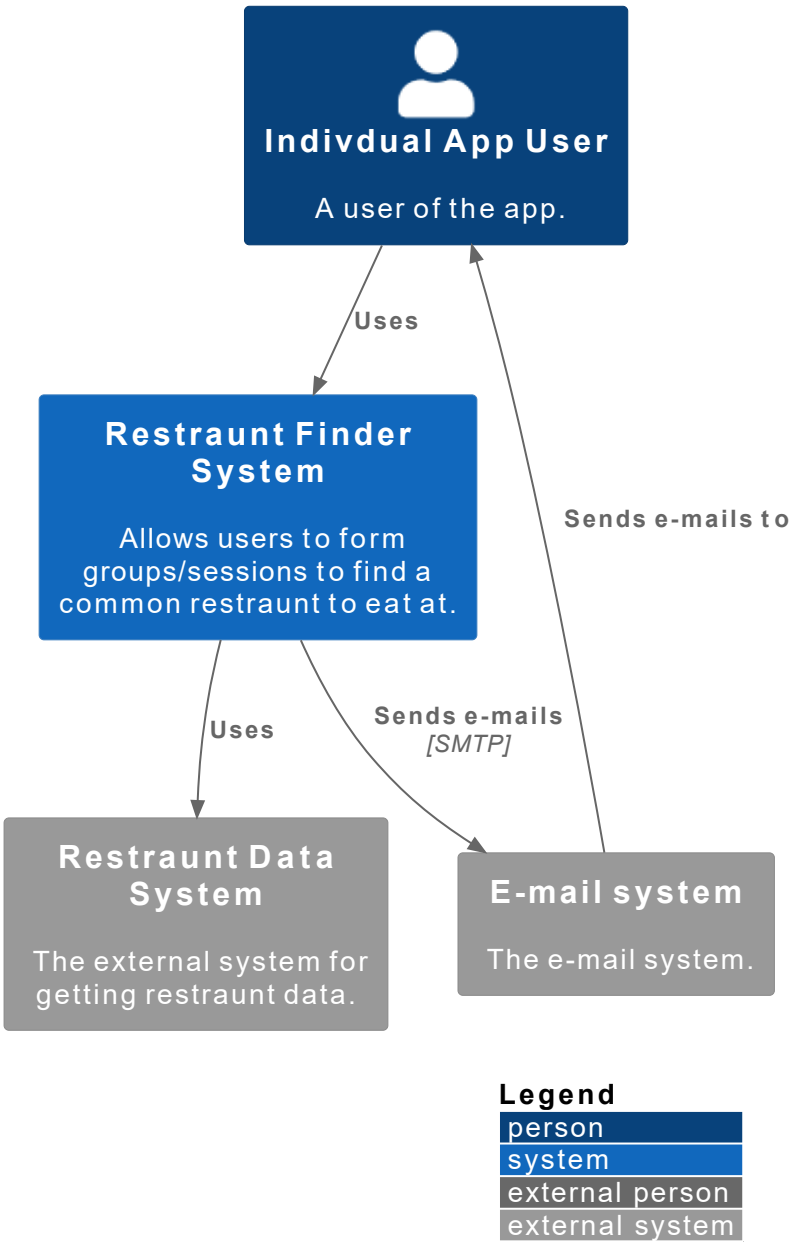
ForkFinderDiagram

Overview



Level 1: System Context Diagram

Context Scope

The scope of the system is a **restaurant finder app** that allows users to swipe through nearby restaurants with friends, selecting preferences and syncing their choices. The system includes a mobile app (frontend), a backend (microservices for authentication, sessions, etc.), and external integrations, such as an email system for notifications.

Primary Elements

The App (System in Scope)

- **Frontend**: The mobile app interface that users interact with. This includes features for registration, logging in, creating restaurant selection sessions, and interacting with other users.
- **Backend**: A series of microservices that handle authentication, session management, restaurant selection logic, and user preferences.

People (Actors/Roles/Personas)

- **Users**: People who use the app to find restaurants and sync sessions with friends. These users interact with both the frontend (to view and swipe restaurants) and the backend (for login, session creation, and management).

