sad

Overview

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.
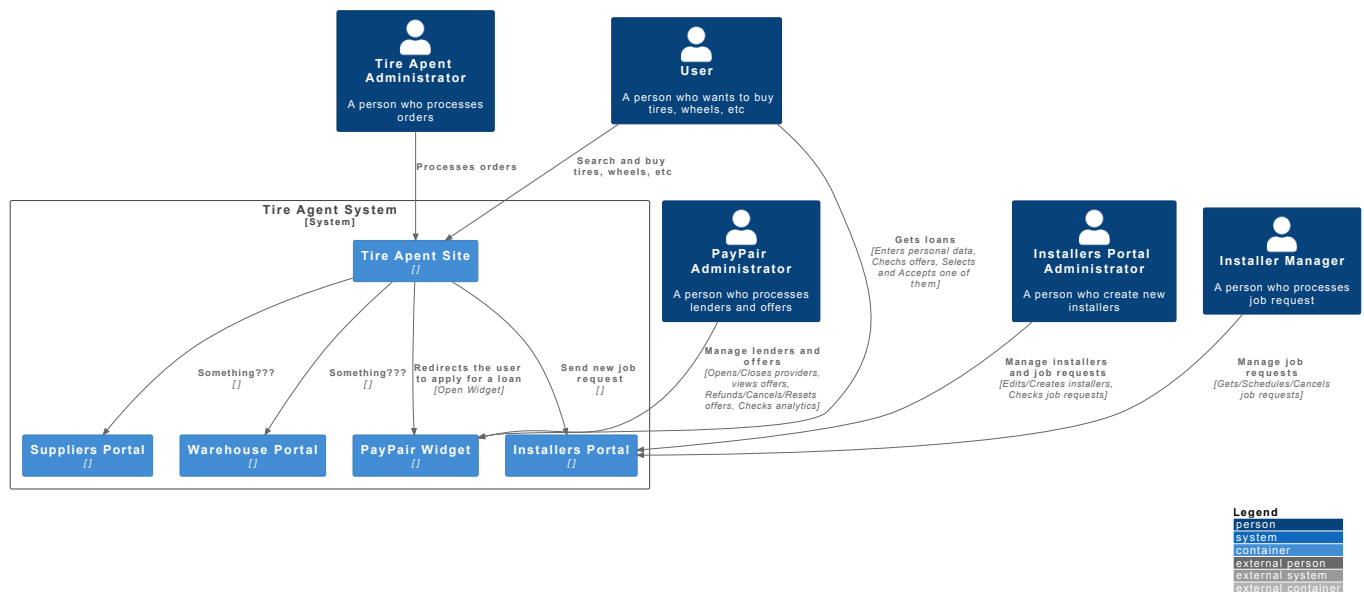
Scope: A single software system.

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.

1.1.1 Diagram for all system

/1.1.1 Diagram for all system

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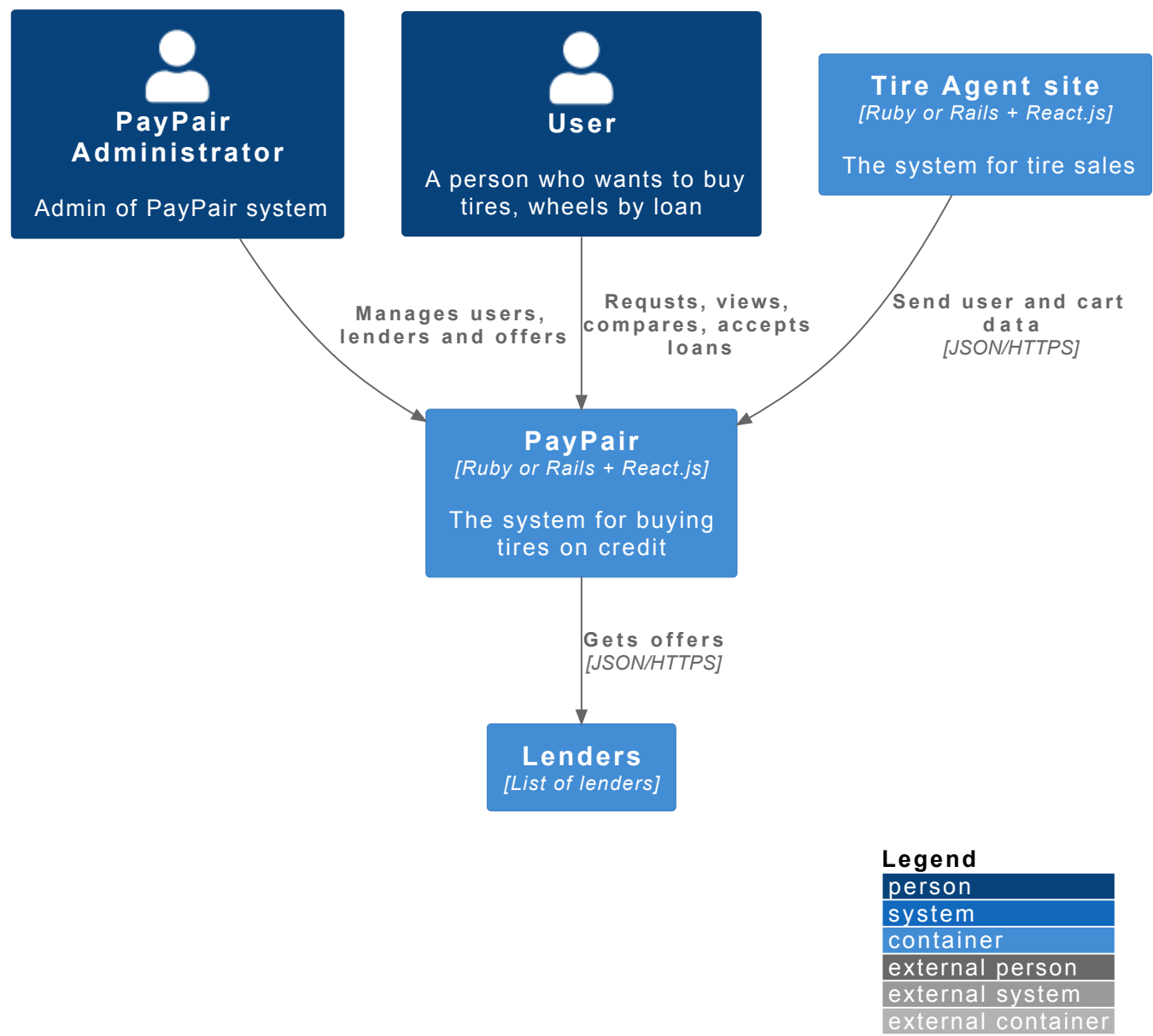
Notes: This diagram says nothing about deployment scenarios, clustering, replication, failover, etc.

