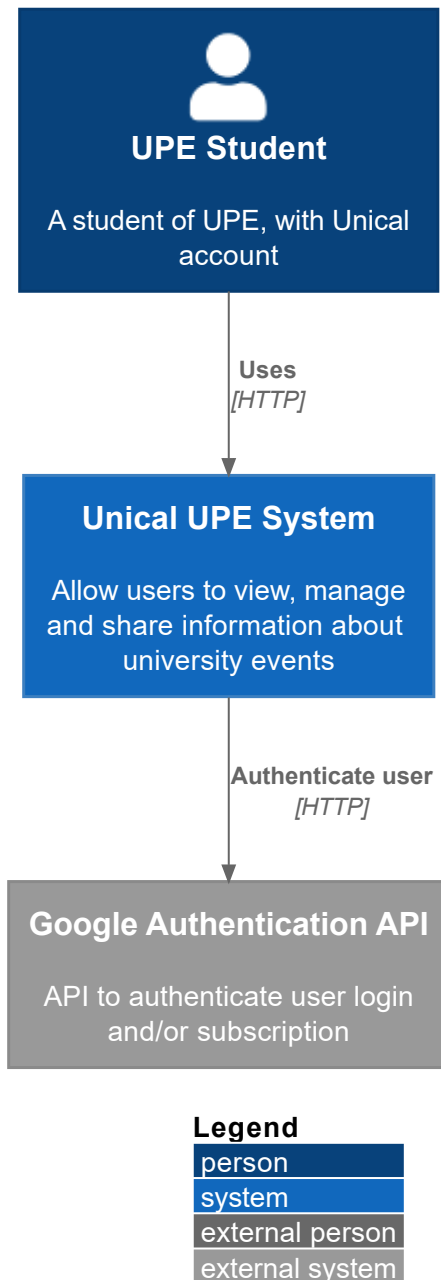


Unical UPE

- Context
 - 1 Unical UPE System
 - 1.1 APIComponent
 - 1.2 Mobile App
-

Context



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.

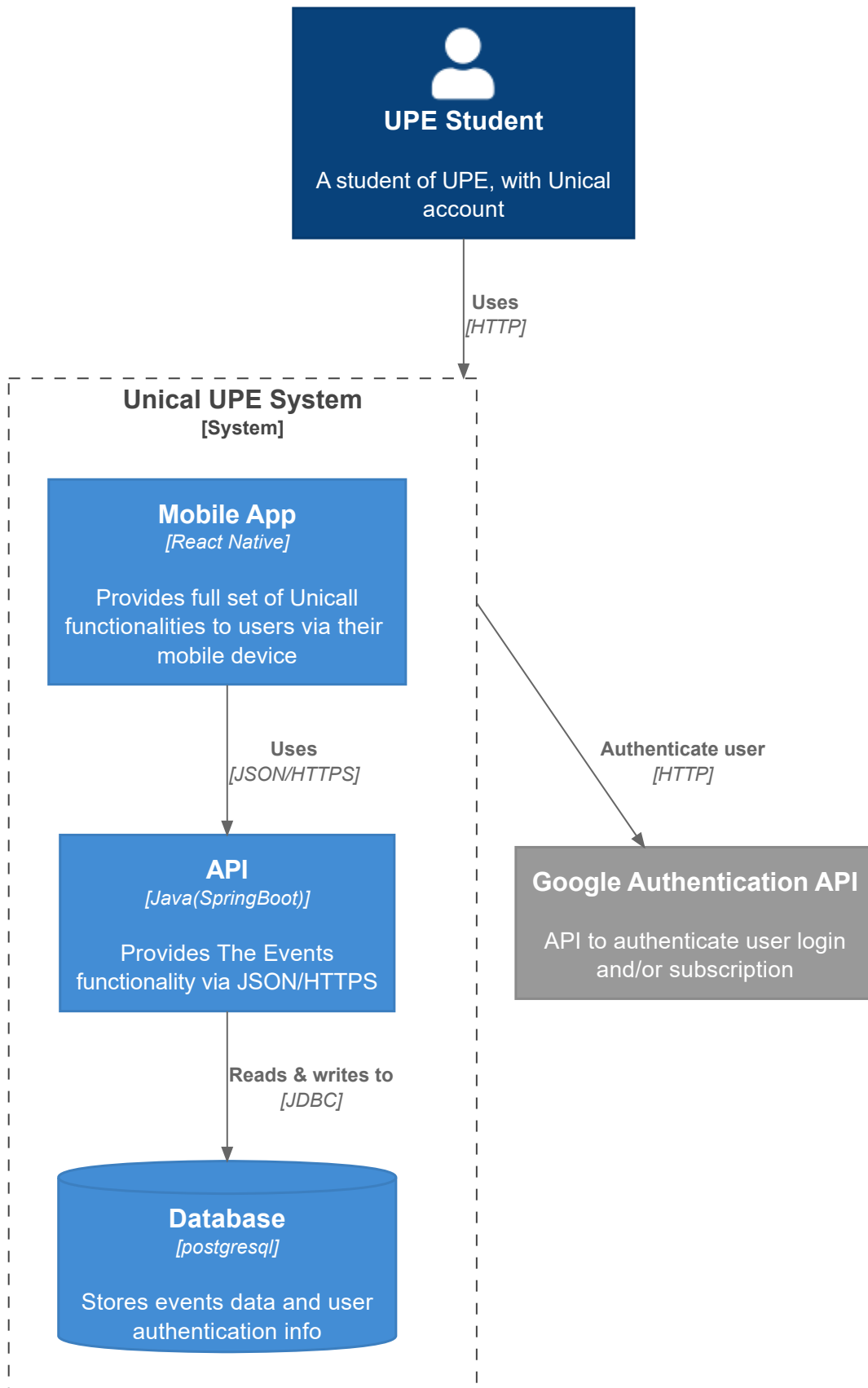
Scope: A single software system.

