Forray Gabriel

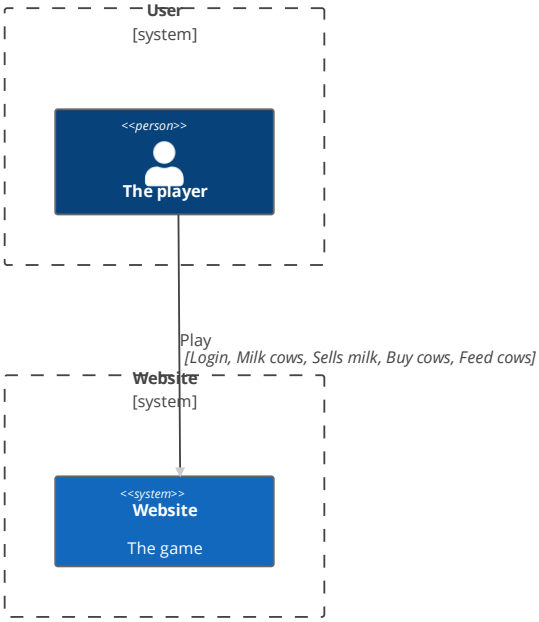
Erasmus

Service Oriented Architecture Project

System context

Theme of the project/System context

For this project, I wanted to do a simple game. The player owns a farm and has cows. There are two resources, money and milk. Cows can be milked every few minutes and give an amount of milk. The milk can then be sold to have money. Money allows the player to buy new cows or to feed the existing ones, making them increase their level. When fed, a cow will need more time between two milking, but will give more milk each time. The price of new cows increases with the number of cows and the cost to feed them increase with their level. In order to have a third-party integration, the game will also display the current weather in Cluj. To play, the username is “a” and the password is “a”.



Container diagram

For this project, the frontend use webpack shared context to share some components between different servers, and the backend is divided in microservices using NestJs.

The frontend is made in React on two servers. The first one, running on the port 3001, is the main one. If the user is not connected, it redirect them on a login page. Then the user can access the single page website that is the game. This page will get some informations from the backend server, and display different components. It will also get the weather from a third-party website using their api, to display it.

The second one, running on the port 3002, contains only one component, wich is a top bar. This component takes the money and milk amounts as parameters and integrate them in the bar. This component is then made available.

