Music Recommender – Documentation booklet

Giuseppe Fosci
Emilio Martino
Emanuele Roncioni
Eric Welmillage
Alessio Rago

Music Recommender – Microservice architecture

The system relies on two Dockerized container networks: Internalnet and Interfacenet. This separation reflects their intended recipient for communication, with the containers in Interfacenet being meant for user-to-service communication, and the ones in Internalnet being meant for service-to-service communication. All the containers run Python Flask webservers and communicate via JSON documents sent over HTTP.

The clear network separation marks which containers are meant to act as the “elaboration” group for the composition, and the endpoints that offer the elaborated data to users.

Interfacenet features the **Interfacevis**, **Interfaceapi** and **Businesslogic** containers, with the latter being included to allow networked communication. In this architecture, Businesslogic acts as an intermediary towards the rest of the container composition, abstracting the behind-the-scenes processes and providing convenient endpoints for the “interface” containers, which only need to worry about supplying the correct JSON data to Businesslogic and performing some preprocessing of inputs and responses.

This clear separation between the implementation and the user interface for it provided significant flexibility during development: once the REST routes were agreed upon, the team could split and develop the business logic component indipendently to the interface.
It also allowed for a significant amount of code reuse, especially in the JSON-only API interface.

Interfacevis is our HTML, user-facing container. Requests sent to this endpoint are prepared and sent over to BusinessLogic for elaboration. The provided response is then used to provide a populated HTML page.

Interfaceapi operates similiarly, but the data is instead sent back as JSON formatted data. It's more limited in capabilities compare to Interfacevis, but it offers the most important services. The user can get an API token to utilize in their profile upon signing in.

Internalnet is the container composition that takes care of most of the actual computation of responses. It includes the aforementioned BusinessLogic, a MariaDB/MySQL instance, Interfacespot and mlengine, the container that performs some light machine learning to find a suitable song to recommend. **BusinessLogic** acts as an orchestrator for the other containers, collecting needed data from them and providing it to the Interface nodes when requested.

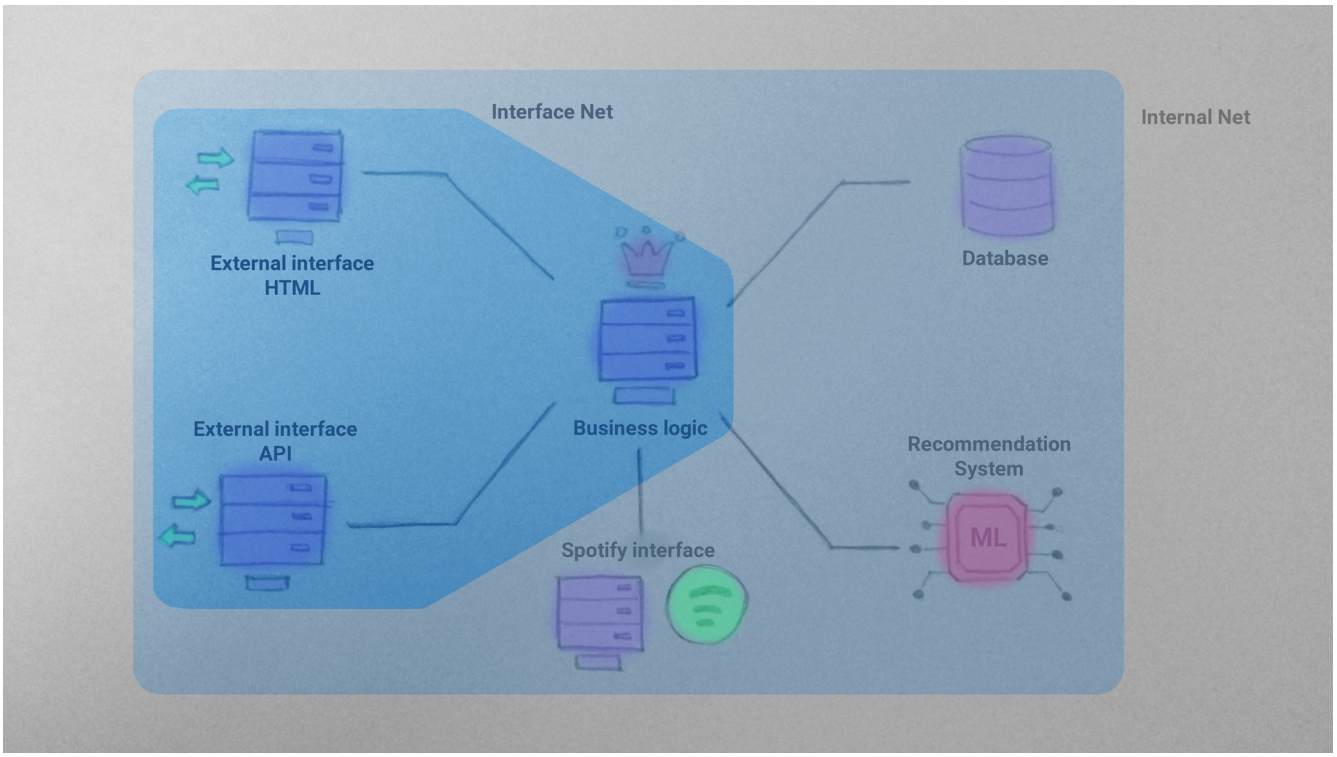
Interfacespot performs external HTTP requests against Spotify's API endpoints. These are used to obtain Spotify links to songs and their snippets, so that the user can have a preview for their searches/recommandations.

Mlengine operates with two different datasets. These two are joined when the container is built. It receives the song artist and title from BusinessLogic, and then it starts vectorization operations on the 10.000 most popular songs available (to reduce computation times). The resulting vectors are used to pick the most similar song (from a different artist).

While the extent of containerization proved beneficial, especially since it allowed the developed system to run correctly on the fairly heterogenous mix of software (every major OS, either virtualized or running on bare metal) and hardware, with various benefits in modularity and code encapsulation, in some regards the multiple layers of abstractions had made debugging more laborious than it should've been, as some errors required the developer to dig relatively deeper into the abstraction layer than normally necessary

In our inexperience we also failed to clearly name new unexpected routes beforehand, leading to some extra friction as during coding, one would often need to check the route name directly in the code. This was exhasperated by our choice of framework, as Flask is relatively barebones, offering no default CRUD routes and often requiring the developer to “reinvent the wheel”.

It would’ve also made sense to hide the interfaces behind NGINX HTTPS proxies.



Music Recommender – Development process, SCRUM overview

Development followed the SCRUM AGILE framework, taking 3 virtual sprints in total, with the sprint’s total length mapped to 2 week time spans, plus a “setup” period to gather user stories and initial microservice design.

The **first sprint** ran between early March to mid April, as the team started development. It involved the first creation of the designed container architecture (docker-composes and Dockerfiles), first setup of the DBMS/Businesslogic interaction, subsequent unexpected bug fixes and an eventual switch to an available, more robust MySQL management library

(SQLAlchemy), which took the biggest percentage of the sprint time. The remaining hours were focused on implementing some simple routes (recommendation, alongside the recommendation model) and user creation.

The sprint methodology took some getting used to, especially since we could've easily postponed some user stories in favor of other, more pressing ones, but by the end of the first sprint the "mechanism" were better understood and it helped maintaining momentum while developing. For a short period of time, work on user stories belonging to the first and second sprint overlapped.

The **second sprint** officially took part in the last two thirds of April, cut short by the summer exam session. While not every user story was completed before most of the team needed to move on, we chose to officially end this sprint early, moving the remaining user stories to the last sprint. This was done for the purpose of having an uninterrupted sprint to start back up on, to follow the SCRUM philosophy.

This sprint mostly revolved around the development of the Spotify API interface microservice and general authentication systems. The latter in particular took longer than it should have, as the barebones nature of Flask led us to trying different approaches to authentication before settling on JWT authentication. The Spotify microservice on the other hand turned out to be a far more care-free task, and was later converted to a standard library for more consistent usage.

The **third and last sprint** took most of October, resembling a "conventional" sprint the most. This included every remaining user story plus some leftovers that needed other user stories to be fully tested and developed.

This included the implementation of the song clustering algorithm to calculate recommendations, a snippet system that allows previewing the found songs, a rework of the recommendations model and interface, some light visual improvements and the token “payment” system, alongside the review routes to recover the spent tokens.

WORK BREAKDOWN STRUCTURE	TASK TITLE	TASK OWNER	AMOUNT OF WORK IN HOURS			SPRINT	START DATE	END DATE	DURATION	PCT OF TIME TAKEN
			ESTIMATE	COMPLETED	REMAINING					
0	Design and Setup (early March)		157	215	-58				102	137%
0.1	User stories	Entire team	5	6	-1	1	3/2/2024	3/4/2024	3	120%
0.2	Function points	Entire team	5	8	-3	1	3/3/2024	3/6/2024	3	120%
0.3	Container design	Entire team	5	6	-1	1	3/2/2024	3/3/2024	2	120%
1	Sprint 1 (March to mid april)		76	107	-31				48	141%
1.3	Container composition setup	Emanuele	5	8	3	1	3/3/2024	3/8/2024	2	100%
1.8	User model creation	Eric, Giuseppe, Alessio	3	3	0	1	3/12/2024	3/19/2024	2	100%
1.9	User creation route (first draft)	Alessio, Giuseppe, Eric	5	3	2	1	3/29/2024	3/30/2024	2	60%
1.4	DBMS setup and SQL interaction	Entire team	5	11	-6	1	3/18/2024	3/30/2024	13	220%
1.6	User past recommendations route, rec r	Emanuele, Emilio, Alessio	3	2	1	1	3/23/2024	3/23/2024	2	67%
1.7	First CSS draft	Emilio, Eric, Giuseppe	5	5	0	1	3/23/2024	3/23/2024	1	100%
1.7	User token route	Eric	2	1	1	1	3/23/2024	3/23/2024	1	50%
2.1	DBMS fix	Entire team	3	10	-7	2	4/8/2024	4/14/2024	7	333%
2.1	Switch to SQLAlchemy for DBMS	Entire team	10	13	-3	2	4/16/2024	4/20/2024	5	130%
2	Sprint 2 (Mid to late April)		25	32	-7				11	128%
2.4	Authentication and session system	Entire team	10	19	-9	2	4/8/2024	4/20/2024	2	190%
2.5	Switch to token authentication	Emilio, Emanuele	5	7	-2	2	4/16/2024	4/20/2024	5	140%
2.7	Spotify container and API calls	Emilio, Emanuele, Giuseppe	10	6	4	2	4/20/2024	4/24/2024	2	60%
2.7	Spotify service integration	Emilio, Emanuele, Eric	5	4	1	2	4/20/2024	4/24/2024	2	80%
3	Sprint 3 (October + extra summer days)		40	47	-7				18	118%
3.1	ML readup and design (sprint 1)	Emanuele	5	12	-7	3	8/22/2024	10/11/2024	51	240%
3.1	Song snippets system	Emilio	5	5	0	3	10/11/2024	10/11/2024	1	100%
2.2	Recommendations model rework (sprint	Emanuele, Giuseppe, Eric, Em	5	4	1	2	10/26/2024	10/26/2024	1	80%
2.3	Recommendations interface (sprint 2)	Emilio, Giuseppe	10	7	3	2	10/26/2024	10/26/2024	2	70%
3.1	Interface improvements	Eric, Emilio, Giuseppe	4	5	-1	3	10/11/2024	11/1/2024	22	125%
3.2	Token payment (sprint 1)	Emilio, Giuseppe	5	5	0	3	10/11/2024	10/11/2024	1	100%
2.6	Token validations (sprint 1)	Emilio, Alessio	6	5	1	2	10/11/2024	10/11/2024	1	83%
3.3	Recommendation system with ML	Emanuele, Emilio, Giuseppe	10	14	-4	3	10/11/2024	10/26/2024	16	140%
3.4	Recommendation interface	Emilio, Giuseppe, Alessio	10	10	0	3	10/26/2024	10/30/2024	5	100%
3.6	Review route and interface	Emilio, Emanuee, Giuseppe, A	5	8	-3	3	10/30/2024	11/2/2024	2	100%