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 Unical UPE System

\1 Unical UPE System



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.

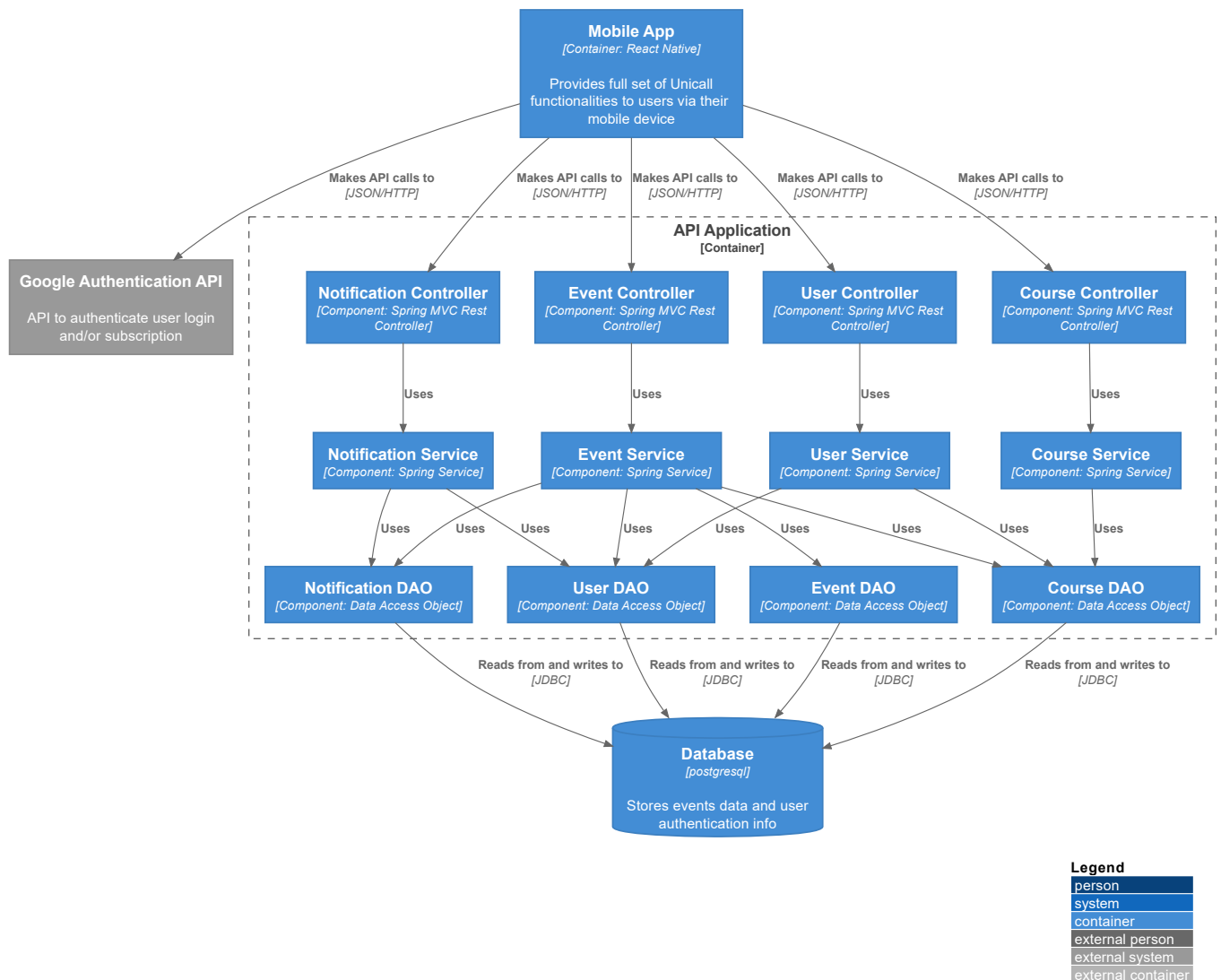
Primary elements: Containers within the software system in scope. Supporting elements: People and software systems directly connected to the containers.

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.

1.1 APIComponent

\1 Unical UPE System\1.1 APIComponent



Level 2: 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.

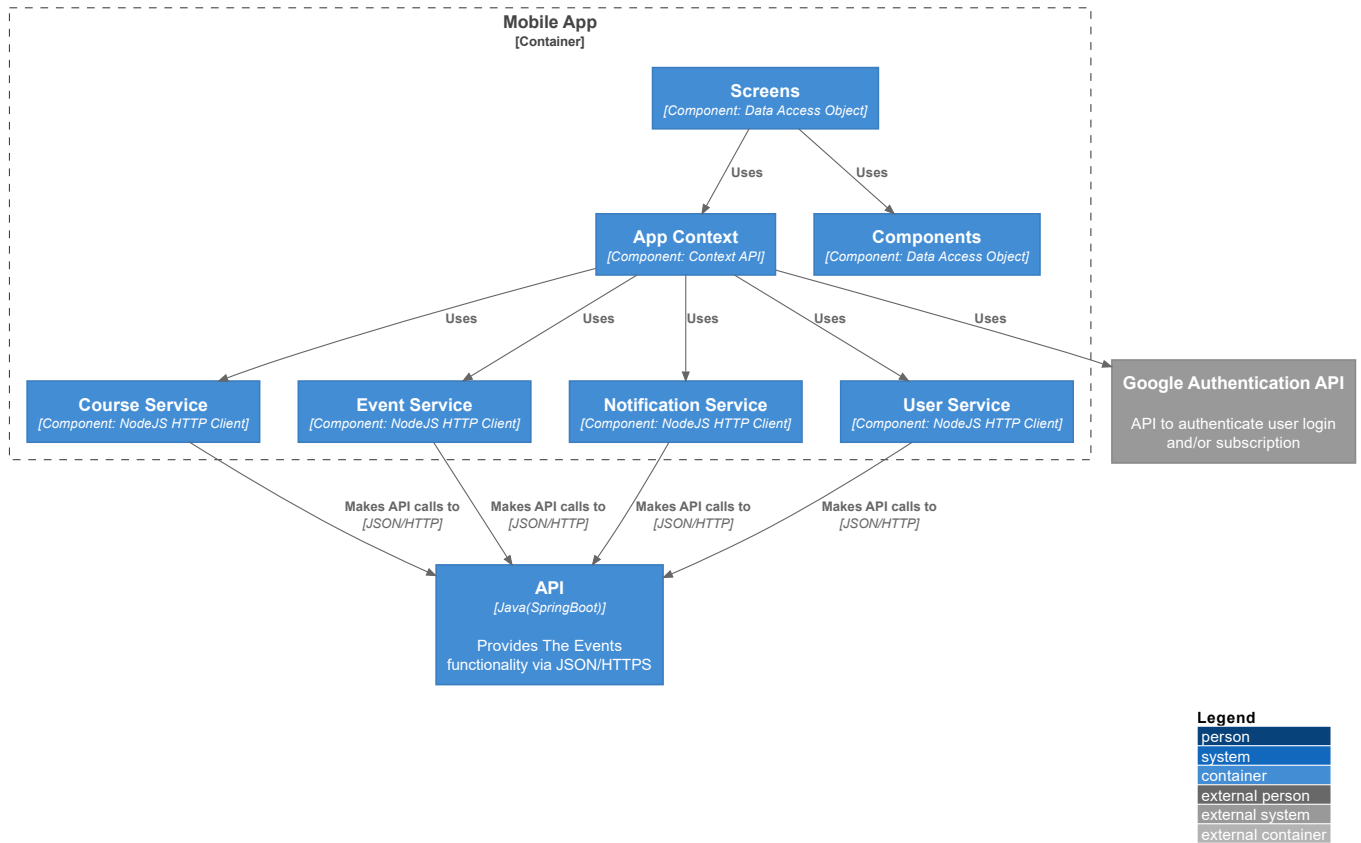
Primary elements: Components within the container in scope. Supporting elements: Containers (within the software system in scope) plus people and software systems directly connected to the components.

Intended audience: Software architects and developers.

Notes: uses component diagrams if you feel they add value, and consider automating their creation for long-lived documentation.

1.2 Mobile App

\1 Unical UPE System\1.2 Mobile App



Level 2: 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.

Notes: uses component diagrams if you feel they add value, and consider automating their creation for long-lived documentation.