Software Systems (External Dependencies)

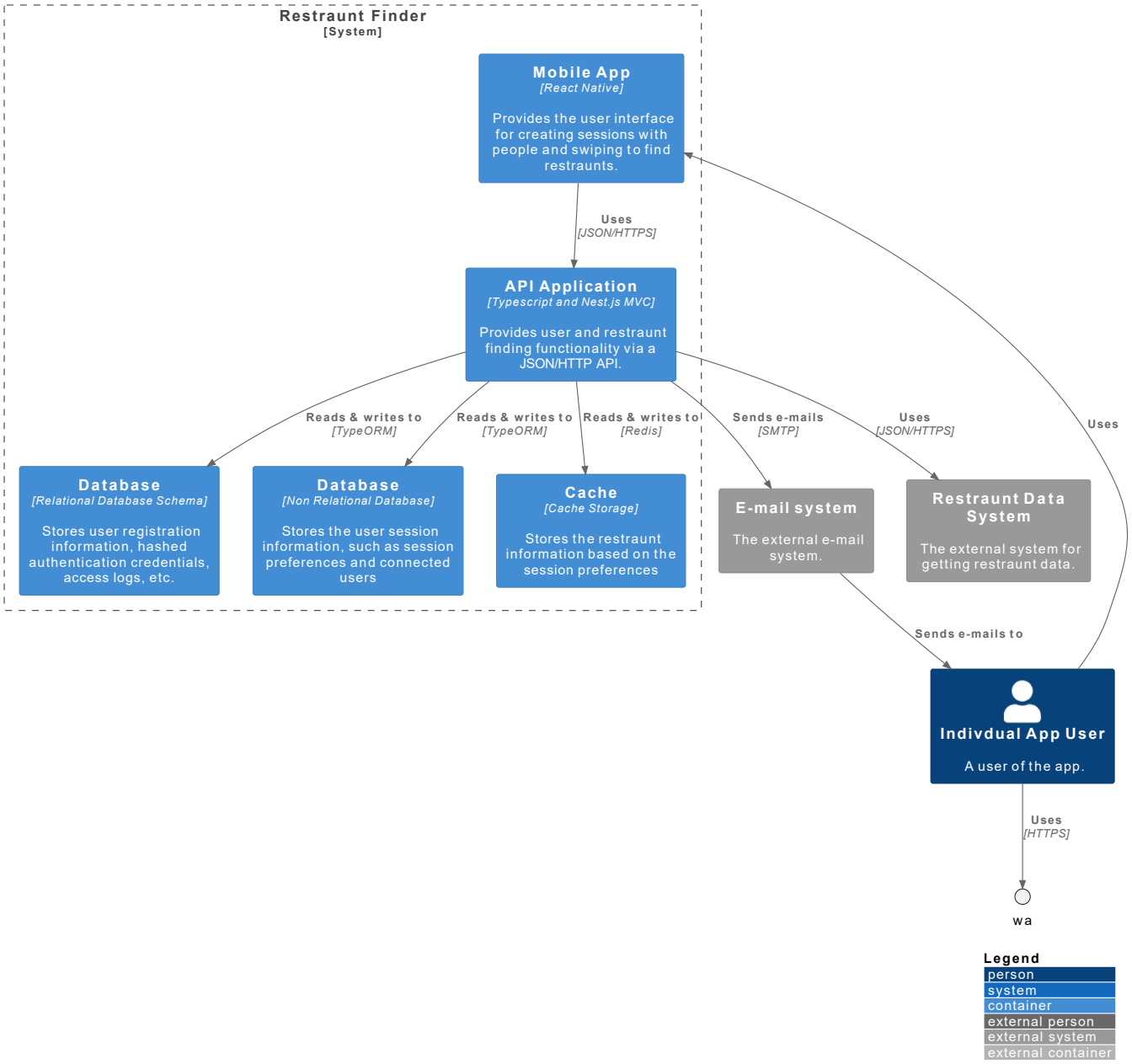
- **Email System**: The external system that handles sending emails for notifications, such as confirming account registration, password recovery, or session-related notifications. This could be an external service like AWS SES, SendGrid, or the Gmail API.
- **Restaurant Data API**: A third-party system (Google Places) that the app uses to fetch restaurant data for users to swipe through.

Intended Audience

This diagram is aimed at both **technical and non-technical stakeholders**. It provides a high-level view of how the restaurant finder app interacts with its users and external systems, allowing everyone, including managers, developers, designers, and external partners, to understand the relationships between the app's components without getting into technical details.