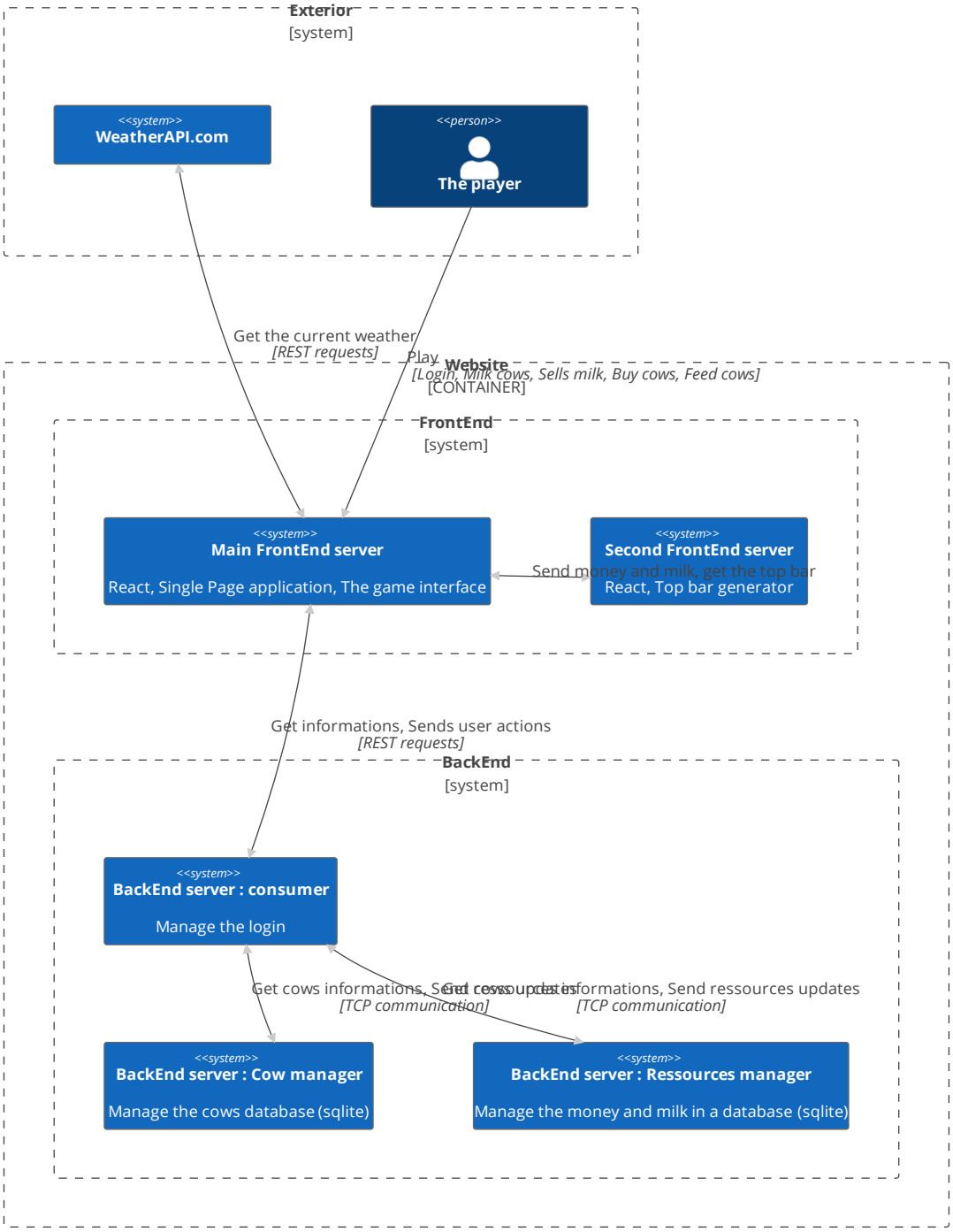
The first frontend server then give the money and milk amount to the second one, which return a topbar with these informations.

The first server of the backend is the consumer of the microservices. It is a web server that listen for rest requests on the port 3000. It is just a modified version of the example of a securised server. It's role is to authentificate the user and consume the data from the two providers. It communicate with them on the TCP layer.

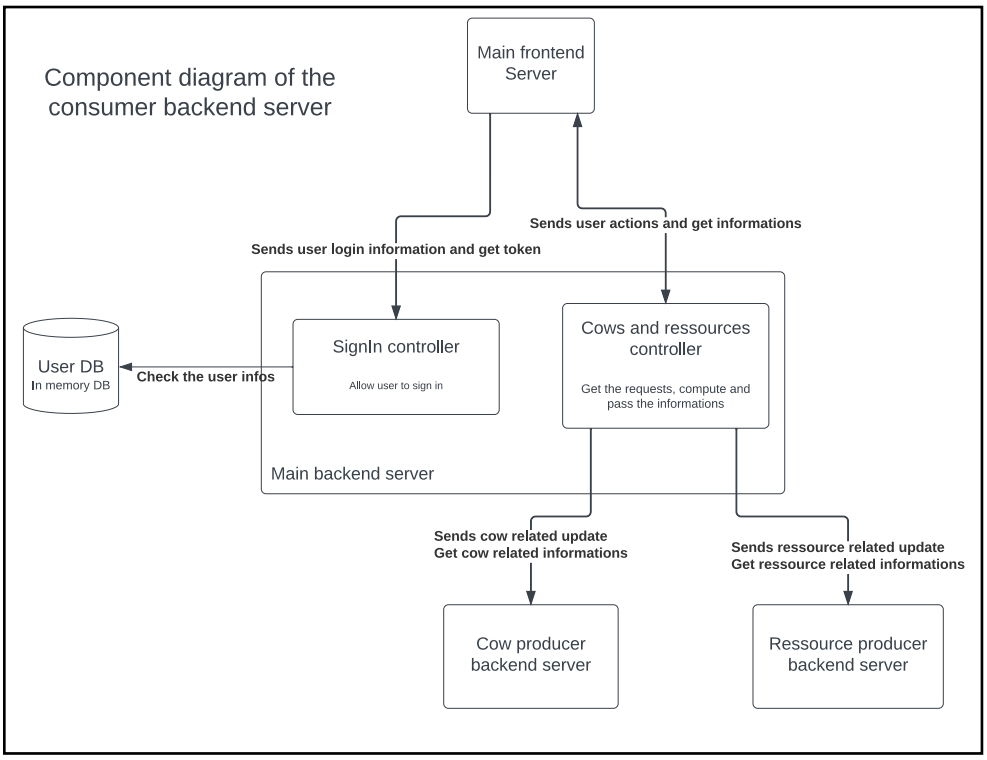
The first microservice provider is the one managing the ressources. It uses a sqlite database to store the money and milk amounts. It runs on the port 5001.

