quizTime

Laboratory of Advanced Programming 2021-22

Alessandro Marzilli	1811092
Andrea Mazzitelli	1835022
Giorgia Ristich	1839919
Andrea Rodriguez	1834937

Index

The Idea

We created an application that makes TV game shows interactive. The audience can use a web app to play the game answering the questions asked live in the studio. They have an interface to choose between correct answers and be included in a scoreboard in the show.

At the start of the game, each user that opens the mobile web app will be assigned an identifier code and will be asked to give a nickname. When a question is asked in the studio, all the players will see it through the television and 4 buttons will appear on their smartphone screen. Each button will be assigned to a possible answer. The player will only have to press the button linked to the answer they think is correct. The web app will then send to the server the response together with the id of the player, his nickname and an identifier of the question. When time runs out the buttons disappear.

Having a lot of participants, the server managing users' answers is distributed. We have a set of workers, each one handling only a fraction of the total requests. In this way the system can tolerate data burst. A load balancer is used to split the requests among all the workers.

The admin has an interface to set the correct answers of the questions. For each round he will trigger messages to notify the start of a question to the web applications and to the server accepting the responses. There will be a time out for the question, so the workers stop accepting data and send the scores of every player to Elasticsearch. Using Kibana we generate statistical graphs and the scoreboard to be displayed in the admin interface and in the show for the audience. At the end of the game every player will receive his score and the top 10 will be displayed in the show for the audience.