1 Fork Finder System

```
\1 Fork Finder System
```



Level 2: Container Diagram

Context Scope

This diagram outlines the structure and major components of the **Restaurant Finder App**. It provides a high-level view of the system's containers (frontend, backend, database, etc.), external systems it interacts with (email, restaurant data), and how these components communicate with one another.

Containers and Relationships

People

- **Individual App User (IAU)**: A person who uses the app to find restaurants and manage sessions.

External Systems

- **E-mail System (ES)**: An external system that sends notifications and emails to users (e.g., for registration confirmation, session updates).
- **Restaurant Data System (RDS)**: An external system used to fetch restaurant data based on user preferences and location (e.g., Yelp API, Google Places).

Internal Containers

- **Mobile App (MA)**:
    - **Technology**: React Native
    - **Description**: The mobile app provides the user interface for creating sessions, inviting friends, and swiping to find restaurants.
- **Database (Relational) (DB)**:
    - **Technology**: Relational Database Schema
    - **Description**: Stores persistent user information, such as registration data, hashed authentication credentials, and access logs.
- **Non-Relational Database (NRDB)**:
    - **Technology**: NoSQL (Non-Relational) Database
    - **Description**: Stores user session data, including session preferences, active sessions, and connected users.
- **Cache Storage (CACHE)**:
    - **Technology**: Redis
    - **Description**: Temporary storage for restaurant data based on session preferences to reduce repeated API calls to external systems.
- **API Application (API)**:
    - **Technology**: Typescript, Nest.js MVC
    - **Description**: Handles core functionality including user authentication, restaurant data retrieval, and session management. Provides this functionality via a JSON/HTTP API.

Container Communication

- **API ↔ E-mail System**: Sends emails through the external email system via SMTP.
- **API ↔ Restaurant Data System**: Fetches restaurant data from the external system using JSON/HTTPS requests.
- **API ↔ Relational Database**: Reads and writes user and session data to the relational database using TypeORM.
- **API ↔ Non-Relational Database**: Reads and writes session-specific data to the NoSQL database using TypeORM.
- **API ↔ Cache Storage**: Interacts with Redis to cache restaurant data for faster access.
- **Mobile App ↔ API**: Communicates with the API to authenticate users, fetch restaurant data, and manage sessions via JSON/HTTPS requests.

People Communication

- **Individual App User** ↔ **Mobile App**: Users interact with the mobile app over HTTPS to perform various actions.
- **E-mail System** ↔ **Individual App User**: The external email system sends emails to the user (e.g., for registration confirmation or session updates).

Relationships

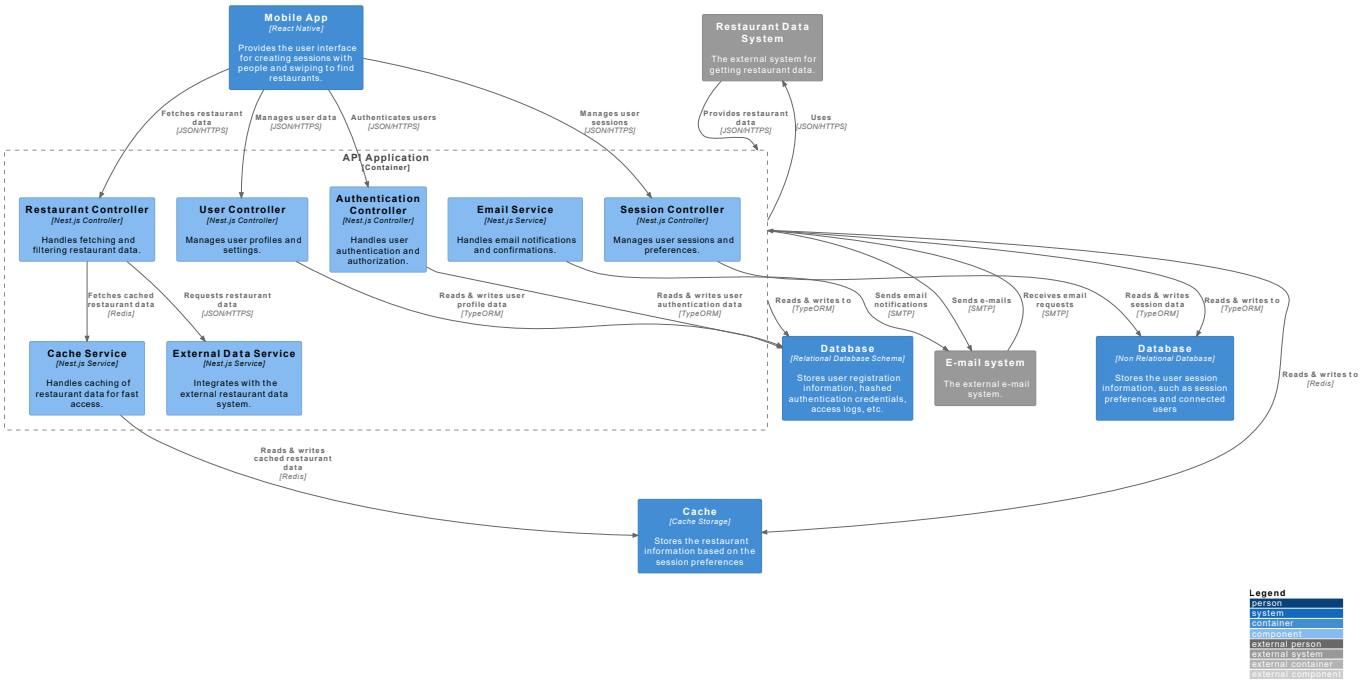
- **MA** → **API**: The mobile app communicates with the API using JSON/HTTPS to retrieve restaurant data and manage sessions.
- **API** → **ES**: The API communicates with the email system over SMTP to send emails to users.
- **API** → **RDS**: The API fetches restaurant data from external sources via JSON/HTTPS.
- **API** → **DB**: The API reads from and writes to the relational database using TypeORM.
- **API** → **NRDB**: The API interacts with the NoSQL database to store and retrieve session information.
- **API** → **CACHE**: The API caches restaurant data in Redis based on session preferences to avoid redundant external API calls.