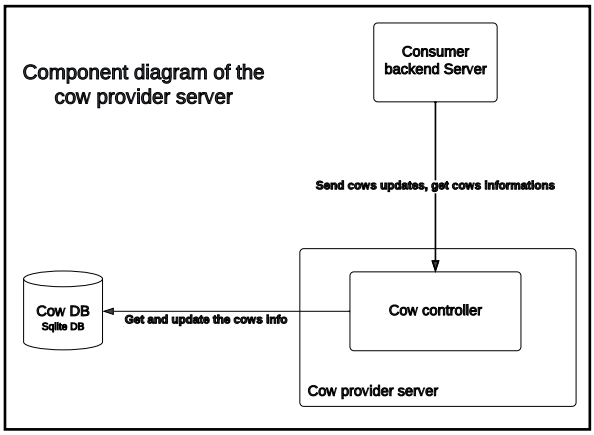
The second microservice provider is the one managing the cows. It also uses a sqlite database to store the differents informations about the cows. It runs on the port 5002.

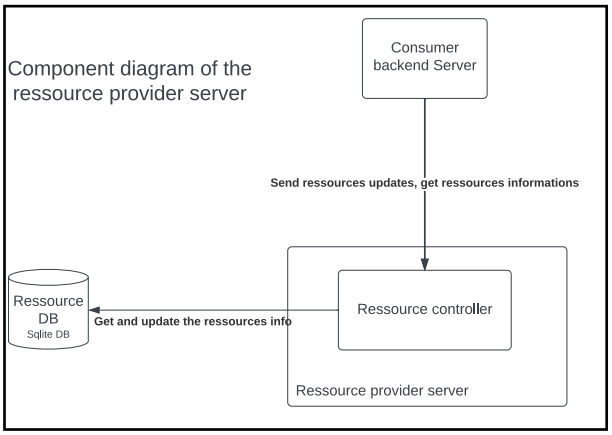
Container diagram

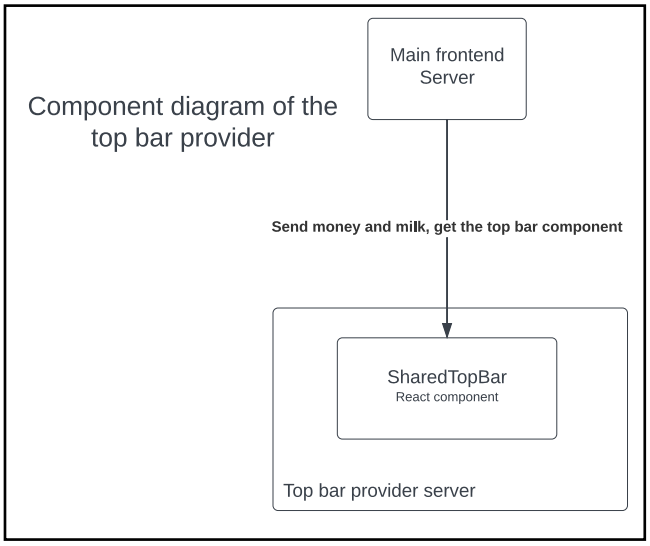


Component diagram



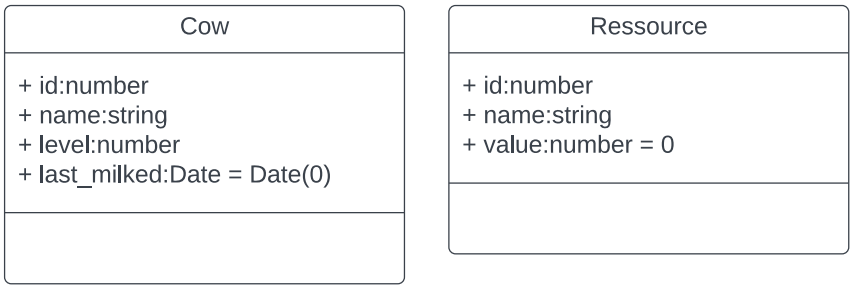






Cows and Resources classes

There are only two resources. The first one is money and the second one is milk.



Routes

Here is a summary of all the routes available on the backend consumer:

Method	Route	Parmeters	Description
POST	/api/auth/login	username password	Send the user credentials and get the token
GET	/money		Get the amount of money
PUT	/money	amount	Change the amount of money by the parameter amount
GET	/milk		Get the amount of milk
