CIRI2 Architecture Document

Jordy Walraven

Version history:

Version	Author(s)	Description	Date
1	Jordy Walraven	Setup Initial architecture	27-3-2024

Distribution history:

Version	Distributed to	Comments	Date

Table of contents

Context 2

Application architectural styles 3

C4 Architecture 3

Level1 3

Level 2 5

Actors and Their Interactions 5

PC Gamer 5

Admin 5

Containers and Their Interactions 6

Single Page Application 6

API Gateway 6

Microservices 6

Databases 6

Diagram Description 7

Context

Ciri2 is a platform designed for PC gaming enthusiasts. At its core, Ciri2 serves as a hub where users can assess their system's capability to run specific games and benchmark their performance against others in the community. Here's an overview of its key functionalities:

Account Creation:

Users can easily create personalized accounts, unlocking access to a host of features tailored to their gaming needs.

PC Rig Configuration:

Within the application, users have the flexibility to construct and customize their PC rigs using intuitive part pickers for various components. This empowers them to tailor their systems to meet the demands of their favorite games.

Game Overview:

Through Ciri2, users gain access to a comprehensive overview of games, including key details such as system requirements, gameplay mechanics, and community-generated performance data. This empowers gamers to make informed decisions about which titles to explore based on their hardware capabilities.

FPS Submission:

Gamers can submit their frames per second (FPS) benchmarks for specific games and graphics presets, allowing them to share and compare their performance metrics with fellow users. This feature fosters a sense of community and healthy competition among players.

Application architectural styles

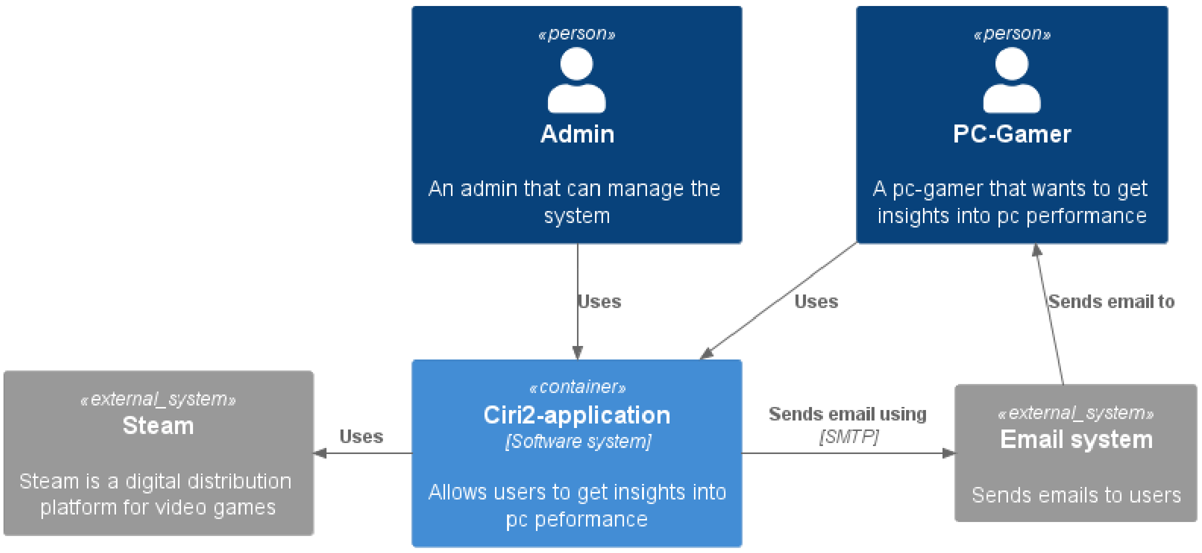
For CIRI2 we will be using the following architectural styles:

- Microservice pattern
- Layered pattern
- Client-Server pattern
- Event-Driven pattern
- Broker pattern
- Component-based pattern
- Service oriented architecture

Because our application is split into microservices, we need to keep this in mind when designing our architecture.

C4 Architecture

Level1


«person»
Admin
An admin that can manage the system
«person»
PC-Gamer
A pc-gamer that wants to get insights into pc performance
Uses
Uses
Sends email to
«external_system»
Steam
Steam is a digital distribution platform for video games
Uses
«container»
Ciri2-application
[Software system]
Allows users to get insights into pc peformance
Sends email using
[SMTP]
«external_system»
Email system
Sends emails to users

Overview

The System Context diagram for the Cirri2-Application provides a high-level overview of the system's interactions with its users and external systems. It serves as a starting point for understanding the system's environment and relationships.

Actors and Their Interactions

Admin

- Role: System Administrator
- Responsibilities:
  - Oversee the operations of the Cirri2-Application.
  - Ensure system performance and reliability.
- Interactions:
  - Utilizes the Cirri2-Application for management tasks.

PC-Gamer

- Role: End User
- Responsibilities:
    - Seeks insights into PC performance for an enhanced gaming experience.
- Interactions:
    - Engages with the Cirri2-Application to monitor and analyze PC performance.
    - Receives performance reports and notifications via the Email system.

Systems and Their Interactions

Ciri2-Application

- Description: A software system designed to provide insights into PC performance.
- Interactions:
    - Serves the Admin and PC-Gamer by providing relevant data and analytics.

Steam

- Description: An external digital distribution platform for video games.
- Interactions:
    - Integrated with the Cirri2-Application to offer seamless access to games and related performance metrics.

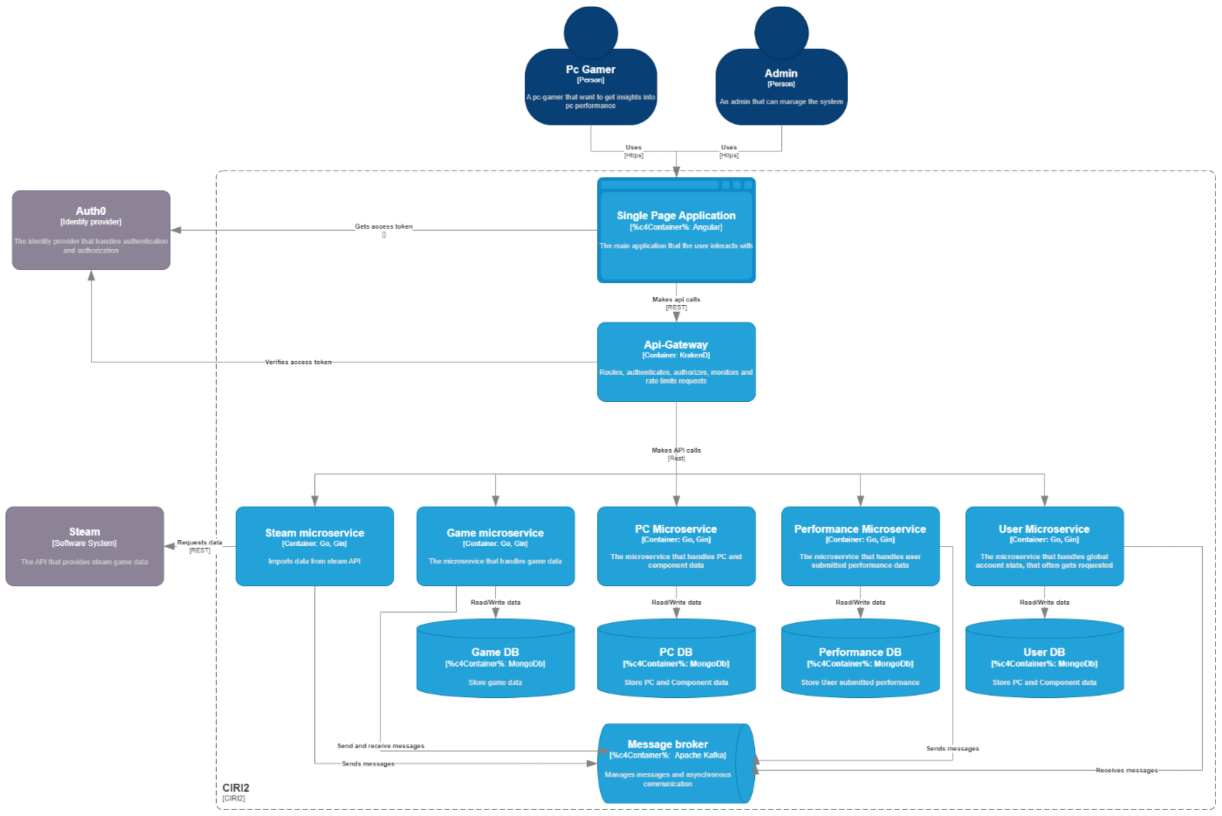
Email System

- Description: An external system responsible for delivering emails to users.
- Interactions:
    - Collaborates with the Cirri2-Application to send emails to PC-Gamers, enhancing communication and engagement.

Diagram Description

The diagram illustrates the Ciri2-Application at the center, surrounded by its primary users, the Admin and PC-Gamer, and external systems, Steam and the Email system. Arrows indicate the flow of interactions, emphasizing the system's role in connecting users with the services they require.

Level 2


Pc Gamer
[Person]
A pc-gamer that want to get insights into pc performance
Admin
[Person]
An admin that can manage the system
Uses
[Https]
Uses
[Https]
Auth0
[Identity provider]
The identity provider that handles authentication and authorization
Gets access token
Single Page Application
[%c4Container%: Angular]
The main application that the user interacts with
Makes api calls
[REST]
Api-Gateway
[Container: KrakenD]
Routes, authenticates, authorizes, monitors and rate limits requests
Verifies access token
Makes API calls
[Rest]
Steam
[Software System]
The API that provides steam game data
Requests data
[REST]
Steam microservice
[Container: Go, Gin]
Imports data from steam API
Game microservice
[Container: Go, Gin]
The microservice that handles game data
PC Microservice
[Container: Go, Gin]
The microservice that handles PC and component data
Performance Microservice
[Container: Go, Gin]
The microservice that handles user submitted performance data
User Microservice
[Container: Go, Gin]
The microservice that handles global account stats, that often gets requested
Read/Write data
Read/Write data
Read/Write data
Read/Write data
Game DB
[%c4Container%: MongoDb]
Store game data
PC DB
[%c4Container%: MongoDb]
Store PC and Component data
Performance DB
[%c4Container%: MongoDb]
Store User submitted performance
User DB
[%c4Container%: MongoDb]
Store PC and Component data
Send and receive messages
Sends messages
Message broker
[%c4Container%: Apache Kafka]
Manages messages and asynchronous communication
Sends messages
Receives messages
CIRI2
[CIRI2]

Overview

The Container diagram for the Ciri2-Application delves deeper into the system's architecture, revealing the high-level technology choices and how responsibilities are distributed among them.

Actors and Their Interactions

PC Gamer

- **Role**: End User
- **Responsibilities**:
  - Engage with the Ciri2-Application to get a list of PC-based games.
- **Interactions**:
  - Interacts with the Single Page Application to view and select games.

Admin

- **Role**: System Administrator
- **Responsibilities**:
  - Manage the game catalog within the Cirri2-Application.
- **Interactions**:
  - Uses the Single Page Application to add, remove, or edit games in the catalog.

Containers and Their Interactions

Single Page Application

- **Technology**: Angular Web Application
- **Responsibilities**:
  - Serve as the interface for both PC Gamer and Admin.
- **Interactions**:
  - Communicates with the API Gateway to process requests.

API Gateway

- **Responsibilities**:
  - Route requests to appropriate microservices.

Microservices

- **Steam Microservice**:
  - Fetches data from the Steam API.
  - Interacts with the Steam DB.
- **Game Microservice**:
  - Retrieves game data from the Game DB.
- **PC Microservice**:
  - Accesses PC data from the PC DB.
- **Performance Microservice**:
  - Gathers performance data from the Performance DB.
- **User Microservice**:
  - Manages user preferences in the User DB.

Databases

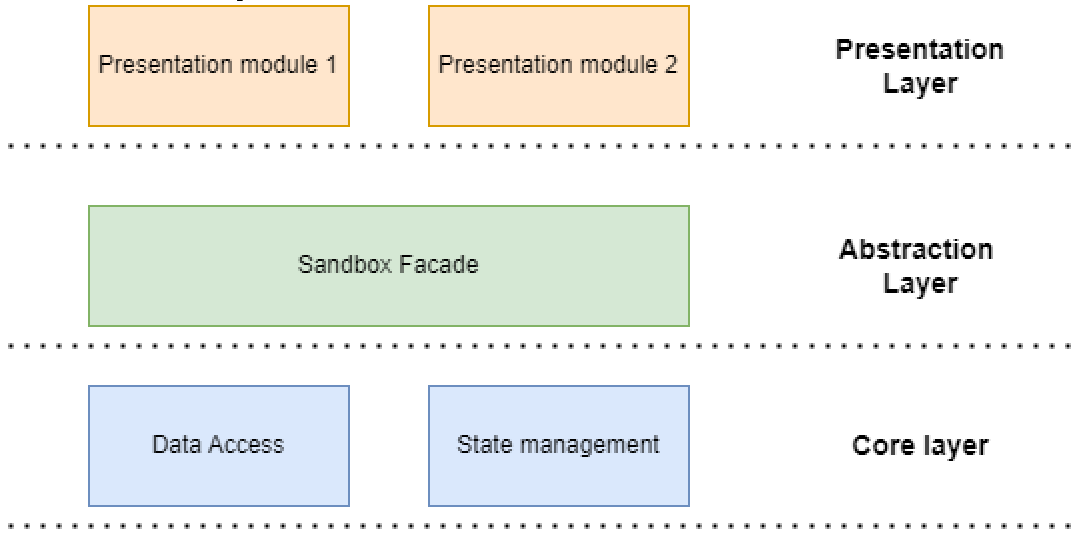
- **Steam DB**:
  - Stores information about Steam games.
- **Game DB**:
  - Contains game information.
- **PC DB**:
  - Holds PC information.
- **Performance DB**:
  - Keeps performance information.
- **User DB**:
  - Maintains user preferences.

Diagram Description

The Level 2 C4 diagram showcases the internal structure of the Ciri2-Application, including the Single Page Application, API Gateway, various microservices, their associated databases, and the message broker. It illustrates how the system is compartmentalized into containers, each with specific roles and interactions, to fulfill the needs of PC Gamers and Admins.

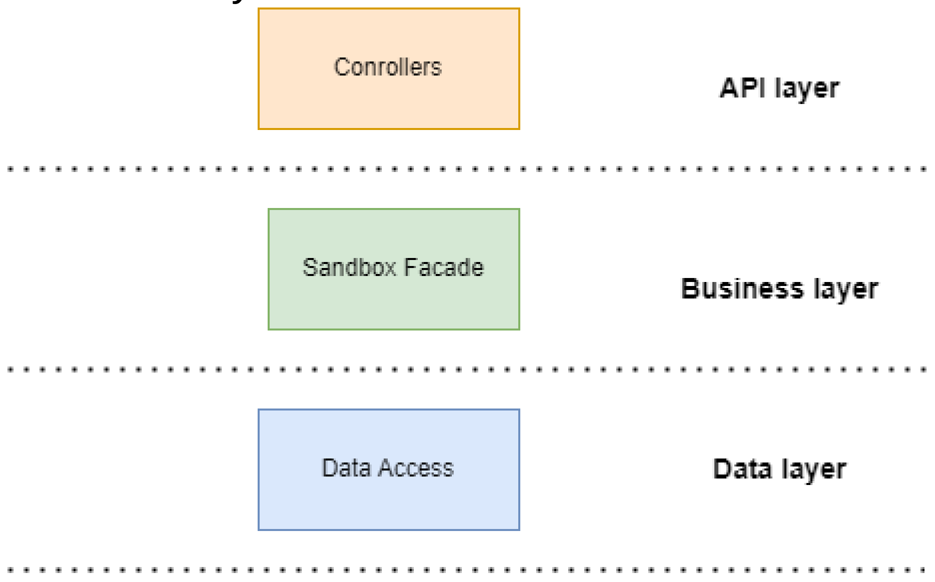
This documentation provides a detailed view of the containers within the Cirri2-Application's architecture, highlighting the technology choices and interaction flows.

Frontend Layers



The frontend will consist of 3 main layers. The layers the user interacts with is the presentation layer. The presentation layer never directly communicates with the core layer, and will always flow through an abstraction layer. In this application we will use sandboxes to keep them separated, this way our frontend doesn't know the implementation of our core, so that we can switch the logic if necessary without disturbing our presentation module. The abstraction layer is also nice for making sure that we hide details that are unnecessary for our presentation module. Then there is the core layer, the core layer handles our states, http requests and other logic. It gets called by the sandbox andnever directly interacts with the presentation module.

Backend Layers



The backend also has 3 layers, the first layer is the API layer, this is where requests come in and our controller will be called. The controller then calls the service, this service then does the logic that is necessary for the request, and after that it will call the repositories to update our database.