Intended Audience

This diagram is intended for **technical staff**, including software architects, developers, and operations/support personnel. It provides an overview of how the containers within the system are structured and communicate with each other.

API Application

`\1 Fork Finder System\API Application`



Level 3 Component Diagram: API Application

Overview

This component diagram decomposes the **API Application** container, illustrating its internal components, their responsibilities, technologies used, and interactions. The purpose of this diagram is to provide software architects and developers with an understanding of the internal architecture of the API container, and how it connects to other containers and external systems.

Components in Scope:

1. **Authentication Controller**
2. **User Controller**
3. **Session Controller**
4. **Restaurant Controller**
5. **Email Service**
6. **External Data Service**
7. **Cache Service**

Supporting Elements:

- **Relational Database (DB)**
- **Non-Relational Database (NRDB)**
- **Cache Storage (Cache)**
- **External E-mail System (E-mail)**
- **Restaurant Data System (RDS)**

Components

1. **Authentication Controller**

- **Technology:** Nest.js Controller
- **Responsibilities:**
  - Manages user authentication and authorization.
  - Verifies user credentials and issues tokens for secure access to the system.
- **Interactions:**
  - **Reads and writes** authentication data from/to the **Relational Database** (DB) using **TypeORM**.
  - Interacts with **Mobile App** (ma) to authenticate users via **JSON/HTTPS**.
  - Sends authentication tokens to the **Mobile App**.

2. **User Controller**

- **Technology:** Nest.js Controller
- **Responsibilities:**
  - Manages user profile data, such as registration, profile updates, and settings.
- **Interactions:**
  - **Reads and writes** user profile data from/to the **Relational Database** (DB) using **TypeORM**.
  - Communicates with the **Mobile App** (ma) to manage user data via **JSON/HTTPS**.

3. **Session Controller**

- **Technology:** Nest.js Controller

- **Responsibilities:**
  - Manages user sessions, including session creation, session preferences, and user connections.
- **Interactions:**
  - **Reads and writes** session data from/to the **Non-Relational Database** (NRDB) using **TypeORM**.
  - Provides session information to the **Mobile App** (ma) through **JSON/HTTPS**.

4. **Restaurant Controller**

- **Technology:** Nest.js Controller
- **Responsibilities:**
  - Handles the fetching, filtering, and presentation of restaurant data.
- **Interactions:**
  - **Fetches cached restaurant data** from the **Cache Service** (Redis) for faster access.
  - **Requests new restaurant data** from the **External Data Service** (RDS) via **JSON/HTTPS**.
  - Communicates with the **Mobile App** (ma) to display restaurant information via **JSON/HTTPS**.