PUT	/milk	amount	Change the amount of milk by the parameter amount
GET	/sellMilk		Add the amount of milk times 10 to the money and set the milk to 0
GET	/getCows		Get a list of all the cows
POST	/buyCow	name	Create a new cow with the name and decrease the money
PUT	/milkCow	id	Get the cow, set the last milked date to now et increase the amount of milk
PUT	/addLevel	id	Get the cow, add a level, and decrease the money

Rules

The price of a new cow is equal to the number of cows times 10.

The price of leveling up a cow is equal to its current level squared times 100.

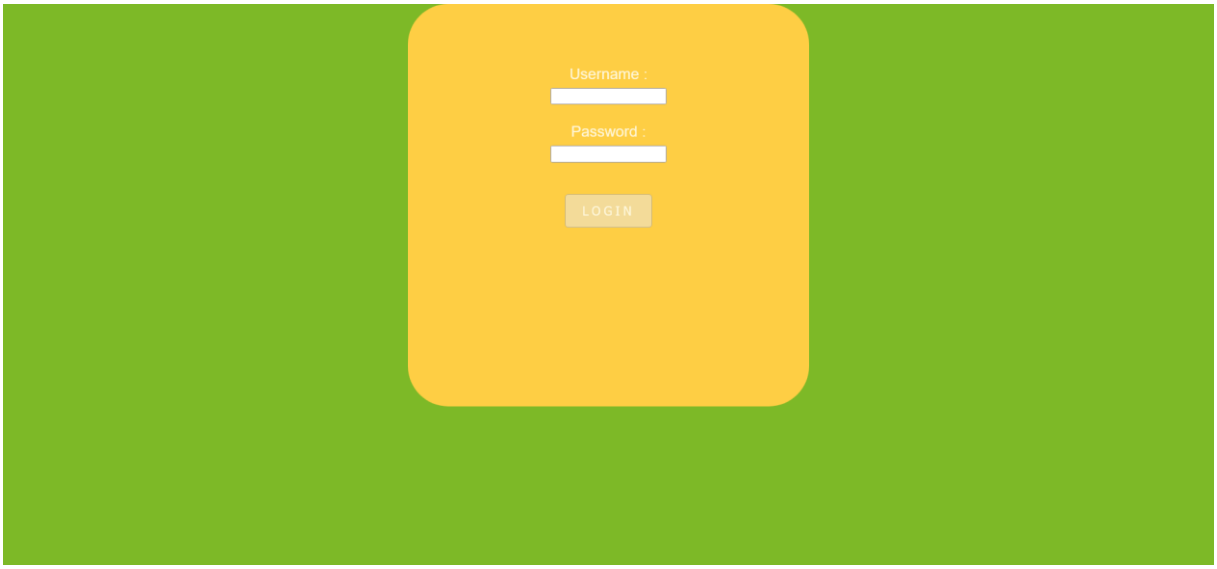
The amount of milk made by a cow is equal to its level at the power 1,5 times 10.

The price of milk is 10$ per liter.

A cow can be milked every 15 seconds times its level.

Examples/screenshots

Login :



Farm in the beginning:

First cow:

More cows: