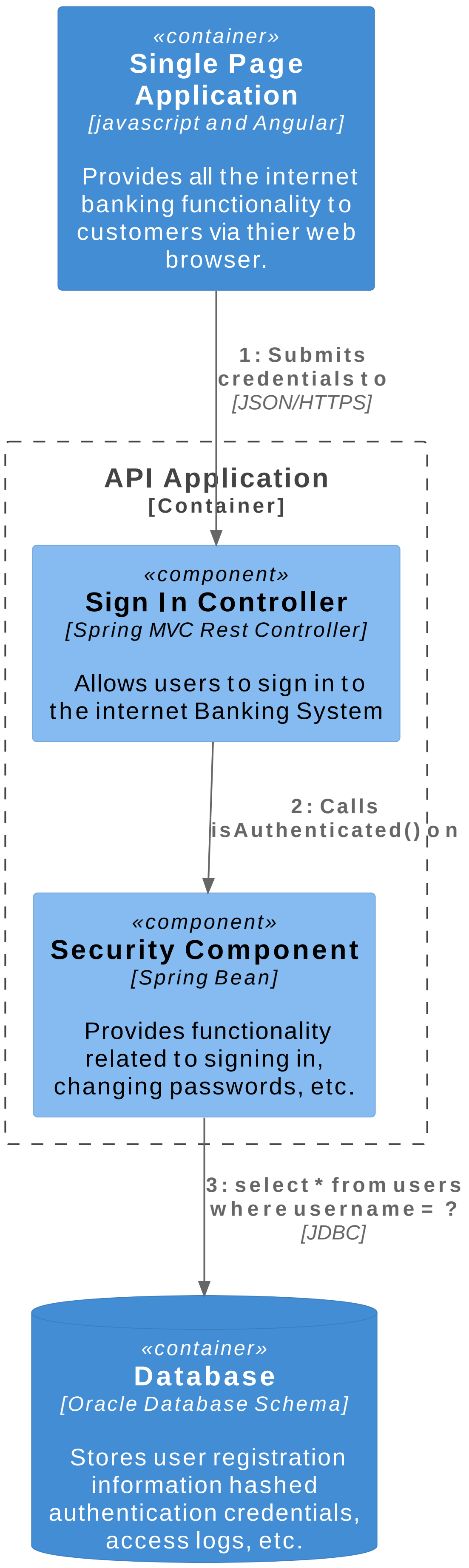
Intended audience: Software architects and developers.

> Example of included local image



Dynamic Diagram

`/src/1 Internet Banking System/Single Page Application/Dynamic Diagram`


«container»
Single Page Application
[javascript and Angular]
Provides all the internet banking functionality to customers via thier web browser.
1: Submits credentials to
[JSON/HTTPS]
API Application
[Container]
«component»
Sign In Controller
[Spring MVC Rest Controller]
Allows users to sign in to the internet Banking System
2: Calls isAuthenticated() on
«component»
Security Component
[Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
3: select * from users where username = ?
[JDBC]
«container»
Database
[Oracle Database Schema]
Stores user registration information hashed authentication credentials, access logs, etc.

Dynamic diagram

A simple dynamic diagram can be useful when you want to show how elements in a static model collaborate at runtime to implement a user story, use case, feature, etc. This dynamic diagram is based upon a UML communication diagram (previously known as a "UML collaboration diagram"). It is similar to a UML sequence diagram although it allows a free-form arrangement of diagram elements with numbered interactions to indicate ordering.

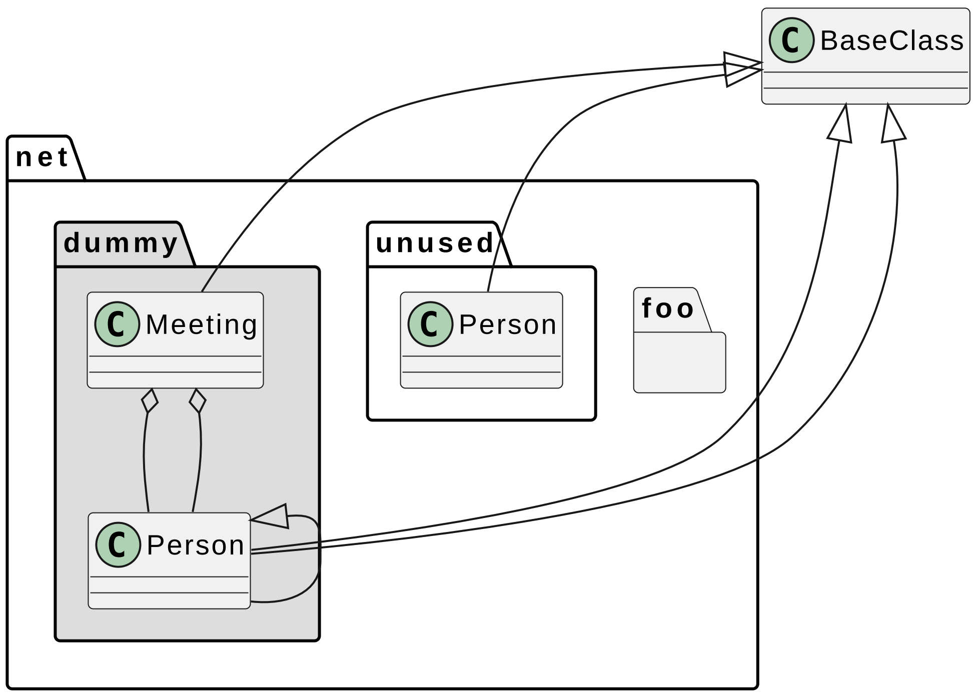
Scope: An enterprise, software system or container.

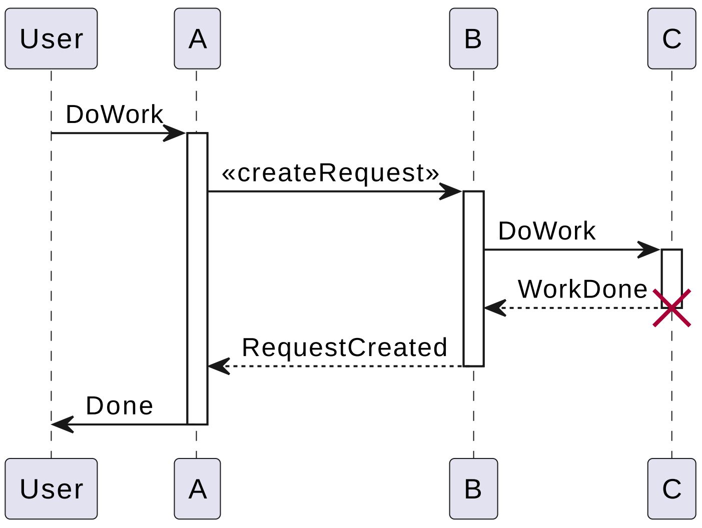
Primary and supporting elements: Depends on the diagram scope; enterprise (see System Landscape diagram), software system (see System Context or Container diagrams), container (see Component diagram).

Intended audience: Technical and non-technical people, inside and outside of the software development team.

Extended Docs

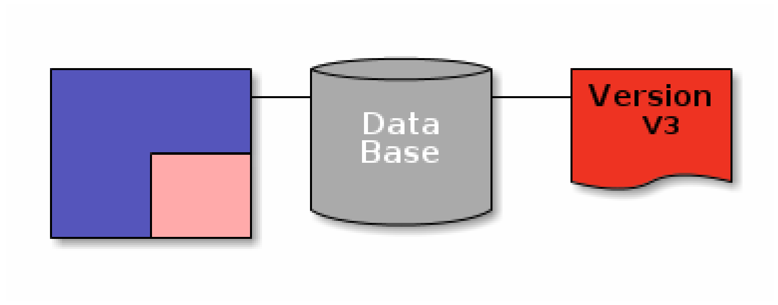
`/src/1 Internet Banking System/Single Page Application/Extended Docs`





Multiple markdowns can be ordered using `<name>.1.md, <name>.2.md .. <name>.<n>.md`

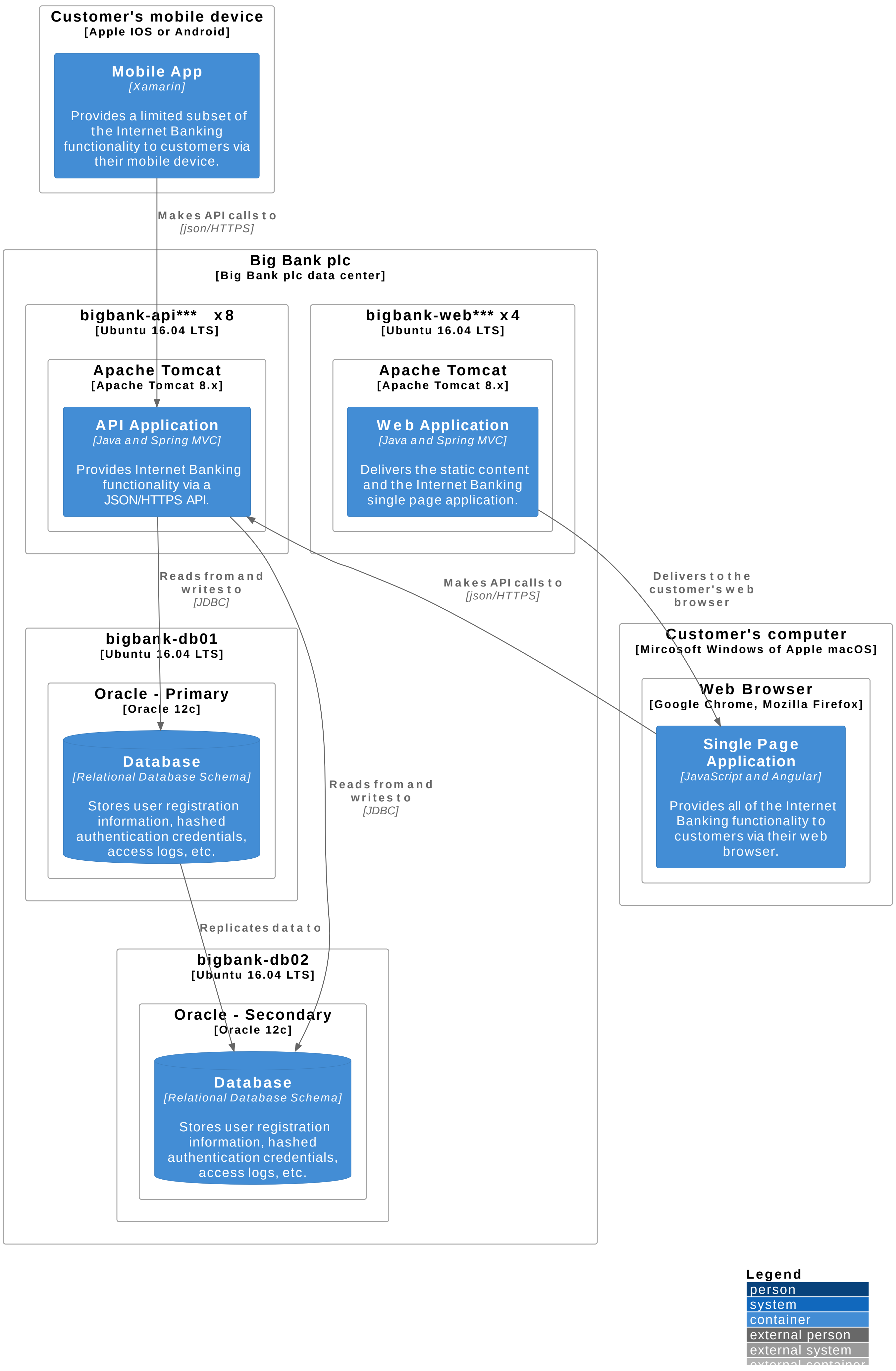
You can choose where to place a certain diagram by using `![name](<diagram name>.puml)`



Feel free to add any additional details necesary.

2 Deployment

`/src/2 Deployment`



Customer's mobile device
[Apple IOS or Android]
Mobile App
[Xamarin]
Provides a limited subset of the Internet Banking functionality to customers via their mobile device.
Makes API calls to
[json/HTTPS]
Big Bank plc
[Big Bank plc data center]
bigbank-api*** x8
[Ubuntu 16.04 LTS]
Apache Tomcat
[Apache Tomcat 8.x]
API Application
[Java and Spring MVC]
Provides Internet Banking functionality via a JSON/HTTPS API.
bigbank-web*** x4
[Ubuntu 16.04 LTS]
Apache Tomcat
[Apache Tomcat 8.x]
Web Application
[Java and Spring MVC]
Delivers the static content and the Internet Banking single page application.
Reads from and writes to
[JDBC]
Makes API calls to
[json/HTTPS]
Delivers to the customer's web browser
Customer's computer
[Mircosoft Windows of Apple macOS]
Web Browser
[Google Chrome, Mozilla Firefox]
Single Page Application
[JavaScript and Angular]
Provides all of the Internet Banking functionality to customers via their web browser.
bigbank-db01
[Ubuntu 16.04 LTS]
Oracle - Primary
[Oracle 12c]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
Replicates data to
bigbank-db02
[Ubuntu 16.04 LTS]
Oracle - Secondary
[Oracle 12c]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Legend
person
system
container
external person
external system

Deployment diagram

A deployment diagram allows you to illustrate how containers in the static model are mapped to infrastructure. This deployment diagram is based upon a UML deployment diagram, although simplified slightly to show the mapping between containers and deployment nodes. A deployment node is something like physical infrastructure (e.g. a physical server or device), virtualised infrastructure (e.g. IaaS, PaaS, a virtual machine), containerised infrastructure (e.g. a Docker container), an execution environment (e.g. a database server, Java EE web/application server, Microsoft IIS), etc. Deployment nodes can be nested.

Scope: A single software system.

Primary elements: Deployment nodes and containers within the software system in scope.

Intended audience: Technical people inside and outside of the software development team; including software architects, developers and operations/support staff.