5. **Email Service**

- **Technology:** Nest.js Service
- **Responsibilities:**
  - Sends email notifications and confirmations to users, such as registration and session updates.
- **Interactions:**
  - **Sends email requests** to the external **E-mail System** (SMTP).
  - Sends email notifications to users as needed.

6. **External Data Service**

- **Technology:** Nest.js Service
- **Responsibilities:**
  - Integrates with external systems to fetch up-to-date restaurant data.
- **Interactions:**
  - **Requests restaurant data** from the **Restaurant Data System** (RDS) via **JSON/HTTPS**.
  - Sends the fetched data to the **Restaurant Controller** for further processing.

7. **Cache Service**

- **Technology:** Nest.js Service
- **Responsibilities:**
  - Manages caching of restaurant data for fast retrieval, ensuring that restaurant-related requests are handled quickly.
- **Interactions:**
  - **Reads and writes** cached restaurant data from/to the **Cache Storage** (Redis).
  - Communicates with the **Restaurant Controller** to provide cached restaurant data when requested.

Relationships

Internal Component Interactions:

- The **Authentication Controller** interacts with the **Relational Database** for user authentication data and communicates with the **Mobile App** for user authentication.
- The **User Controller** and **Session Controller** interact with the **Relational Database** and **Non-Relational Database** for profile and session management, respectively.
- The **Restaurant Controller** uses the **Cache Service** for fast data retrieval and fetches restaurant data from the **External Data Service** when needed.
- The **Email Service** sends email notifications to users via the **E-mail System** (SMTP).
- The **External Data Service** fetches data from the **Restaurant Data System** (RDS) and sends the data to the **Restaurant Controller** for further processing.

External Interactions:

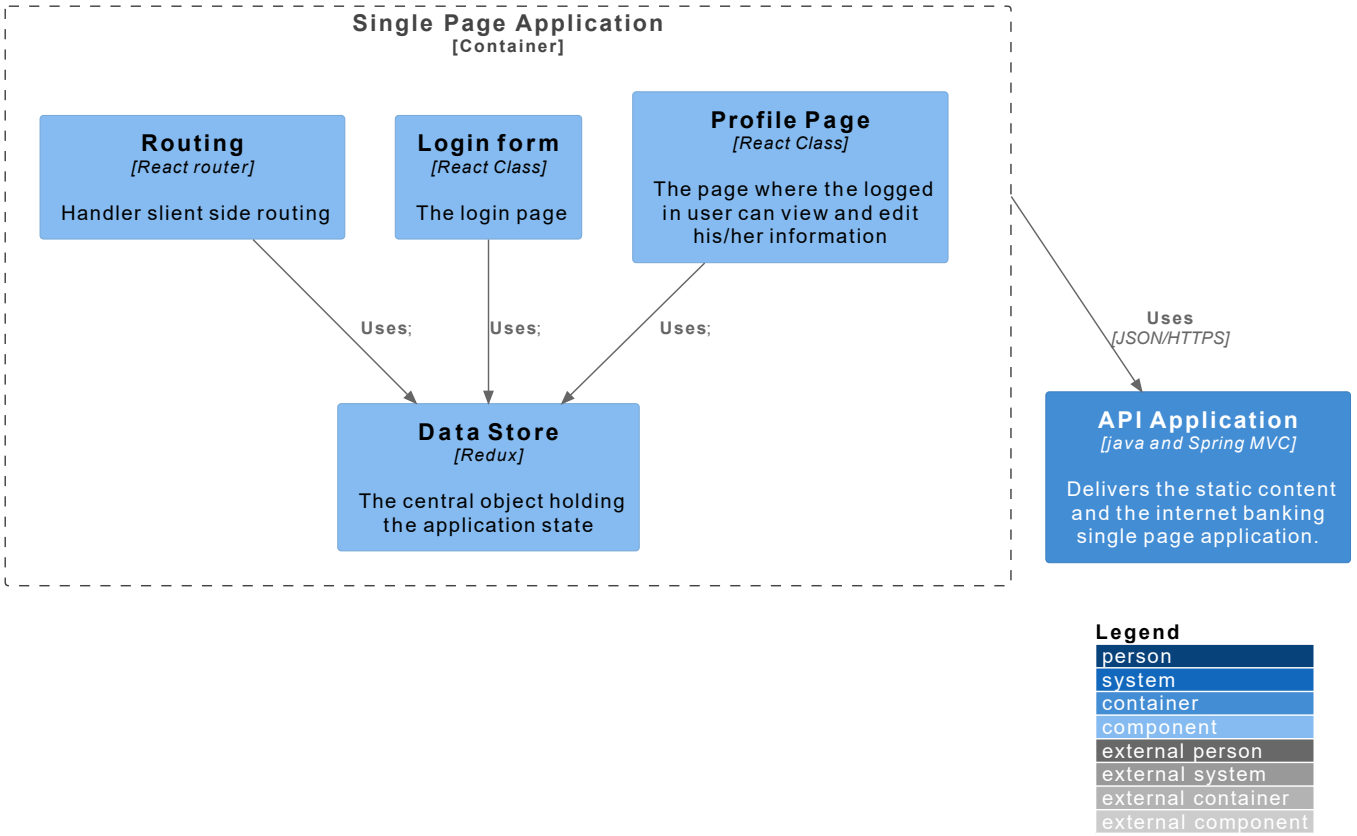
- The **API Application** communicates with the **E-mail System** and **Restaurant Data System** via **JSON/HTTPS** for restaurant data and email notifications.
- The **API** interacts with the **Cache Storage** (Redis) and **Databases** (Relational and Non-Relational) for data persistence and fast access.

