

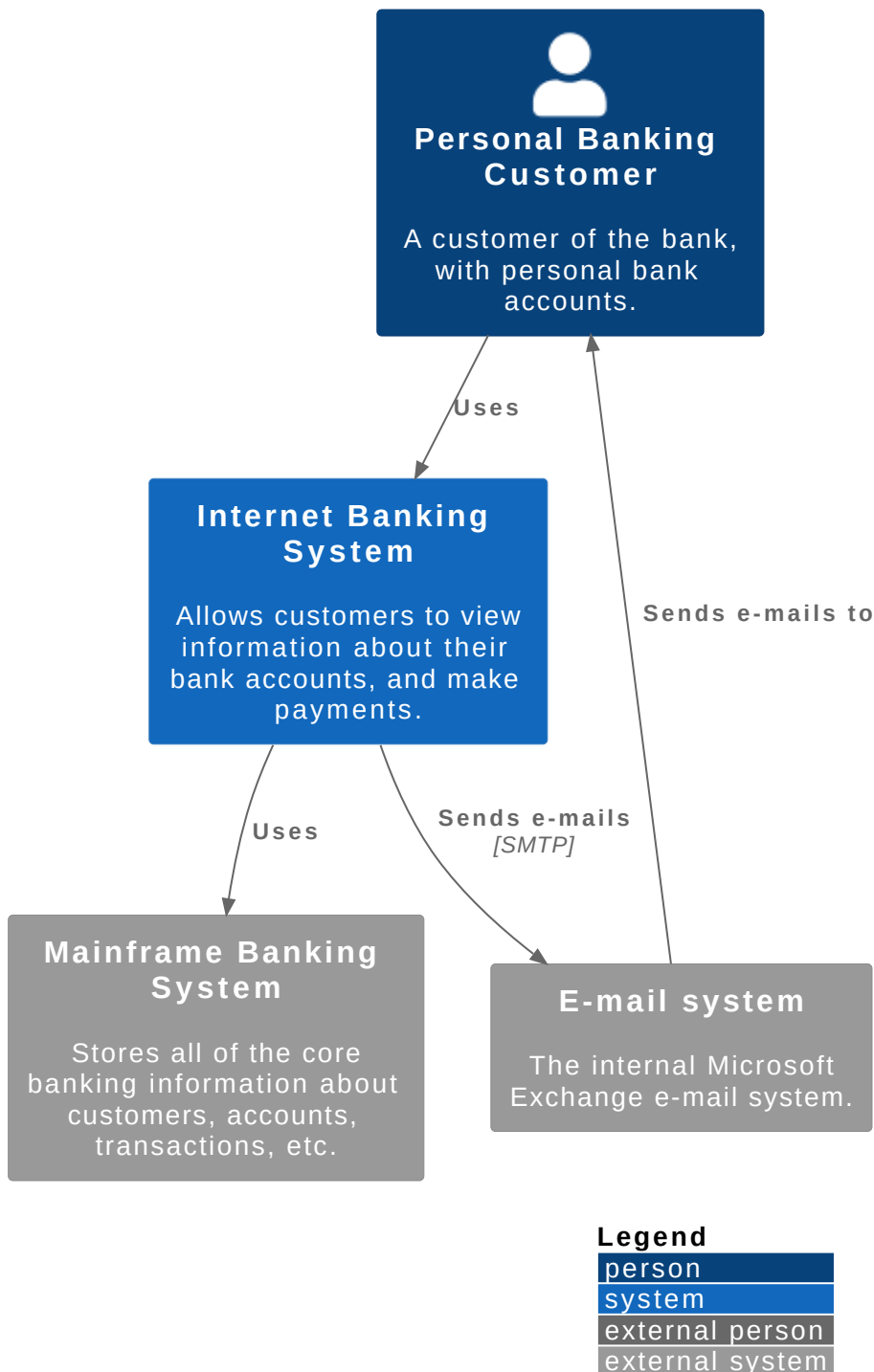
# sad

---

- [Overview](#)
  - [1.1.1 Diagram for all system](#)
  - [1.3 Diagrams for PayPair](#)
    - [1.3.1 Context Diagram](#)
    - [1.3.2 Containers Diagram](#)
    - [1.3.3 Components Diagram](#)
    - [1.3.4 Entity Diagram](#)
    - [1.3.5 Activity Diagram](#)
  - [1.4 Diagrams for TireAgent Installers Portal](#)
    - [1.4.1 Context Diagram](#)
    - [1.4.2 Containers Diagram](#)
    - [1.4.3 Components Diagram](#)
    - [1.4.4 Entity Diagram](#)
    - [1.4.5 Activity Diagram](#)
  - [1.6 Diagrams for TireAgent Supplier Portal](#)
    - [1.6.1 Context Diagram](#)
    - [1.6.2 Containers Diagram](#)
    - [1.6.3 Components Diagram](#)
  - [data](#)

---

## Overview



### 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.

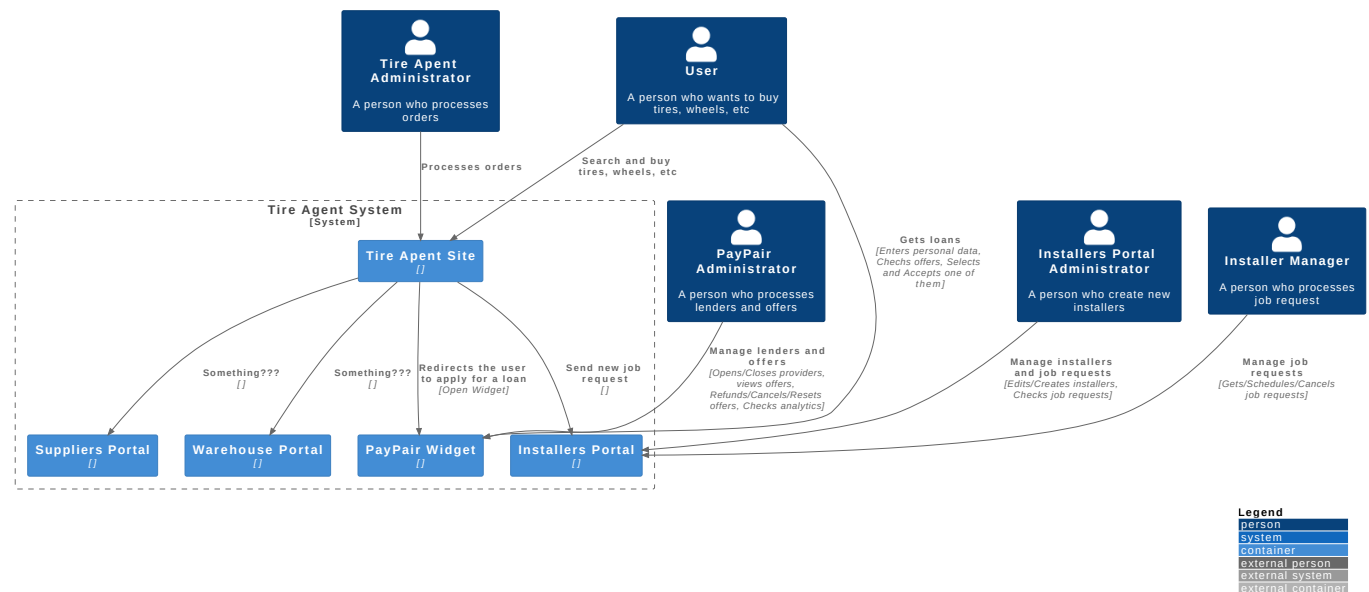
**Scope:** A single software system.