1.3 Diagrams for PayPair

1.3.1 Context Diagram

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.

1.3.2 Containers Diagram

/1.3 Diagrams for PayPair/1.3.2 Containers Diagram

PayPair Administrator

Admin of PayPair system

User

A person who wants to buy tires, wheels by loan

Manages users, lenders and offers

Requsts, views, compares, accepts loans

PayPair System
[System]

PayPair Admin Panel
[Ruby or Rails + JS]

Allows admin manage users and offers

PayPair Widget
[React.js]

Allows users get, check and choose loans

Uses

Make API calls
[JSON/HTTPS]

PayPair Backend
[Ruby or Rails]

Provides PayPair functionality via JSON/HTTPS

Reads and writes
[Rails Active Record ORM]

Send user data

Send offers

Database
[PostgresQL]

Stores user infomation, informations about user requests and provider's offers

Lenders

Flexshopper lender
[]

Paytomorrow lender
[]

Okinus lender
[]

Uown lender
[]

Tempoe lender
[]

Aff lender
[]

Acima lender
[]

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.
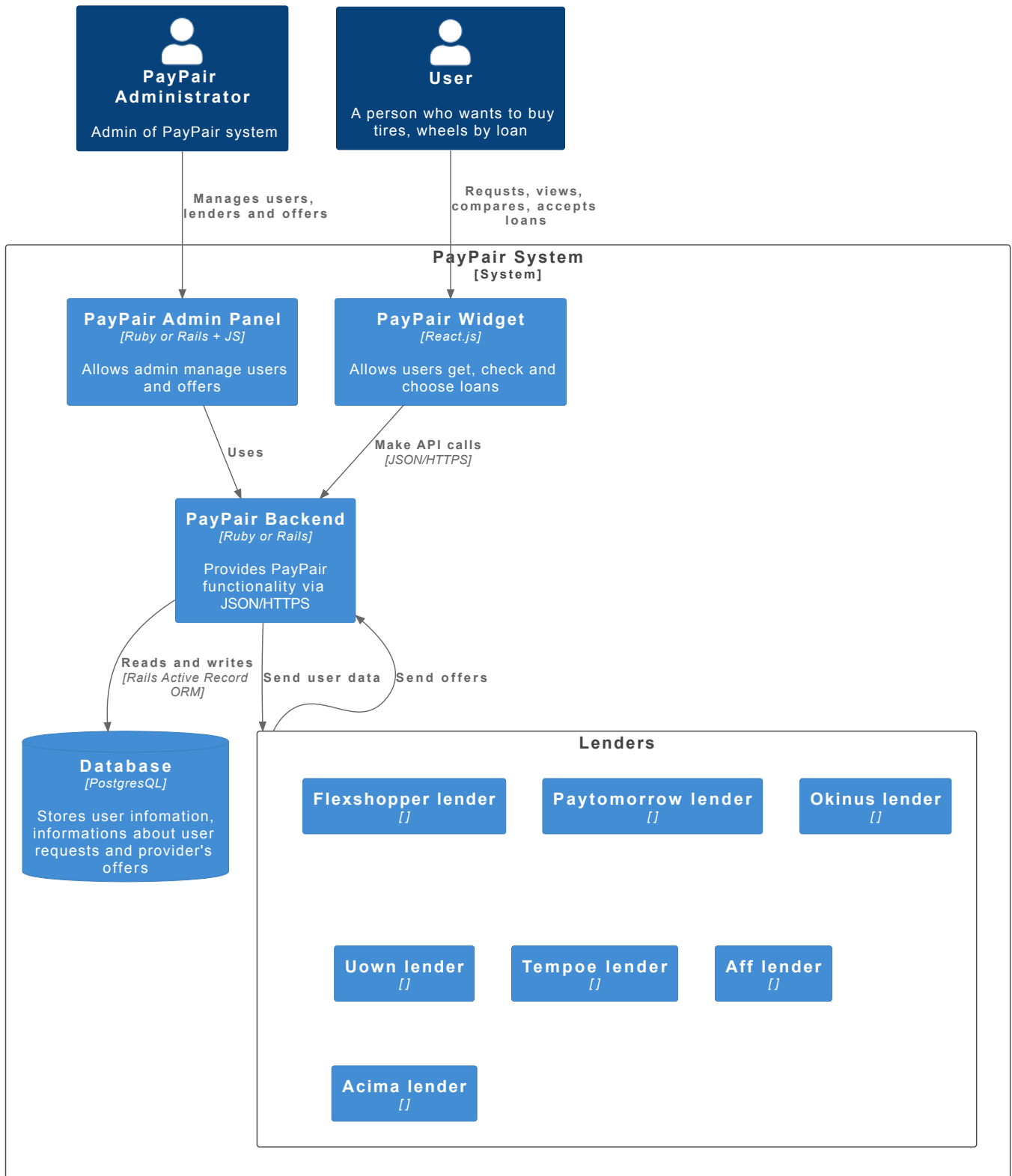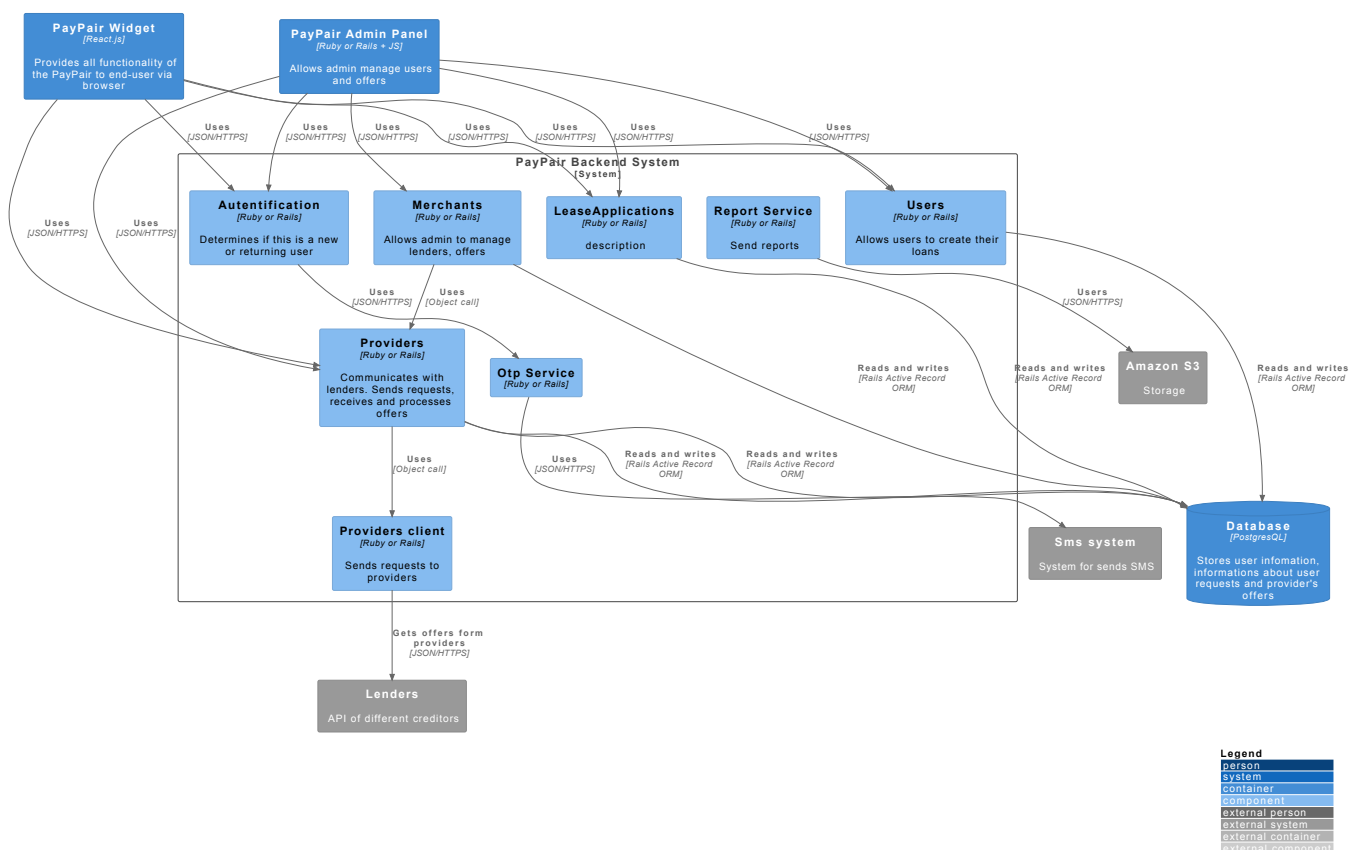
Scope: A single software system.

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.

1.3.3 Components Diagram

/1.3 Diagrams for PayPair/1.3.3 Components Diagram

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.

1.4 Diagrams for TireAgent Installers Portal

/1.4 Diagrams for TireAgent Installers Portal

1.4.1 Context Diagram

/1.4 Diagrams for TireAgent Installers Portal/1.4.1 Context Diagram

Super Administrator

Admin of Installer portal

Installer Administrator

Admin of installer account

Employee

Tire Agent employee

Manage installers accounts

Manages installer's orders

Manage all orders

Installer portal
[Ruby or Rails]

The system for scheduling, managing and installing appointments

Update order's status with installation option
[JSON/HTTPS]

Send orders with installation option
[JSON/HTTPS]

Tire Agent site
[Ruby or Rails + React.js]

The system for tire sales

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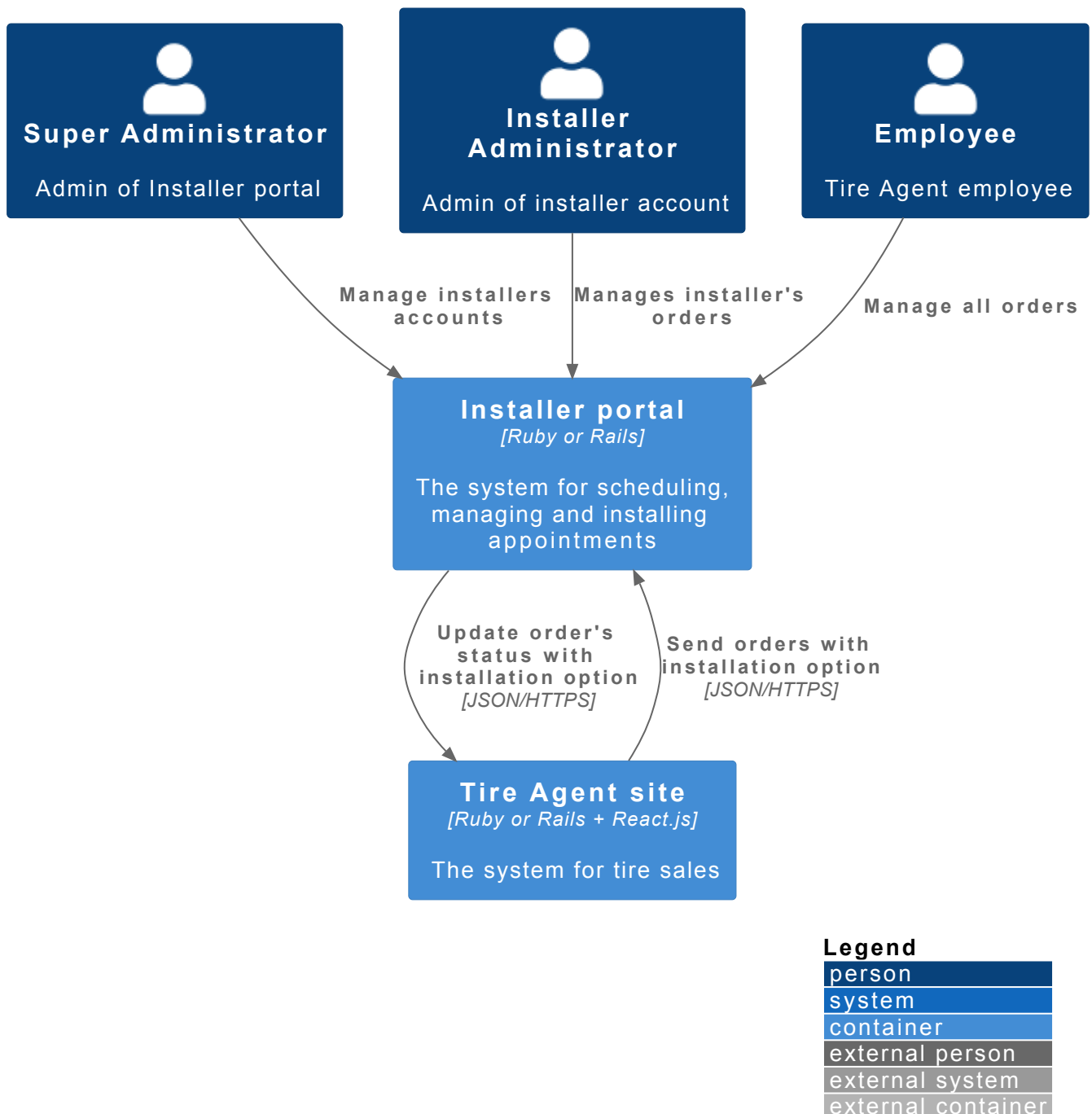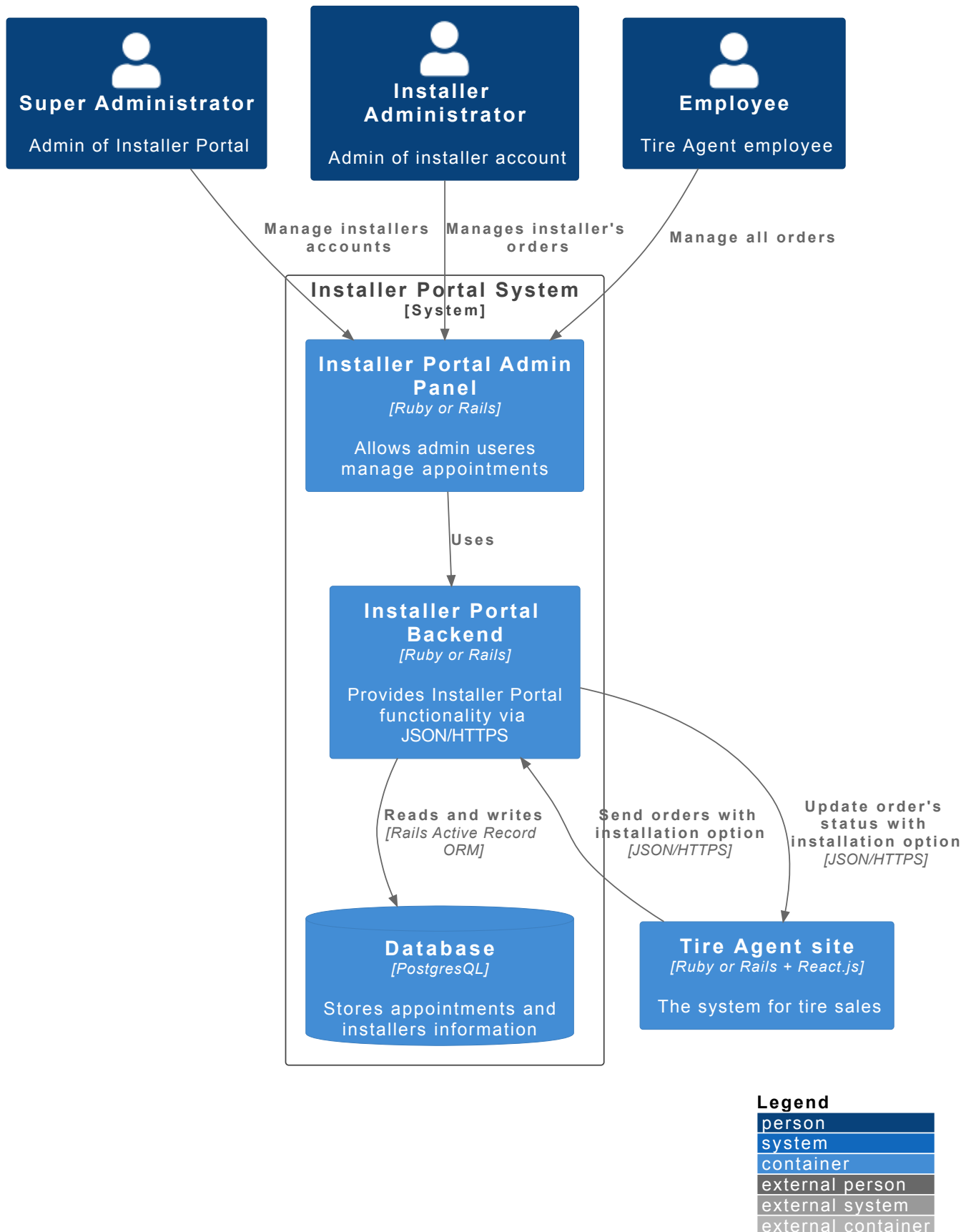
Notes: This diagram says nothing about deployment scenarios, clustering, replication, failover, etc.

1.4.2 Containers Diagram

/1.4 Diagrams for TireAgent Installers Portal/1.4.2 Containers Diagram

Legend

person
system
container
external person
external system
external container

Diagram content:

Super Administrator — Admin of Installer Portal

Installer Administrator — Admin of installer account

Employee — Tire Agent employee

Manage installers accounts

Manages installer's orders

Manage all orders

Installer Portal System
[System]

Installer Portal Admin Panel
[Ruby or Rails]

Allows admin useres manage appointments

Uses

Installer Portal Backend
[Ruby or Rails]

Provides Installer Portal functionality via JSON/HTTPS

Reads and writes
[Rails Active Record ORM]

Send orders with installation option
[JSON/HTTPS]

Update order's status with installation option
[JSON/HTTPS]

Database
[PostgresQL]

Stores appointments and installers information

Tire Agent site
[Ruby or Rails + React.js]

The system for tire sales

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.
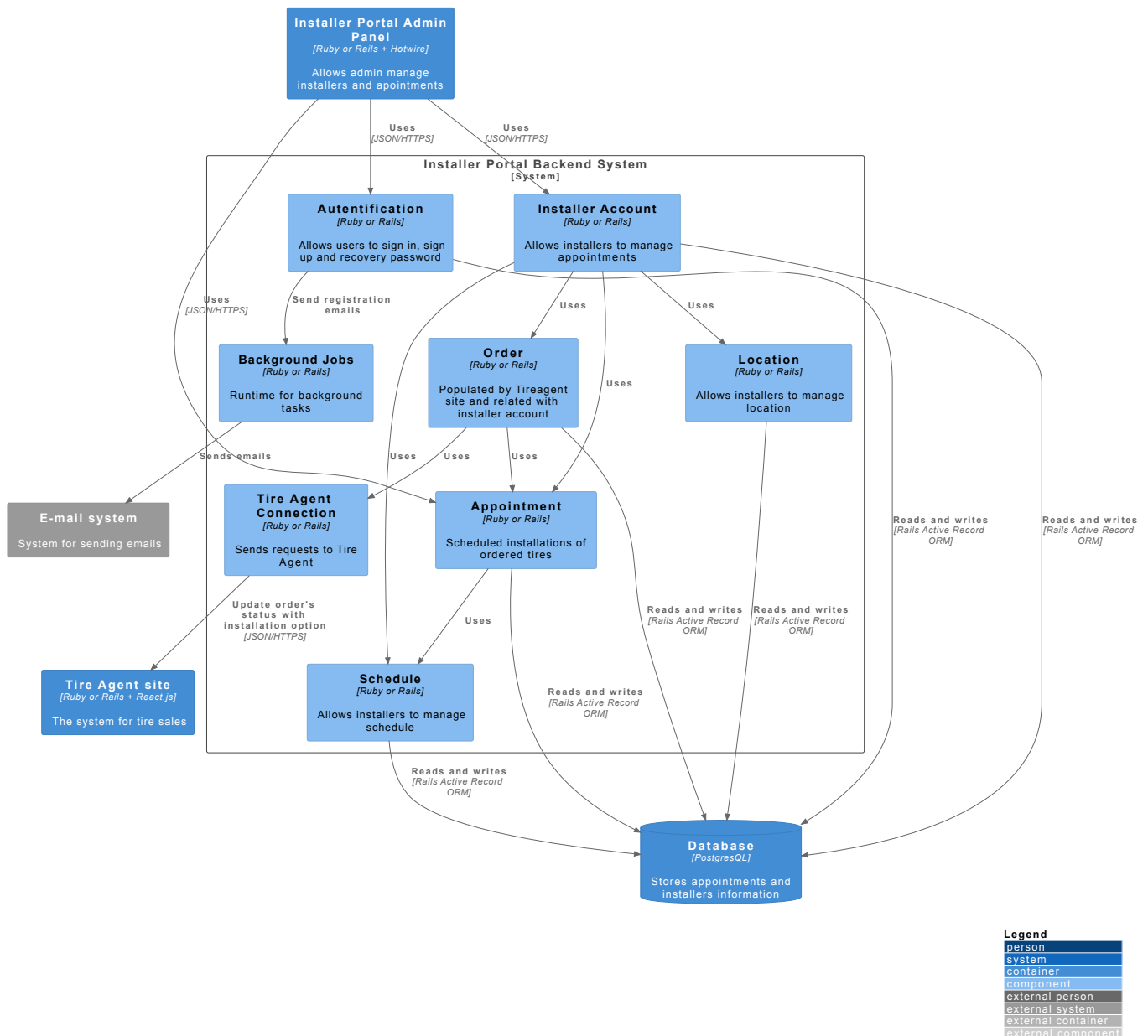
Scope: A single software system.

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.

1.4.3 Components Diagram

/1.4 Diagrams for TireAgent Installers Portal/1.4.3 Components Diagram

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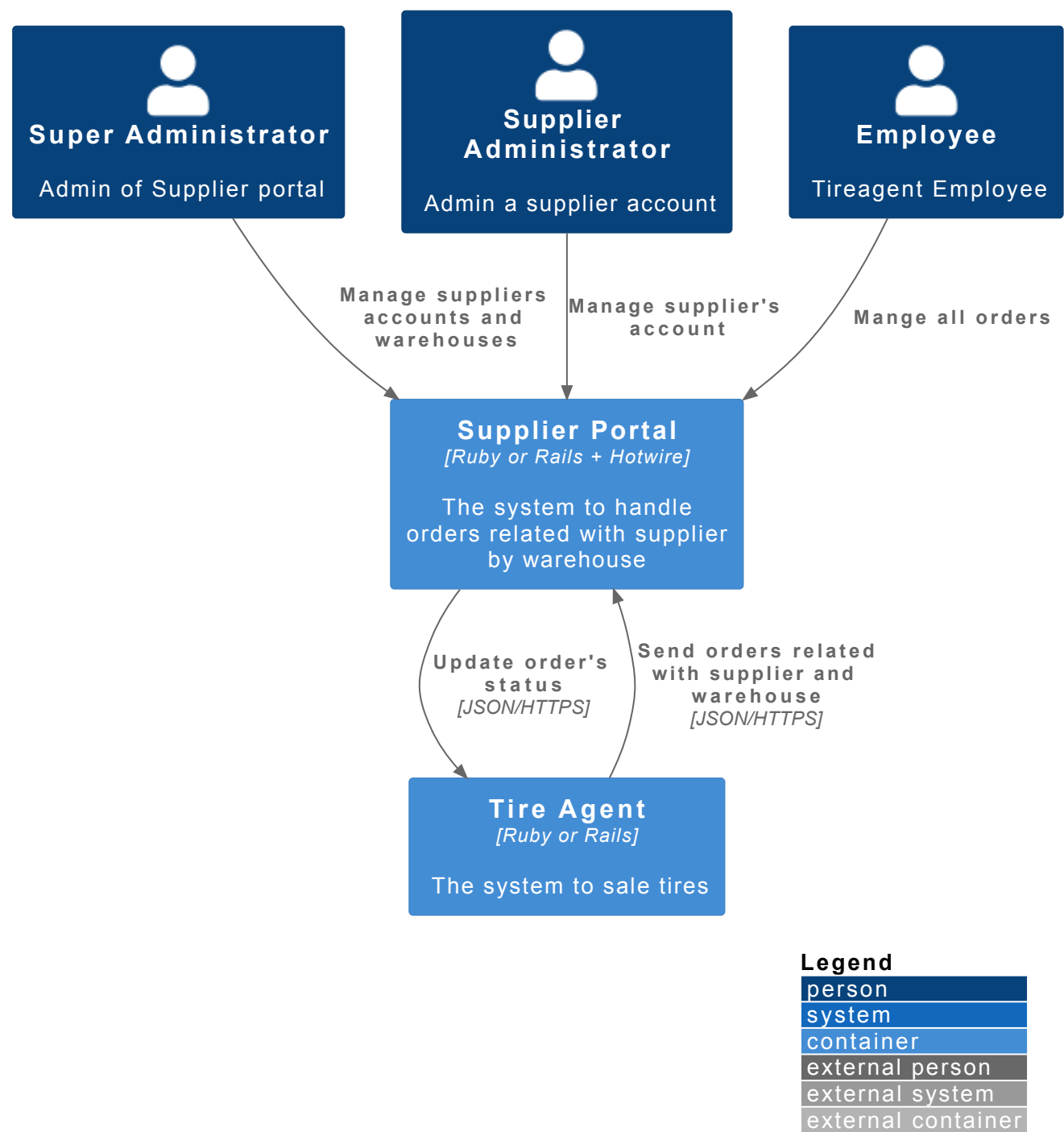
Notes: This diagram says nothing about deployment scenarios, clustering, replication, failover, etc.