Conclusion

This **Level 3: Component Diagram** provides a detailed view of the **API Application** container's internal components, their responsibilities, and how they interact with each other and external systems. This breakdown aids software architects and developers in understanding the modular structure of the API container and its connection to the broader system architecture.

Single Page Application

`\1 Fork Finder 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.

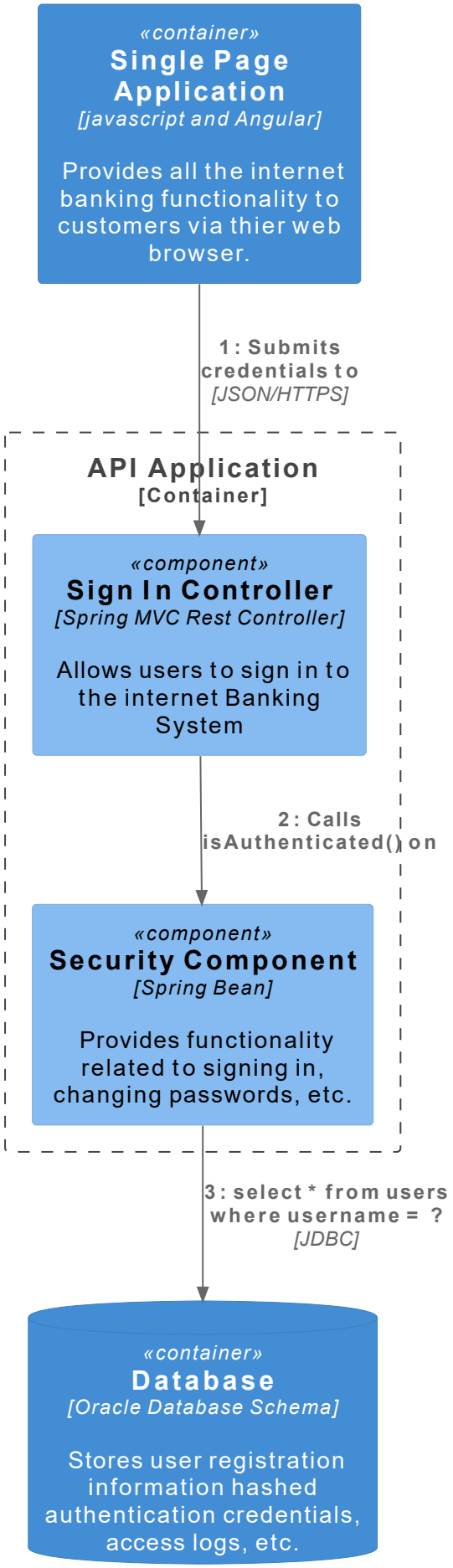
Primary elements: Components within the container in scope. Supporting elements: Containers (within the software system in scope) plus people and software systems directly connected to the components.

Intended audience: Software architects and developers.

> Example of included local image

Dynamic Diagram

`\1 Fork Finder System\Single Page Application\Dynamic Diagram`



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.

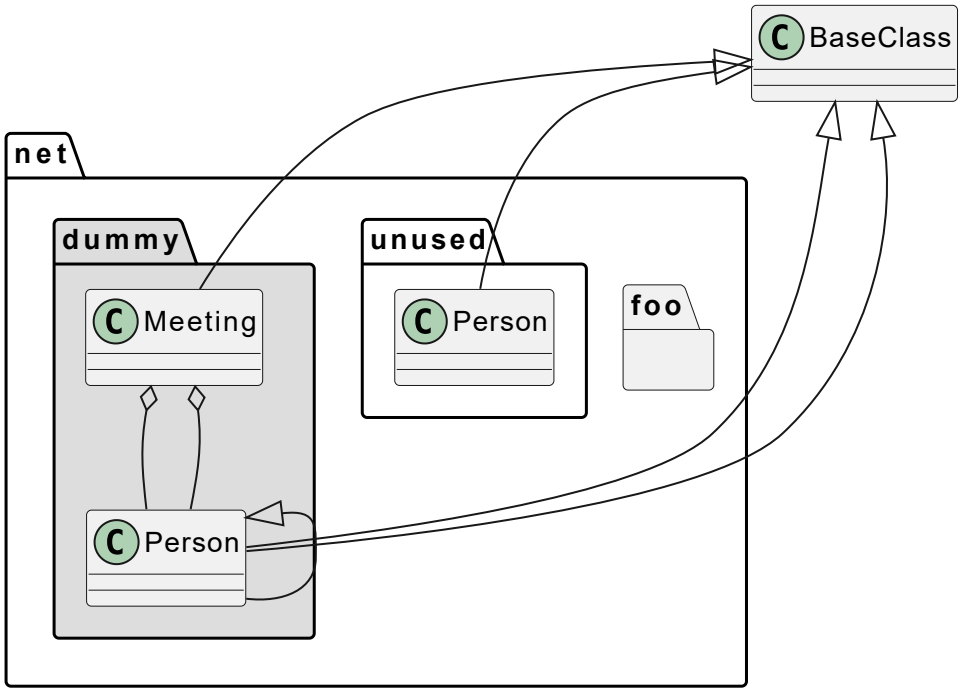
Scope: An enterprise, software system or container.

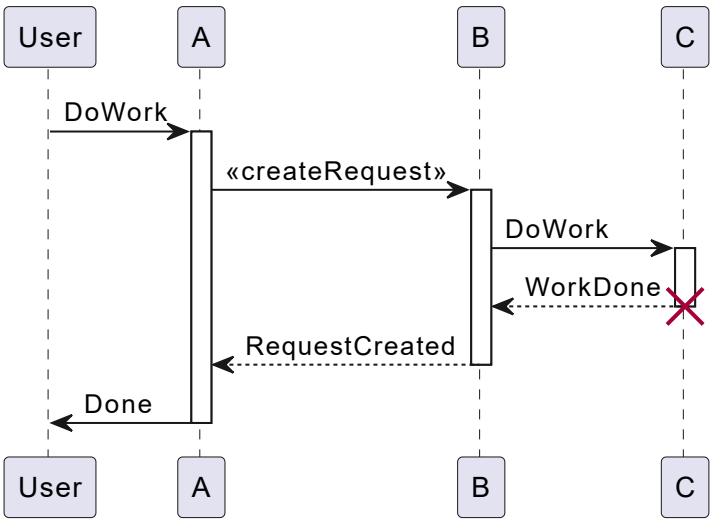
Primary and supporting elements: Depends on the diagram scope; enterprise (see System Landscape diagram), software system (see System Context or Container diagrams), container (see Component diagram).

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

Extended Docs

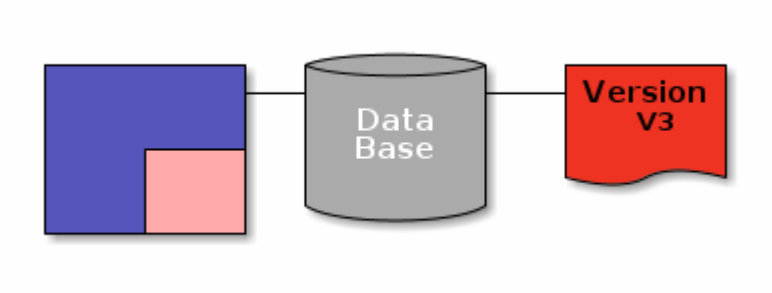
`\1 Fork Finder System\Single Page Application\Extended Docs`





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

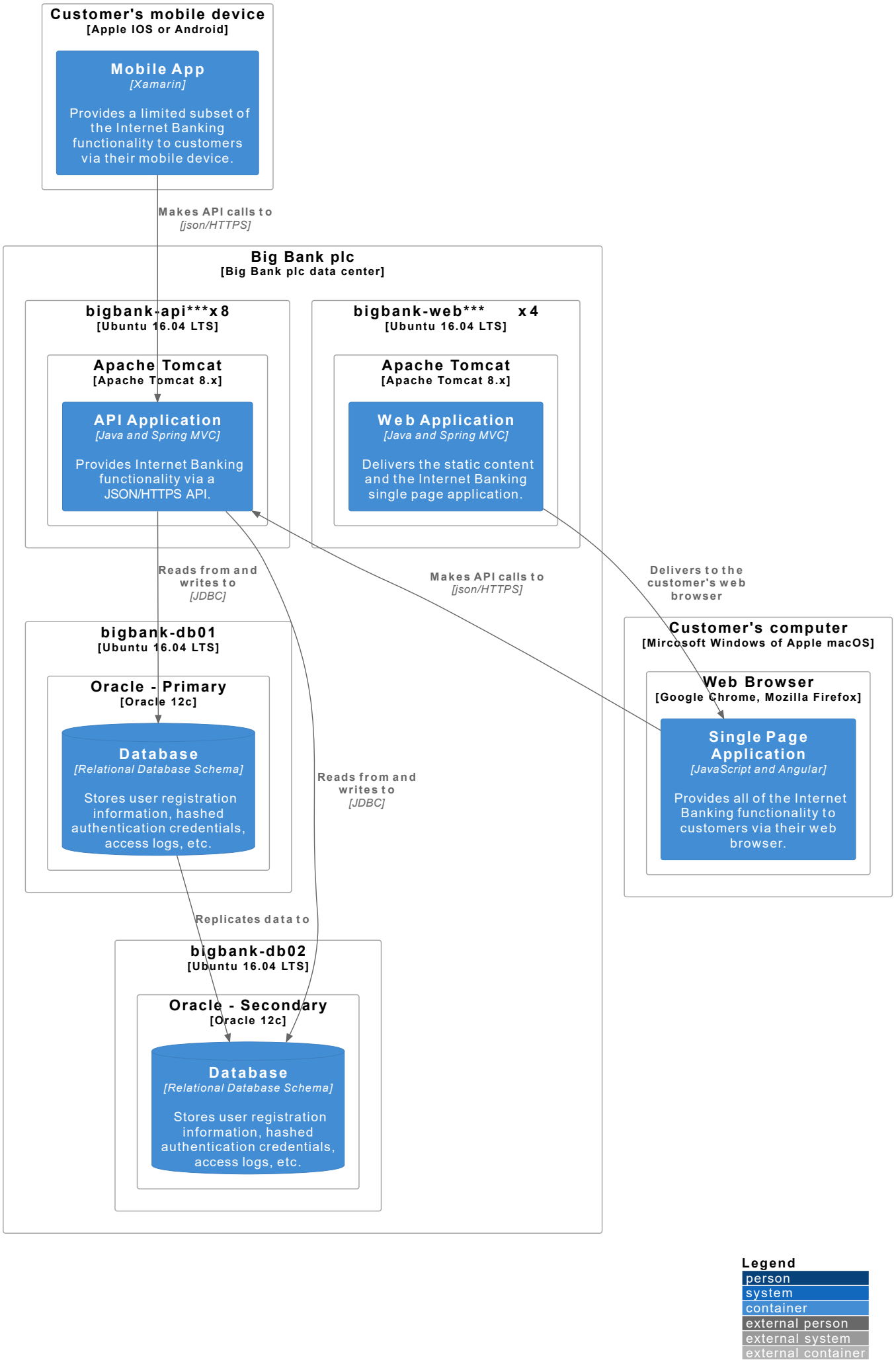
You can choose where to place a certain diagram by using `![name](<diagram name>.puml)`



Feel free to add any additional details necesary.

2 Deployment

`\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
bigbank-db01
[Ubuntu 16.04 LTS]
Oracle - Primary
[Oracle 12c]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
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.
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
external container

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.