Movie blog documentation

Bodea Alexandru group 252 HPC

Project
The software built serves as an e-blog for movie enthusiasts. Such a person can register via email and see the list of movies. Then, based on the comments left by the user these are prioritized in order to be easy to see movies of interest. Further, the user can see comments left by other users and get real time notifications.

Implementation

The **web server** has been designed in a microservices architecture approach. This approach is service-oriented and it is composed of loosely coupled elements that have bounded contexts. This loosely coupled approach will enable updating services independently. Thus, updating one microservice will not require major changes in the other services. Eventually all services will talk to each other through different APIs. The solution utilizes REST API architecture.

The current solutions part exposes four REST services: customer service (user information), movie service (movies information), comment service (comments for movies), notification service (notifications). They were written in Python, using Flask.

The **web app** consumes the specified REST services and uses microfrontend architecture, based on webpack module federation. Implementation was done in Angular 13 and includes two components: the shell-service (which deals with customer authentication/registration) and movie-service (deals with information about movies and comments).

Integration between the web server and other services is realized through an API gateway, also implemented in Python - Flask.

Containers are used to deploy the solution, through Docker and Docker-Compose. Each service (from the backend) has its own Dockerfile for deployment, while Docker-Compose is used to specify the path to each such docker file and start all the docker containers together.

Microservices patterns

Microservice Pattern
From the beginning on we tried to model the system in a decoupled and service oriented way, so this pattern was ideal for that.

API Gateway Pattern
The API Gateway defines how clients access the services in a microservice architecture and is the single entry point for all clients.

Health Check API (Observability Pattern)
A service has an health check API endpoint that returns the health of the service. The API endpoint handler performs various checks, such as the status of the connections to the infrastructure services used by the service instance.

Decompose By Subdomain (Decomposition Pattern)
Define services corresponding to Domain-Driven Design (DDD) subdomains. DDD refers to the application's problem space - the business - as the domain. A domain consists of multiple subdomains. Each subdomain corresponds to a different part of the business.

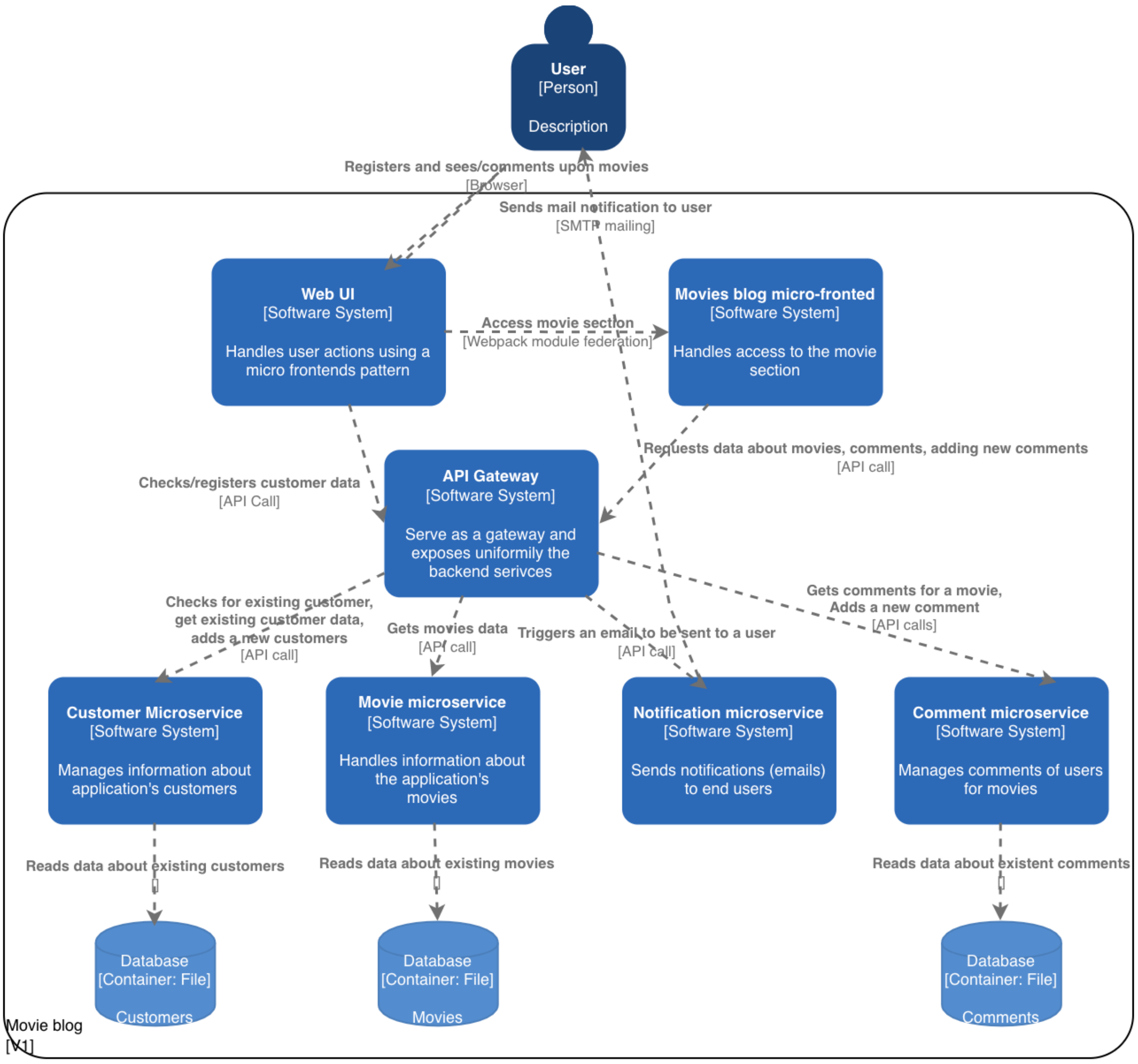
User
[Person]
Description
Registers and sees/comments upon movies
[Browser]
Sends mail notification to user
[SMTP mailing]
Web UI
[Software System]
Handles user actions using a micro frontends pattern
Access movie section
[Webpack module federation]
Movies blog micro-fronted
[Software System]
Handles access to the movie section
Checks/registers customer data
[API Call]
API Gateway
[Software System]
Serve as a gateway and exposes uniformly the backend serivces
Requests data about movies, comments, adding new comments
[API call]
Checks for existing customer, get existing customer data, adds a new customers
[API call]
Gets movies data
[API call]
Triggers an email to be sent to a user
[API call]
Gets comments for a movie, Adds a new comment
[API calls]
Customer Microservice
[Software System]
Manages information about application's customers
Movie microservice
[Software System]
Handles information about the application's movies
Notification microservice
[Software System]
Sends notifications (emails) to end users
Comment microservice
[Software System]
Manages comments of users for movies
Reads data about existing customers
Reads data about existing movies
Reads data about existent comments
Database
[Container: File]
Customers
Database
[Container: File]
Movies
Database
[Container: File]
Comments
Movie blog
[V1]

Figure 1. Container diagram (from C4 model) of the entire application/system