**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.

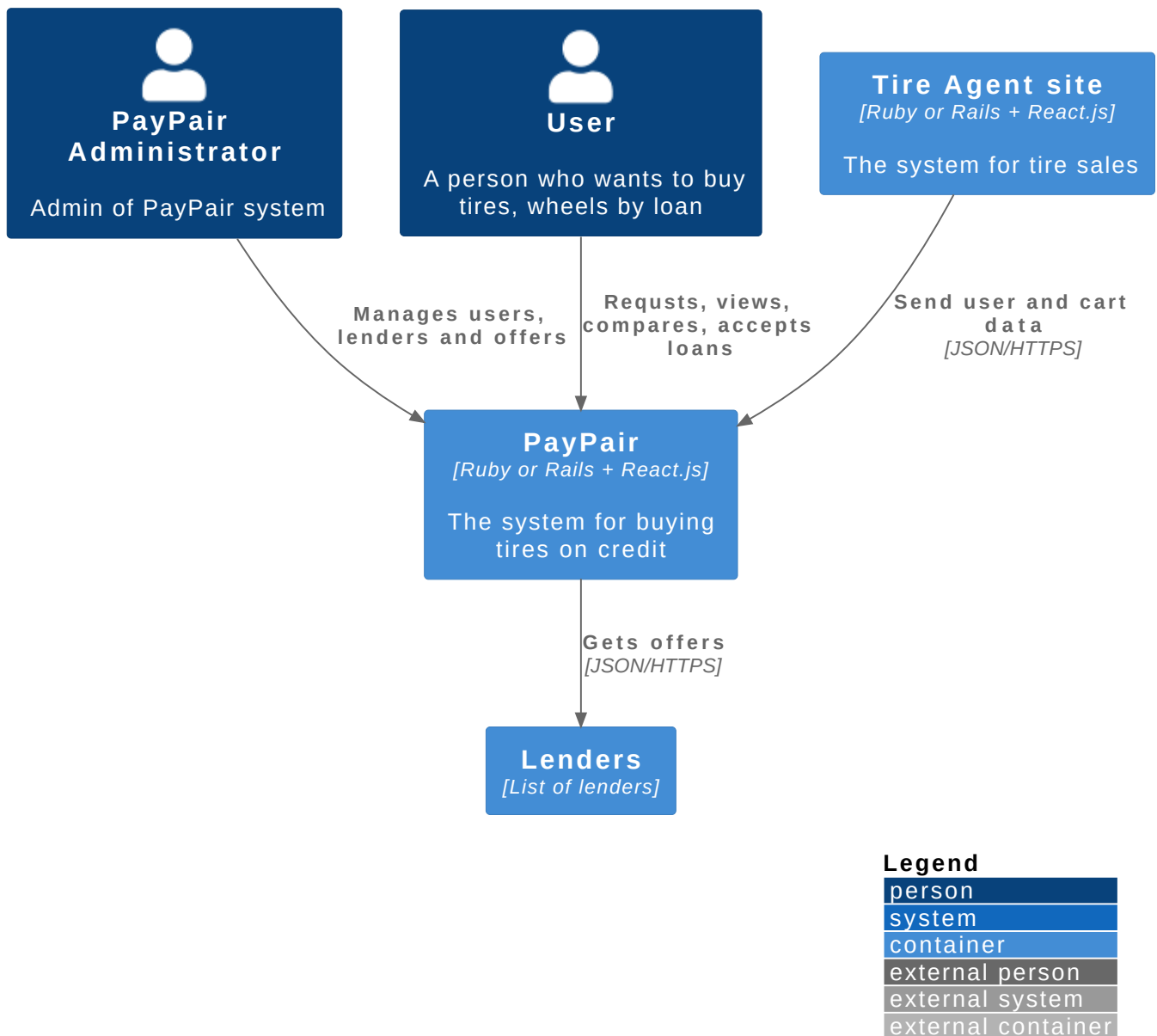
**Notes:** This diagram says nothing about deployment scenarios, clustering, replication, failover, etc.

## 1.3 Diagrams for PayPair

### /1.3 Diagrams for PayPair

#### 1.3.1 Context Diagram

##### /1.3 Diagrams for PayPair/1.3.1 Context Diagram



Persons:

The system used by User and PayPair Admin

**User** is person in system who requested loans. User enters personal data. Then user sees all the loan offers, user can look at each of them, compare and choose the most suitable option.

**PayPair Admin** is person in system who manage users, lenders and loans. The administrator can manage lenders. It can enable/disable lenders. Can view all loan applications and lender survey results. Can cancel or make a partial refund. View analytics

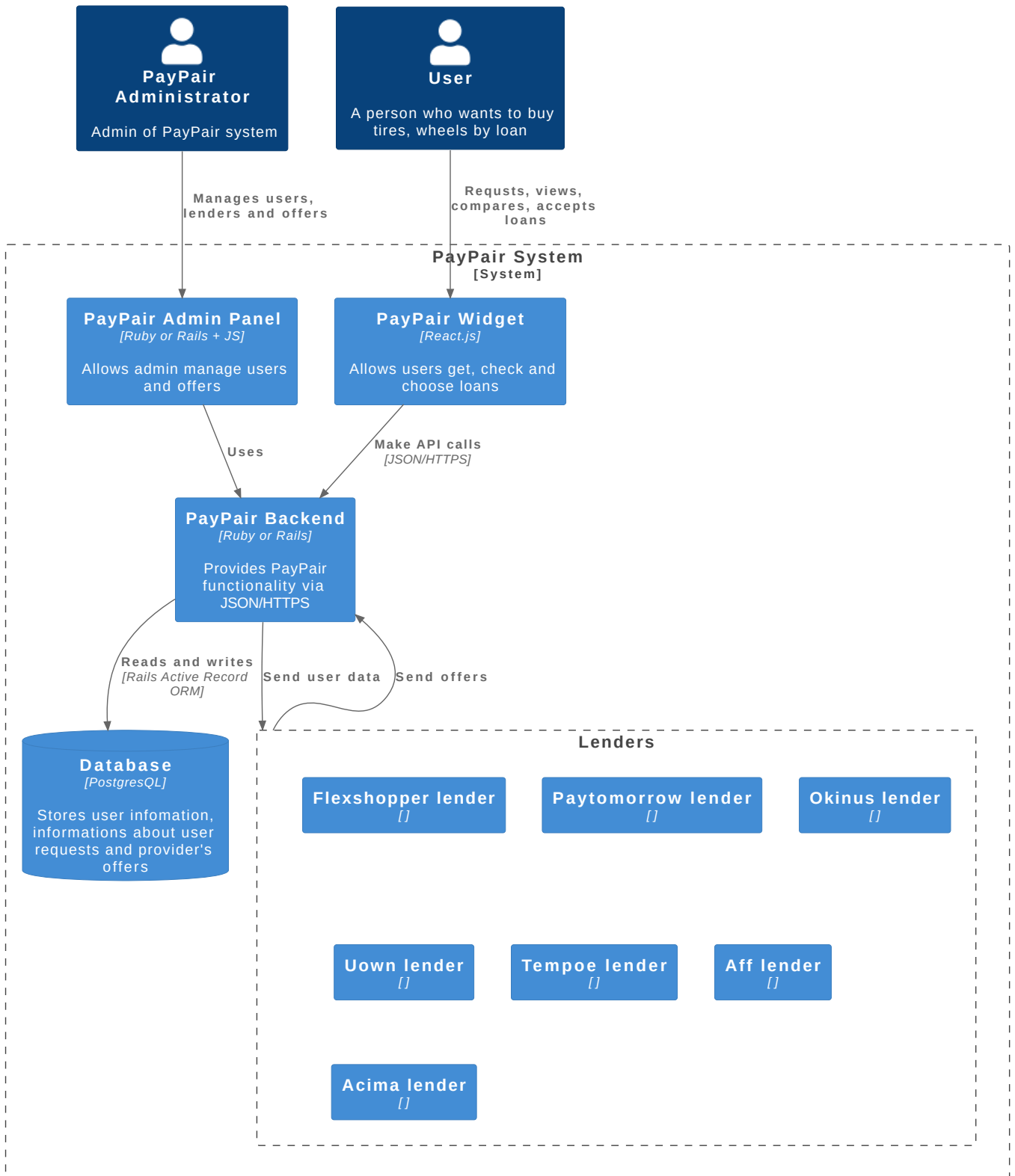
Third-parties Services

**Tire Agent site** is sends information about user: first name, last name, address, phone number, ect, and cadt details. PayPaid uses this data to bould request to lenders.

**Lenders** is companies to provide loans for users.

## 1.3.2 Containers Diagram

/1.3 Diagrams for PayPair/1.3.2 Containers Diagram



The System consists of the following elements:

PayPair Admin Panel. This application is used by the administrator to provide functionality for it. Admin Panel based on [Rails View](#)

PayPair Widget. This application delivers to the end-user browser and provides functionality through it. This application is based on [React/Redux](#).

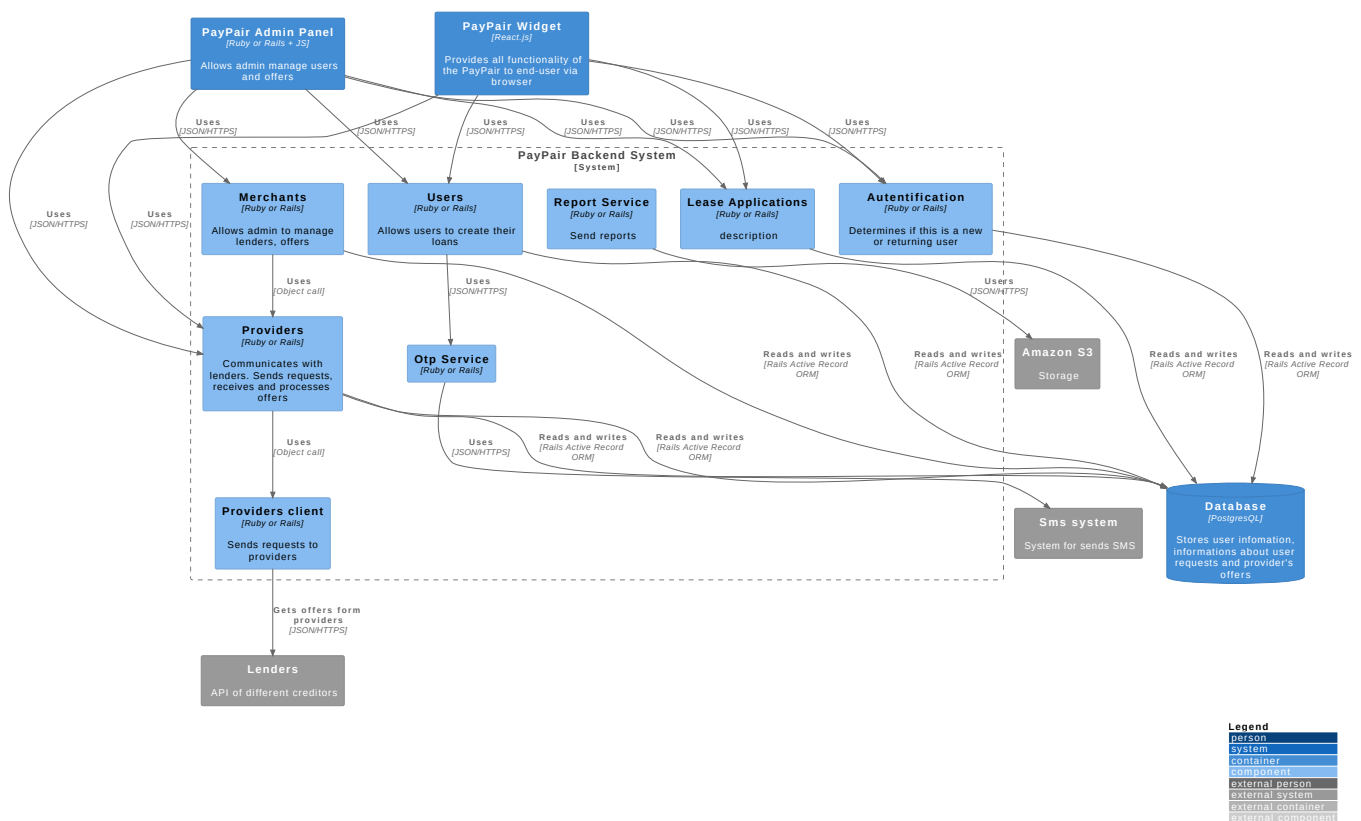
PayPair Backend. This application provides functionality through JSON/HTTP interface. This application is based on [Ruby/Ruby on Rails](#). It has integration with many Lenders services.

Lenders. It is lenders API. They provide loan offers for the end-user

Database. This element is responsible for the data store: users data, offers, providers data, etc.

### 1.3.3 Components Diagram

#### /1.3 Diagrams for PayPair/1.3.3 Components Diagram



API application is a monolith application based on Ruby/Ruby on Rails. API application can be divided into two parts: Admin Panel and Backend Part. Backend logic is separated using [Interactors](#). It means that the functionality of the system is divided into related parts according to the [High cohesion and low coupling](#) principle (the same principles are followed by microservices architecture pattern).

API application consists of the following parts:

Autentification (Ruby on Rails). This module is responsible for Admin sign-in/sign up. [JSON Web Token](#) will be Admin for user authentication purposes.

User (Ruby on Rails). This module is responsible for user creating, searching existing users, checking previous approvals, authorizing etc.

Otp Service (Ruby on Rails). This module is responsible for sending OTP code to users by SMS. Module can use two sms providers. These providers are managed by the administrator.

Providers (Ruby on Rails). This module is responsible for building requests to lenders, response This module is responsible for, creating offers data. ect.

Merchants (Ruby on Rails). This module is responsible for Admin functionality - creating admins, managing providers, checking offers and users, ect.

Providers client (Ruby on Rails). This module is responsible for sending requests to providers.

Lease Applications (Ruby on Rails). This module is responsible for storing users requests, calculating taxes.

Report Service (Ruby on Rails). This module is responsible for creating different reports - new users report, approved offers report, ect.

## 1.3.4 Entity Diagram

### /1.3 Diagrams for PayPair/1.3.4 Entity Diagram

Entity diagram describes entities in the system and relations between them.

Merchant. This entity represents merchant account. It is company which use PayPaid. Each merchant has personal agreements with creditors, settings.

Admin. This entity represents an employee who manages customers (users), creditors, offers and approvals

MerchantLpConfiguration. This entity represents the provider setting and contains the personal credentials of each provider.

Setting. This entity represents merchant settings. This contains settings such as choosing an SMS service for sending a OTP code and setting up work with providers

Subscriber

Trending. This entity represents logs. This shows whith screens user visit and where user close PayPair.

User. This entity represents the end-user in the system.

NewUser. This entity was creating for analytics. This also represents end-user but with needed data for analytics.

ApplicationRequest. This entity represents loan request from user to PayPair system.

LeaseApplication. This entity represents responses form providers. It shows whether the provider approved the loan request or not, information about payment schedule. It is also updated if the user took advantage of the offer or if the user after a while did cancel, refund or settle.

OrderItem. This entity represents user's cart. It contains product information.

RefundedItem. This entity represents items whith was refunded.

## 1.3.5 Activity Diagram

### /1.3 Diagrams for PayPair/1.3.5 Activity Diagram

According to the flow, the process of obtaining a loan will be as follows:



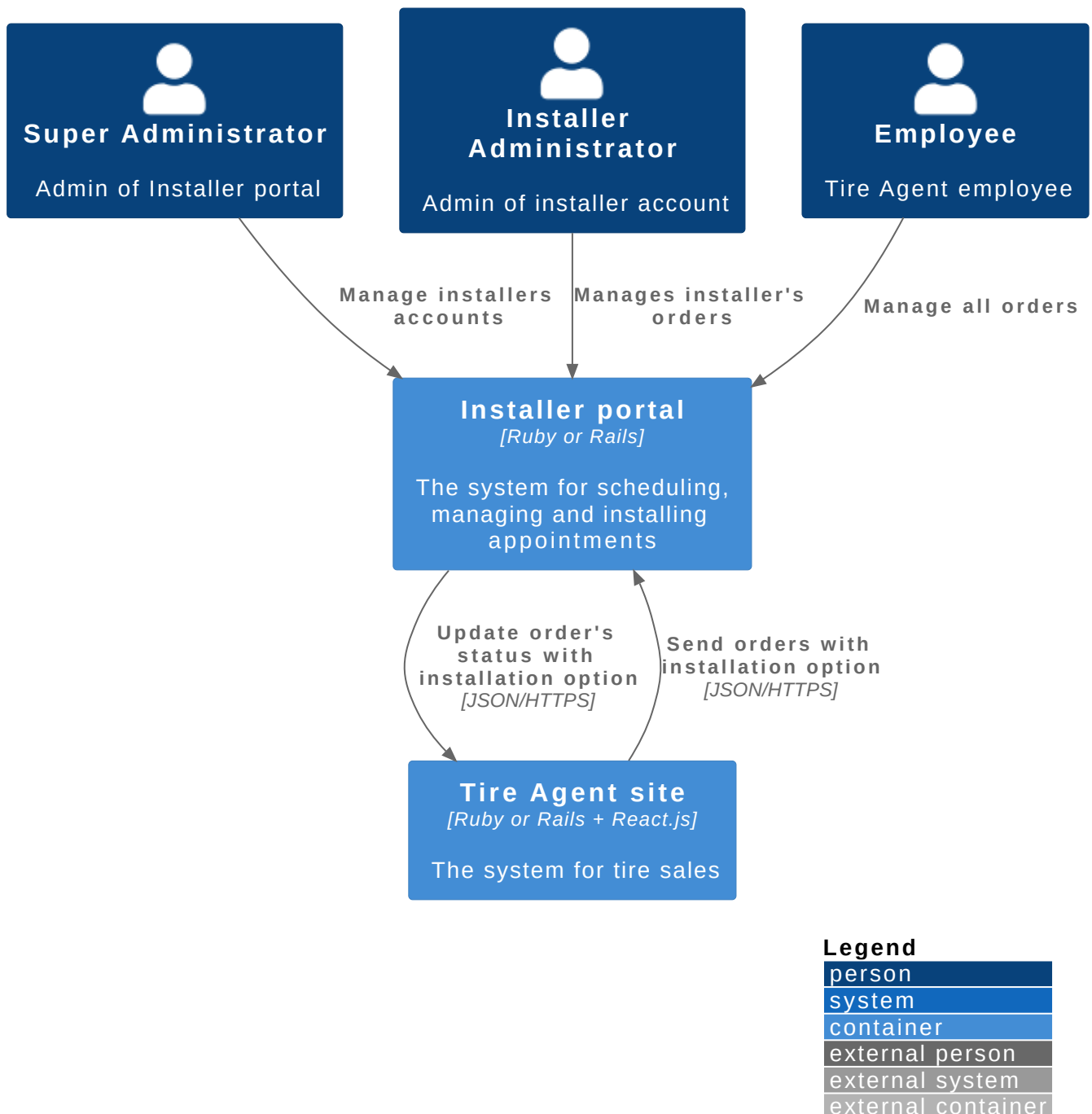
1. On the Tire Agent side, the User clicks on payment with PayPair. Tire Agent sends user info and cart info to PayPair.
2. Widget opens with phone input field
3. User enters phone number
4. Backend gets phone, generate OPT code and send this code to User by SMS.
5. Widget renders OPT input field.
6. User enters OPT code. Widget sends request to Backend.
7. Backend validate OTP code.
8. Widget renders Application Details form.
9. User enters Date of birth, Gross monthly income, How often do you get paid, info about paycheck dates and SNN/ITIN.
10. Frontend part adds to this data user info from Tire Agent and security keys (BlackBoxID, TevId, ThrtMetrixId, ect).
11. Backend builds payload and sends requests to active providers in [Parallel](#)
12. Each provider processes the request and returns a decision.
13. Backend processes responses from providers and returns this data to Widget.
14. Widget processes provider's offers and shows for User in user frindly format.
15. User checks all available offers, can compare and choose one of them.
16. Widget sends selected offer to Backend.
17. Backend sends request to the selected provider.
18. Provider returns payment URL.
19. Backend sends this URL to Widget.
20. Widget puts this link in Iframe.
21. User fills needed data like card info, personal documents data (depends on provider).
22. Widget gets callback from provider that User successfully completed the contract signing process. Sends request to Backend.
23. Backend updates offer status.
24. Widget shows a message about the successful signing of the contract.

## 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](#)



## 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.

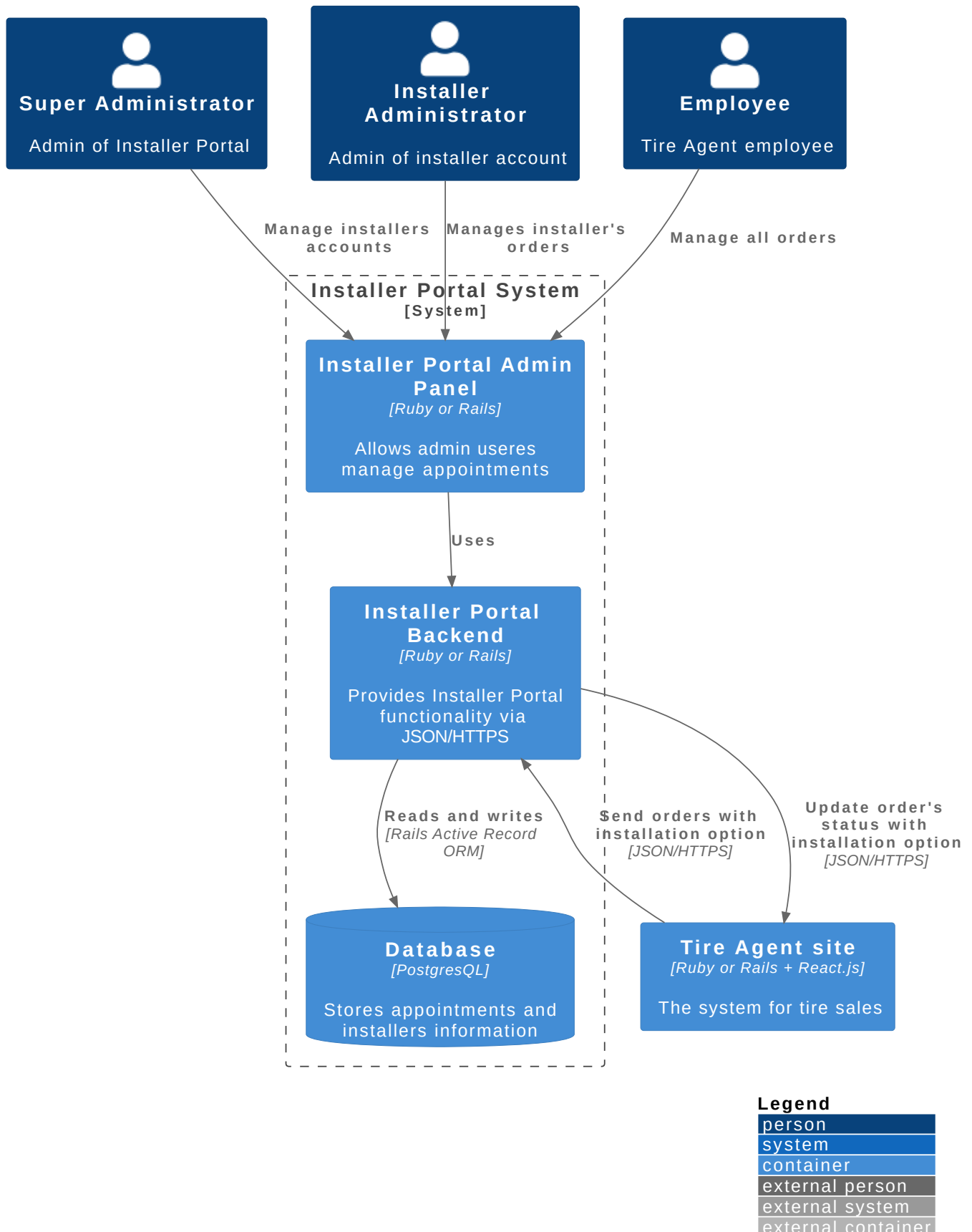
**Primary elements:** Containers within the software system in scope. Supporting elements: People and software systems directly connected to the containers.

**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



## 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.

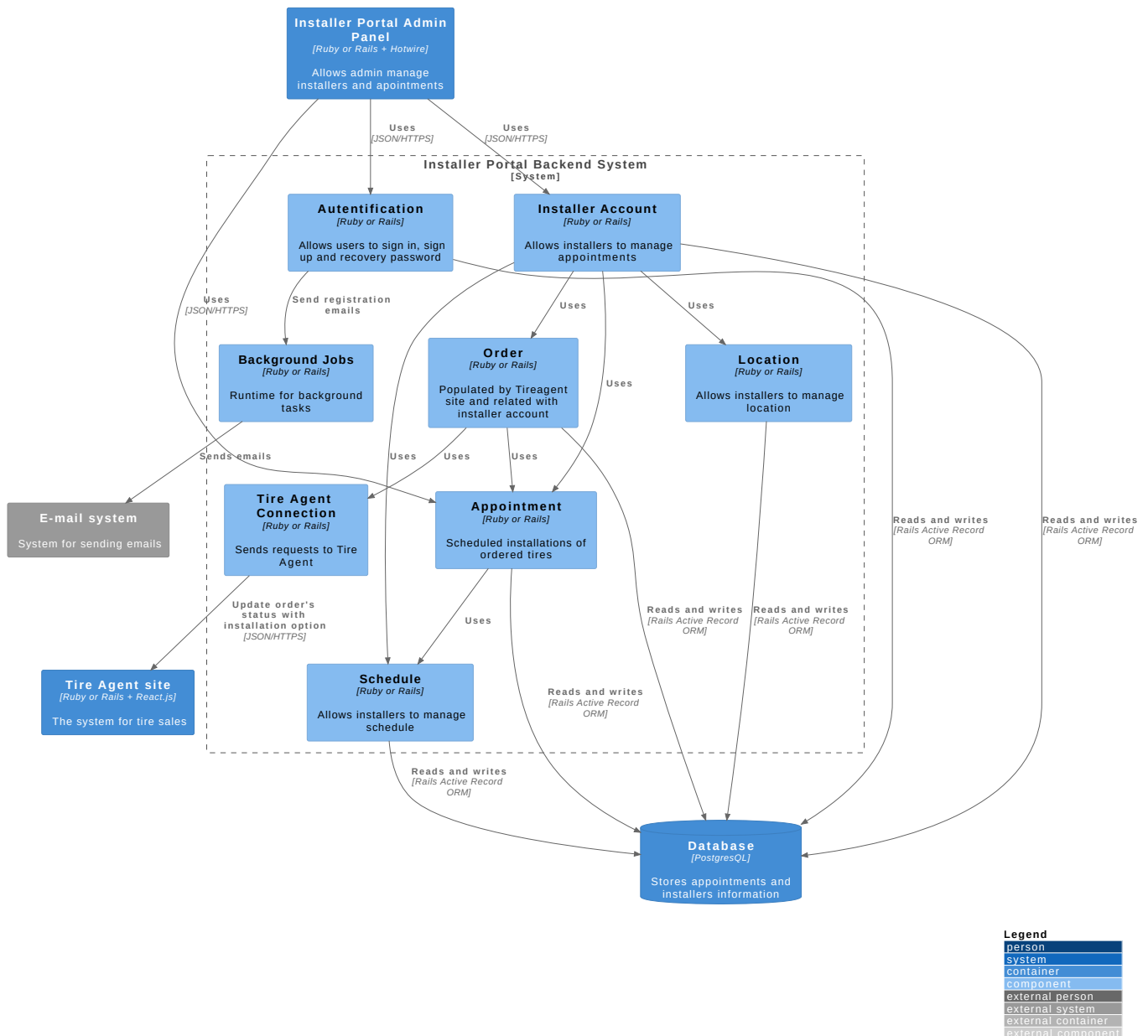
**Primary elements:** Containers within the software system in scope. Supporting elements: People and software systems directly connected to the containers.

**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.

## 1.4.4 Entity Diagram

[/1.4 Diagrams for TireAgent Installers Portal/1.4.4 Entity Diagram](#)

## 1.4.5 Activity Diagram

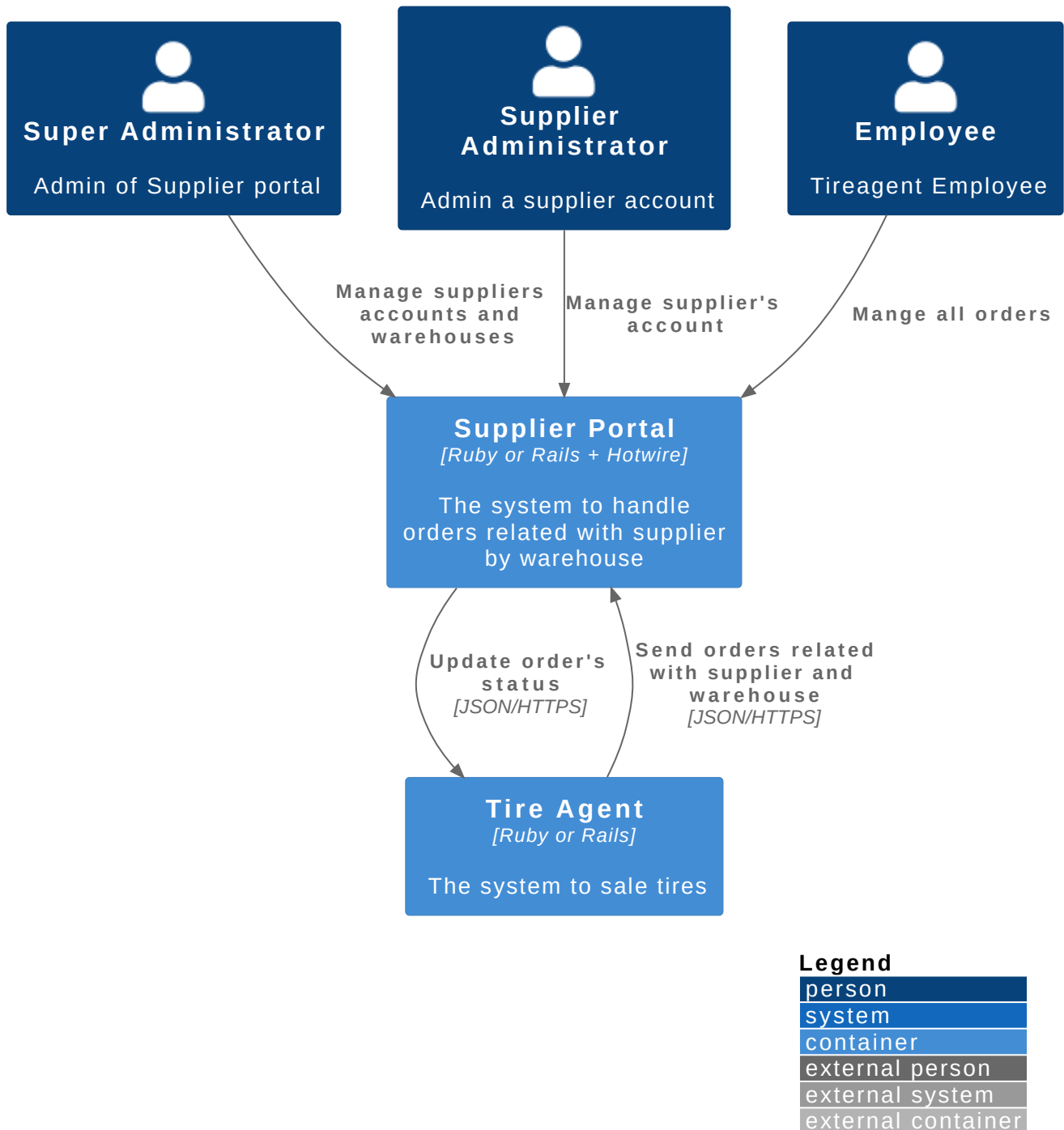
[/1.4 Diagrams for TireAgent Installers Portal/1.4.5 Activity Diagram](#)

## 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](#)



<<<<<<<< HEAD:docs/1.3 Diagrams for PayPair/1.3.4 Entity Diagram/HOME.md

## 1.3.4 Entity Diagram

Entity diagram describes entities in the system and relations between them.

Merchant. This entity represents merchant account. It is company which use PayPaid. Each merchant has personal agreements with creditors, settings.



## Level 2: Container diagram

master:src/1.6 Diagrams for TireAgent Supplier Portal/1.6.1  
Context Diagram/context.md

**Admin.** This entity represents an employee who manages customers (users), creditors, offers and approvals

**MerchantLpConfiguration.** This entity represents the provider setting and contains the personal credentials of each provider.

**Setting.** This entity represents merchant settings. This contains settings such as choosing an SMS service for sending a OTP code and setting up work with providers

**Subscriber**

**Trending.** This entity represents logs. This shows which screens user visit and where user close PayPair.

**User.** This entity represents the end-user in the system.

**NewUser.** This entity was creating for analytics. This also represents end-user but with needed data for analytics.

**ApplicationRequest.** This entity represents loan request from user to PayPair system.

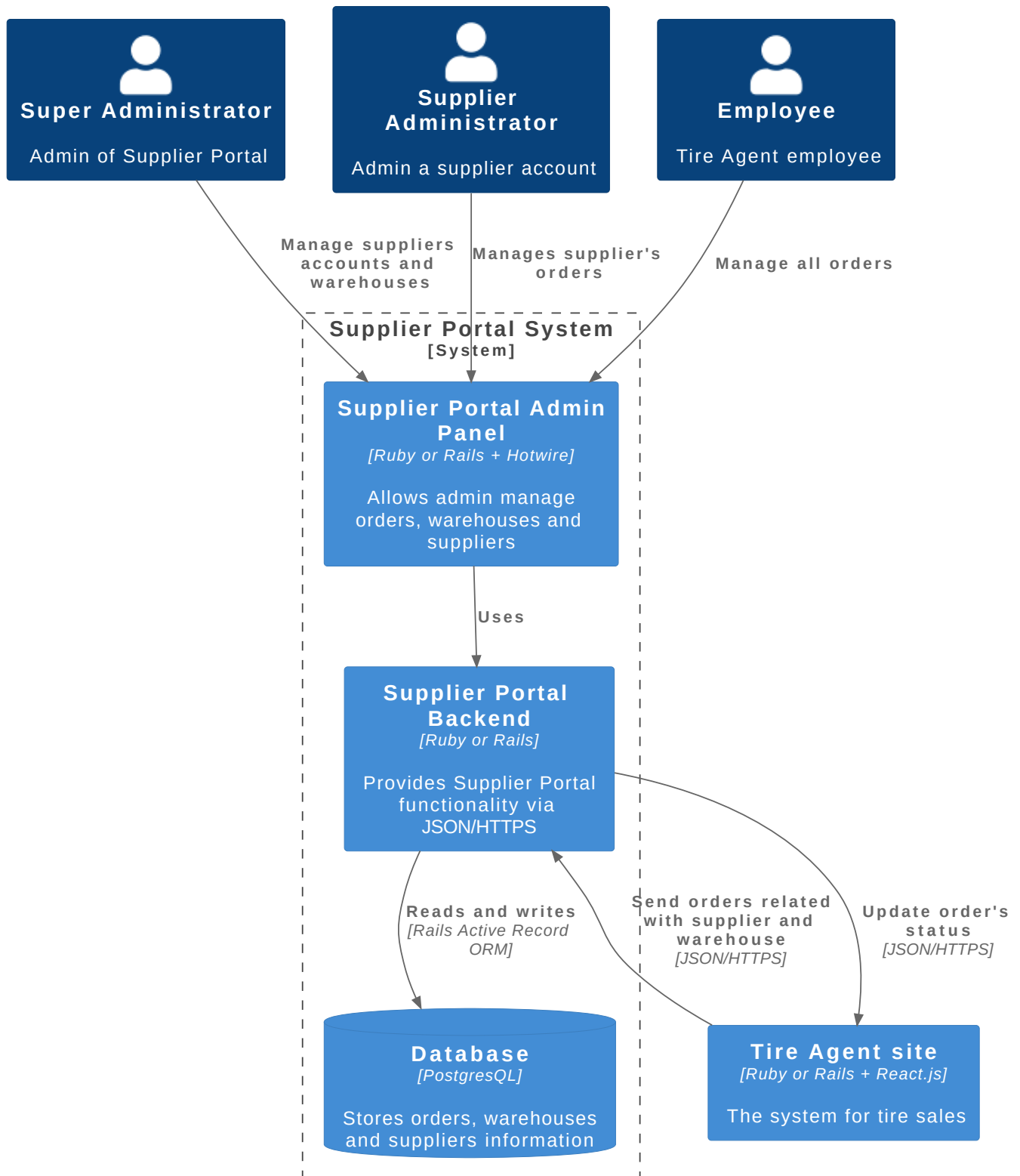
**LeaseApplication.** This entity represents responses from providers. It shows whether the provider approved the loan request or not, information about payment schedule. It is also updated if the user took advantage of the offer or if the user after a while did cancel, refund or settle.

**OrderItem.** This entity represents user's cart. It contains product information.

**RefundedItem.** This entity represents items which was refunded.

## 1.6.2 Containers Diagram

/1.6 Diagrams for TireAgent Supplier Portal/1.6.2 Containers Diagram



#### 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.

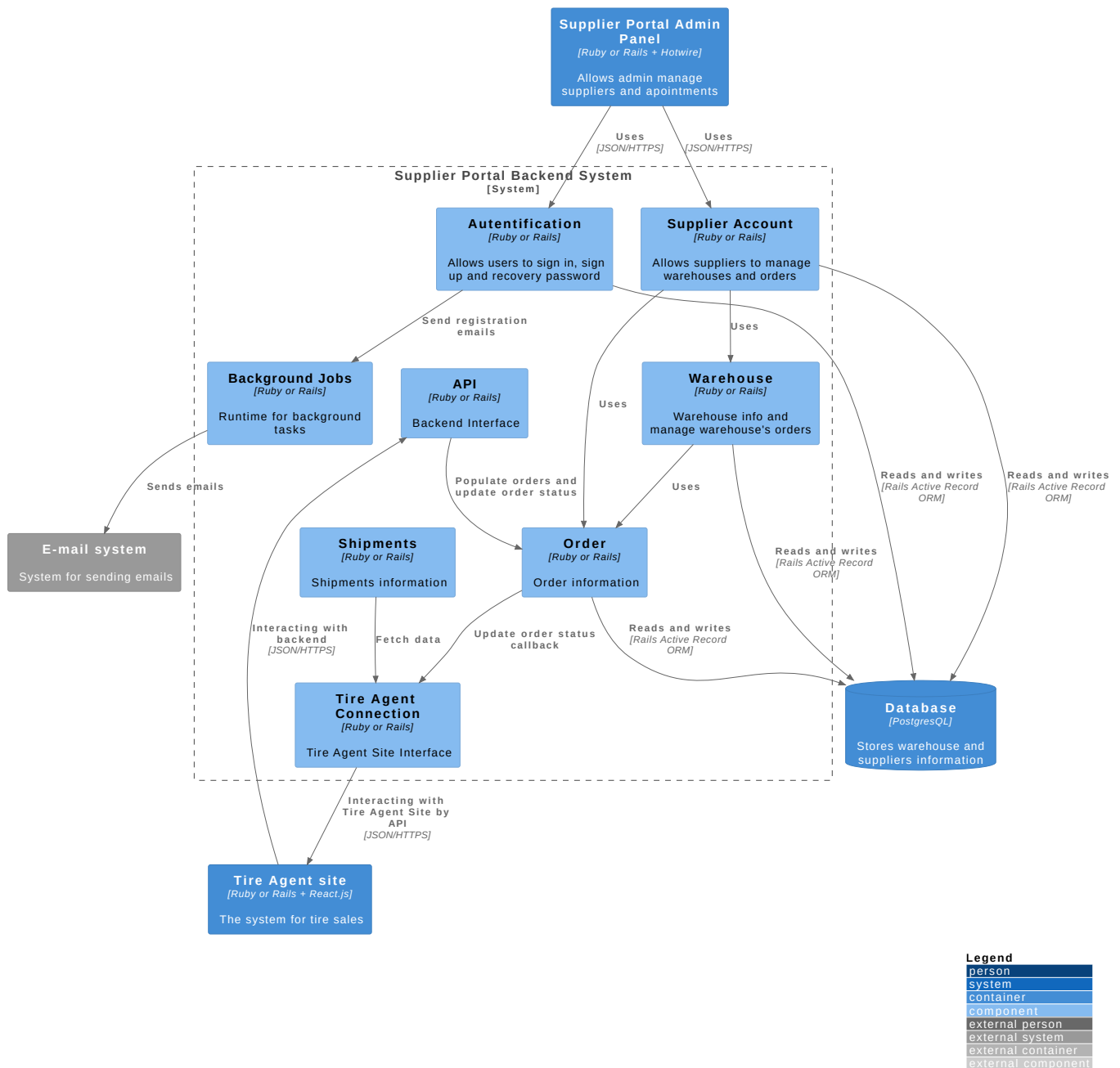
**Primary elements:** Containers within the software system in scope. Supporting elements: People and software systems directly connected to the containers.

**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.

## 1.6.3 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