1.6 Diagrams for TireAgent Supplier Portal

/1.6 Diagrams for TireAgent Supplier Portal

1.6.1 Context Diagram

/1.6 Diagrams for TireAgent Supplier Portal/1.6.1 Context Diagram

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.
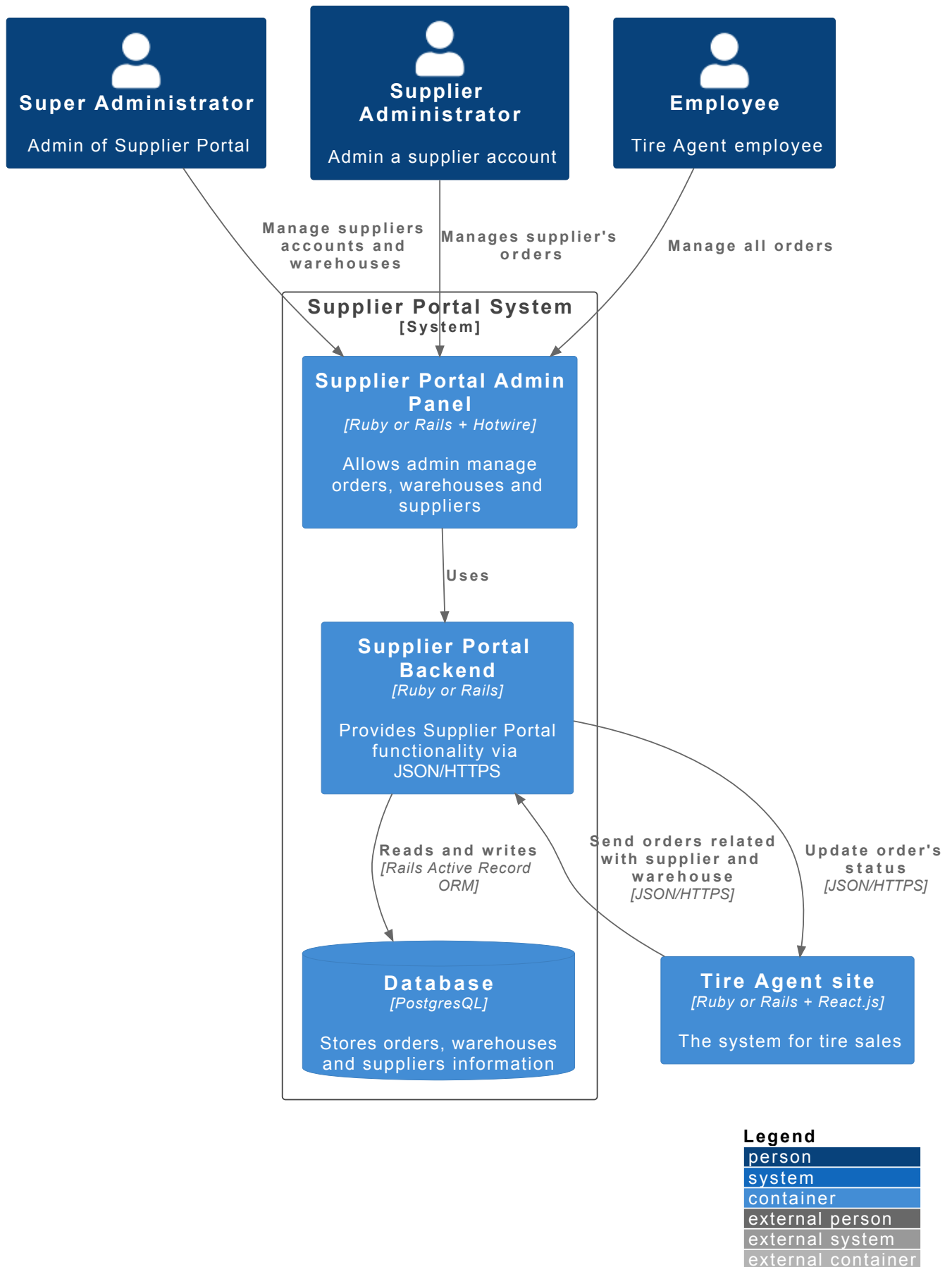
Scope: A single software system.

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.

1.6.2 Containers Diagram

/1.6 Diagrams for TireAgent Supplier Portal/1.6.2 Containers Diagram

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.

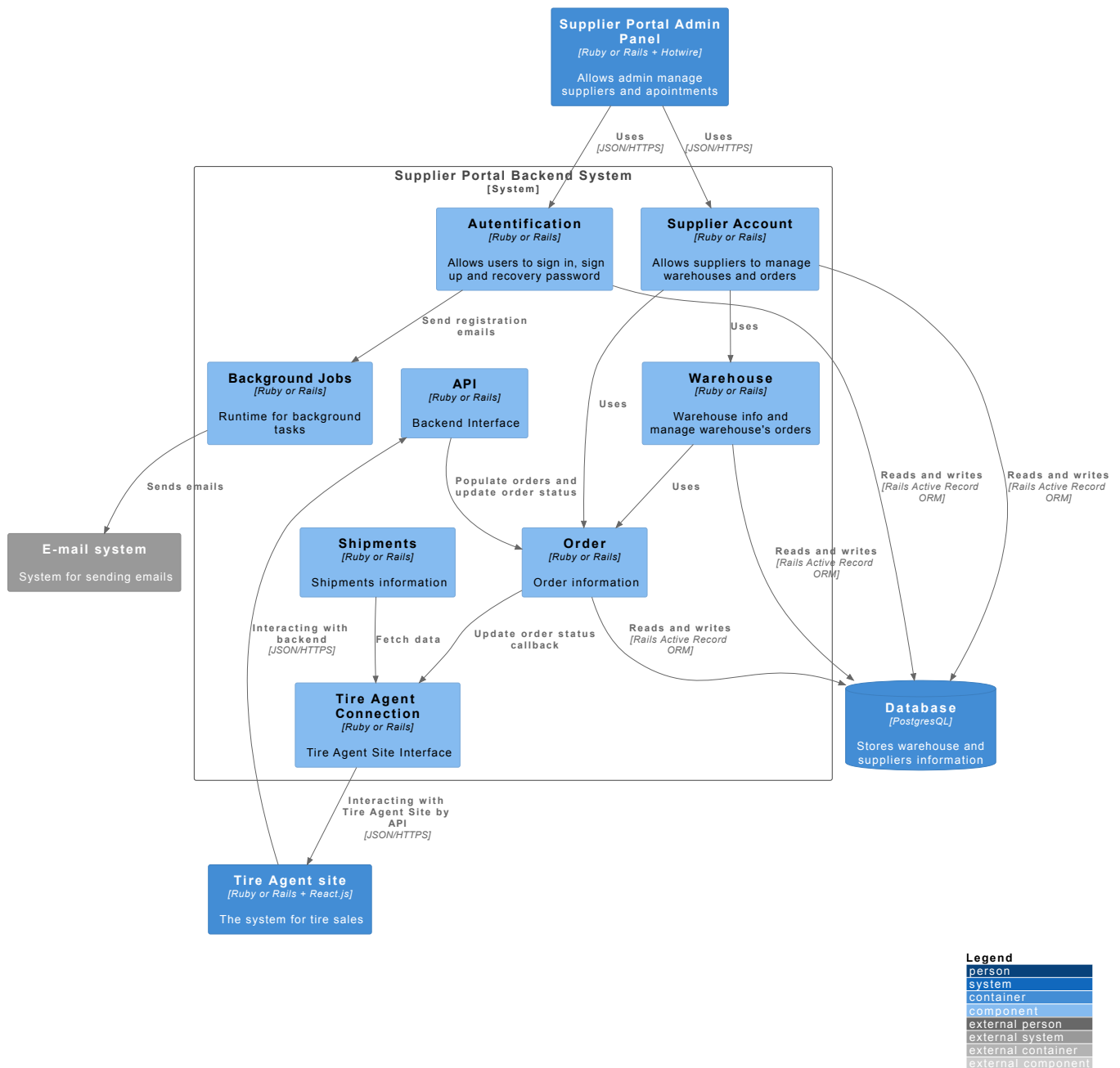**Scope**: A single software system.

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.

1.6.3 Components Diagram

/1.6 Diagrams for TireAgent Supplier Portal/1.6.3 Components Diagram

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.

data

/data