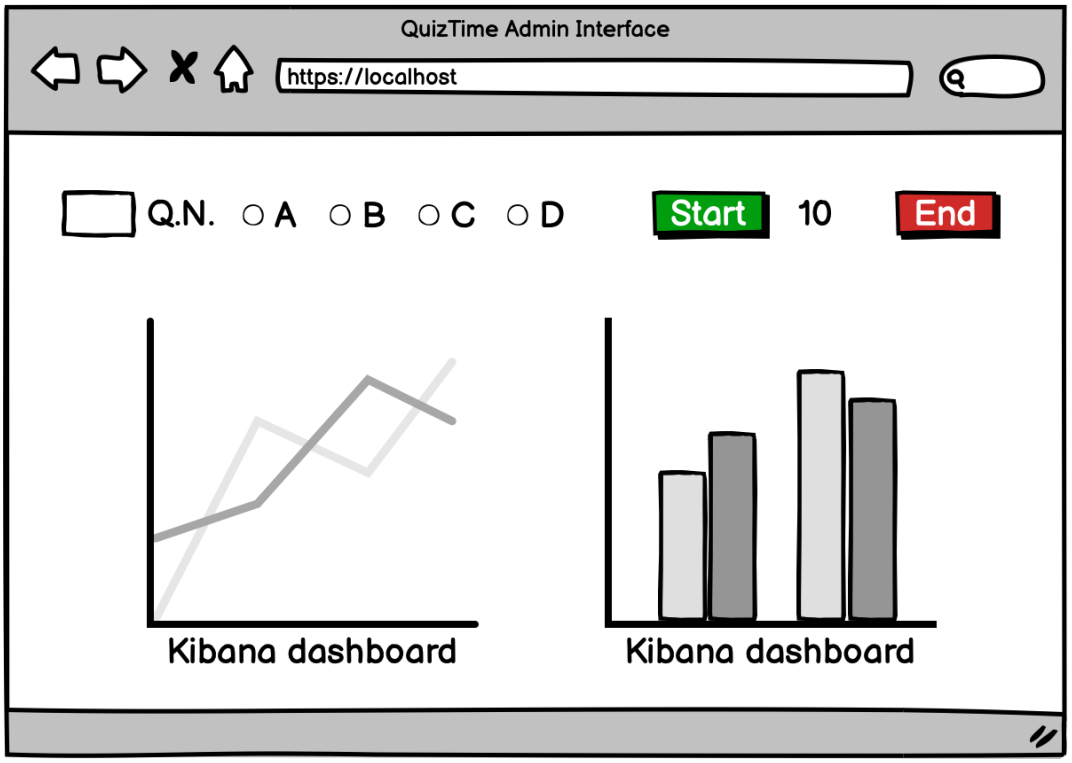
User stories

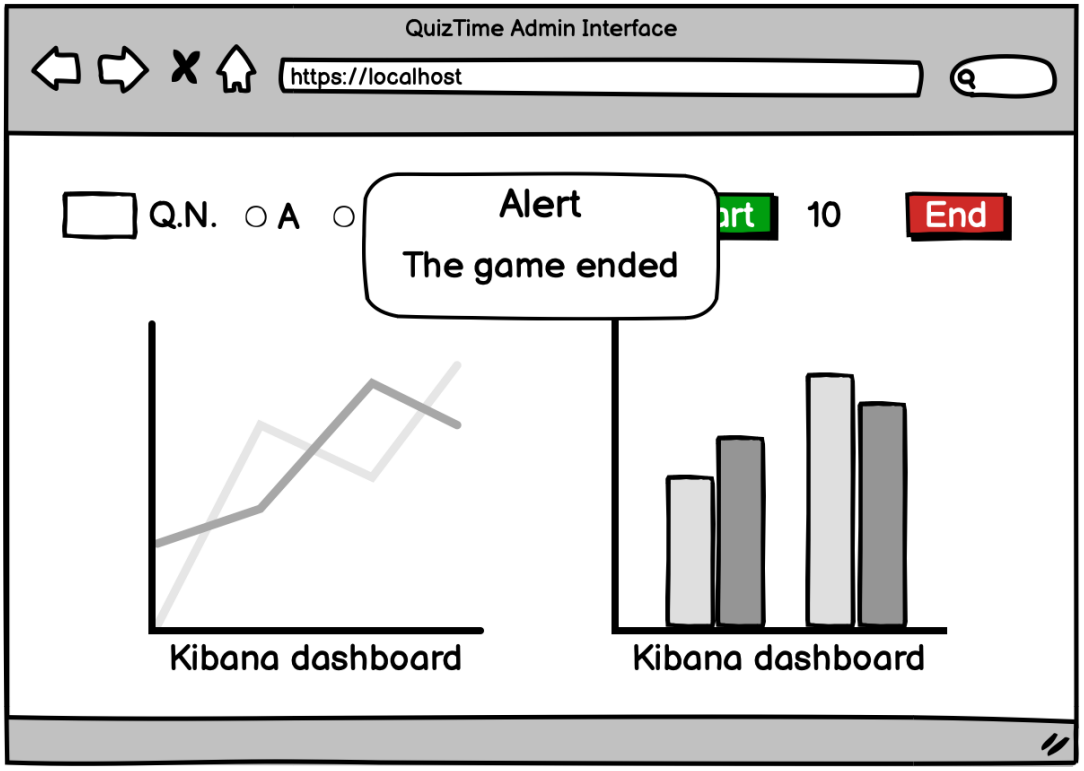
TASK TITLE	TASK DESCRIPTION	ADDED BY	DATE ADDED
User-Interface	As a User I want to have a web interface that I can access so that I can play the game	The Team	1/8/2022
Nickname	As a User I want to have field in the main page where I can add a nickname	The Team	1/8/2022
Question	As a User I want to have I clear way to understand that I am waiting for a question	The Team	1/8/2022
Answer	As a User I want to have an easy way to answer the questions asked in the tv program	The Team	1/8/2022
Final score	As a User I want to see my final score when the game ends	The Team	1/8/2022
Multiple games	As a User I want to be able to play multiple games with different nickname	The Team	1/8/2022
Problem	As a User I want to be able to access again the game in case for some reason I exited	The Team	3/8/2022

Game managing	As an Admin I want to have an interface that allows me to manage the game	The Team	1/8/2022
Start game	As an Admin I want to have in the interface the ability to start a new game	The Team	1/8/2022
Correct answer	As an Admin I want to see the next question number and to select the correct answer from it	The Team	1/8/2022
Admin-interface	As an Admin I want a dedicated interface completely disconnected from the users' one	The Team	1/8/2022
End game	As an Admin I want the ability to end the game so to notify the players of their score	The Team	1/8/2022
Score's overview	As an Admin I want to have an overview of the score of the actually playing users	The Team	1/8/2022
Answers' overview	As an Admin I want to have an overview of the answers to the last question done	The Team	1/8/2022
Problem solving	As an Admin I want some feature that help me avoid making mistakes such as starting a new question too soon	The Team	3/8/2022
Cloud	As a Admin I way the system to run on cloud so that I can monitor and balance at need	The Team	29/8/2022

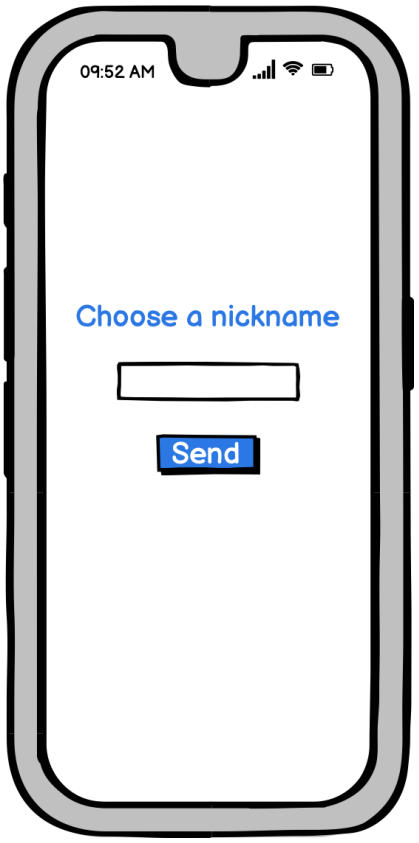
Mockups

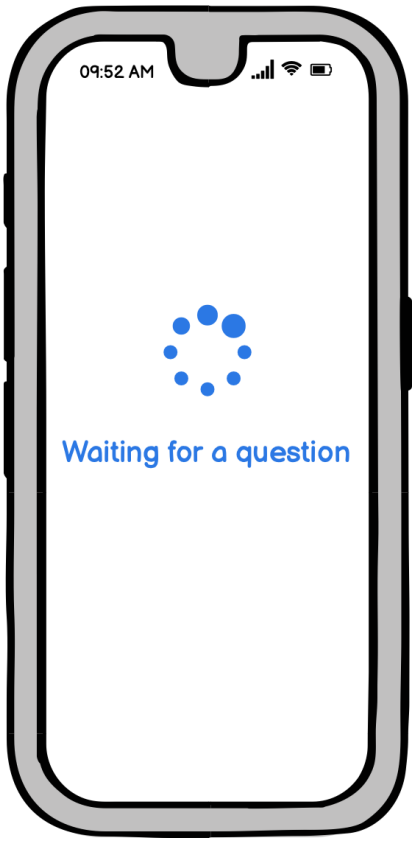
Admin Interface

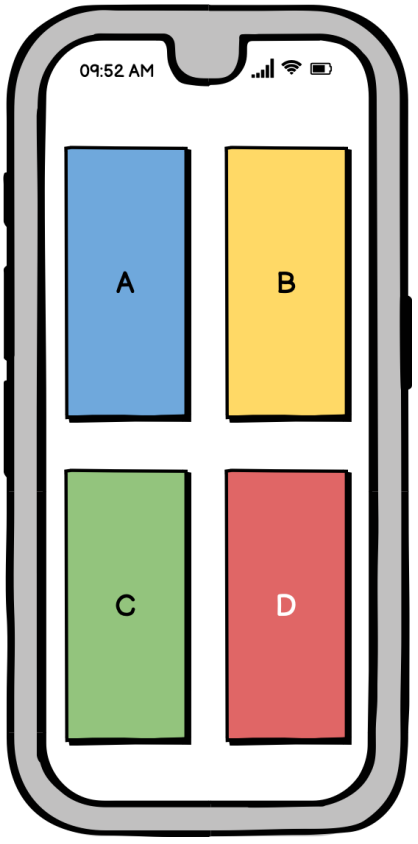


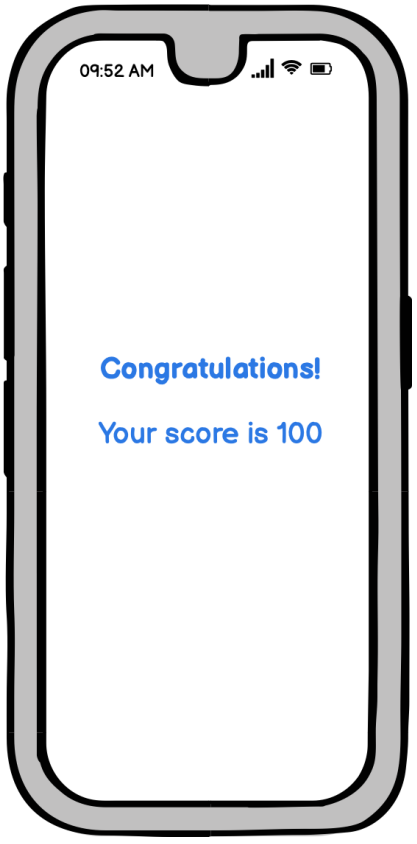


User Interface









Non functional requirements

The main non-functional requirements that we would like to provide are:

- Effectiveness: the user should clearly achieve his goal to play the quiz. Indeed he should be able to enter the game, select an answer and receive his final score without possibility of error using a minimal interface.
- Efficiency: the user must play the game with relative ease.
- Satisfaction: we believe that the minimal interface and the simple interaction help give comfort, pleasure, trust and usefulness to the player while using the product.
- Throughput: the system should be able to process the high quantity of players' answers in a short amount of time, in order to produce the charts to be shown live in the show.
- Scalability: the system must be able to handle a great number of connected players at the same time.
- Reliability: the system should be available during the whole game to avoid players from stopping watching the show. It should also be fault tolerant to manage several different types of device types and connection qualities which could fail. Moreover it must allow the users to recover a game session after having been disconnected for any reason.
- Usability: the system should be easy to understand and to learn for every user. We want to keep the system simple as much as possible to avoid possible mistakes and problems while playing the game.
- Maintainability: the system should be built in order to allow a maintainer to code eventual changes needed to keep up with updates of the show format.
- Portability: the product must be built using containers to let any operator be able to deploy the system on every hardware or software. Moreover the user interface must be compatible with every device a player could use.

Function points and Cocomo

FUNCTION POINT CALCULATION

Adjusted FP	**39,59**

Unadjusted FP	**37**

No.	Module	Function Name	Description	Type	DET	RET / FTR	Complexity	FP	Adjust %	FP adjusted
1	Player	Nickname	Insert a nickname	EI	2	1	Low	3		**3**
2	Player	Buttons	Select an answer	EI	4	1	Low	3		**3**
3	Player	Score	Show the score	EQ	1	1	Low	3		**3**
4	Game	Player	Player database	ILF	3	1	Low	7		**7**
5	Questions	Question	Questions database	ILF	5	2	Low	7		**7**
6	Admin	Question	Start a question	EI	6	2	Low	3		**3**
7	Admin	End	End the game	EI	1	2	Low	3		**3**
8	Admin	Scoreboard	Compute the scoreboard	EO	3	1	Low	4		**4**
9	Admin	Statistics	Produce statistics about the last question	EO	4	1	Low	4		**4**

COCOMO RESULTS for quizTime

MODE	"A" variable	"B" variable	"C" variable	"D" variable	KLOC	EFFORT (in person-months)	DURATION, (in months)	STAFFING
organic	2.2200914736	1.05	2.5	0.38	1665	3.792	0.94	4

Explanation: The coefficients are set according to the project mode selected, (as per Boehm).
The final estimates are determined in the following manner:
effort = a*KLOC^{b}, in person-months, with KLOC = lines of code, (in thousands), and:
staffing = effort/duration
where a has been adjusted by the factors:

Product Attributes
Required Reliability 1.00 (N)
Database Size 0.94 (L)
Product Complexity 1.00 (N)
Computer Attributes
Execution Time Constraint 1.00 (L)
Main Storage Constraint 1.00 (L)
Platform Volatility 1.00 (N)
Computer Turnaround Time 0.87 (L)

Personnel Attributes
Analyst Capability 1.00 (N)
Applications Experience 1.13 (L)
Programmer Capability 1.00 (N)
Platform Experience 1.10 (L)
Programming Language and Tool Experience 1.00 (N)

Project Attributes
Modern Programming Practices 0.91 (H)
Use of Software Tools 1.00 (N)
Required Development Schedule 1.00 (N)
Required reusability 1.00 (L)
Documentation match to life-cycle needs 1.00 (L)
Personnel continuity 1.00 (VH)
Multisite development 1.00 (L)

https://strs.grc.nasa.gov/repository/forms/cocomo-calculation/

Technologies

We used the following technologies:

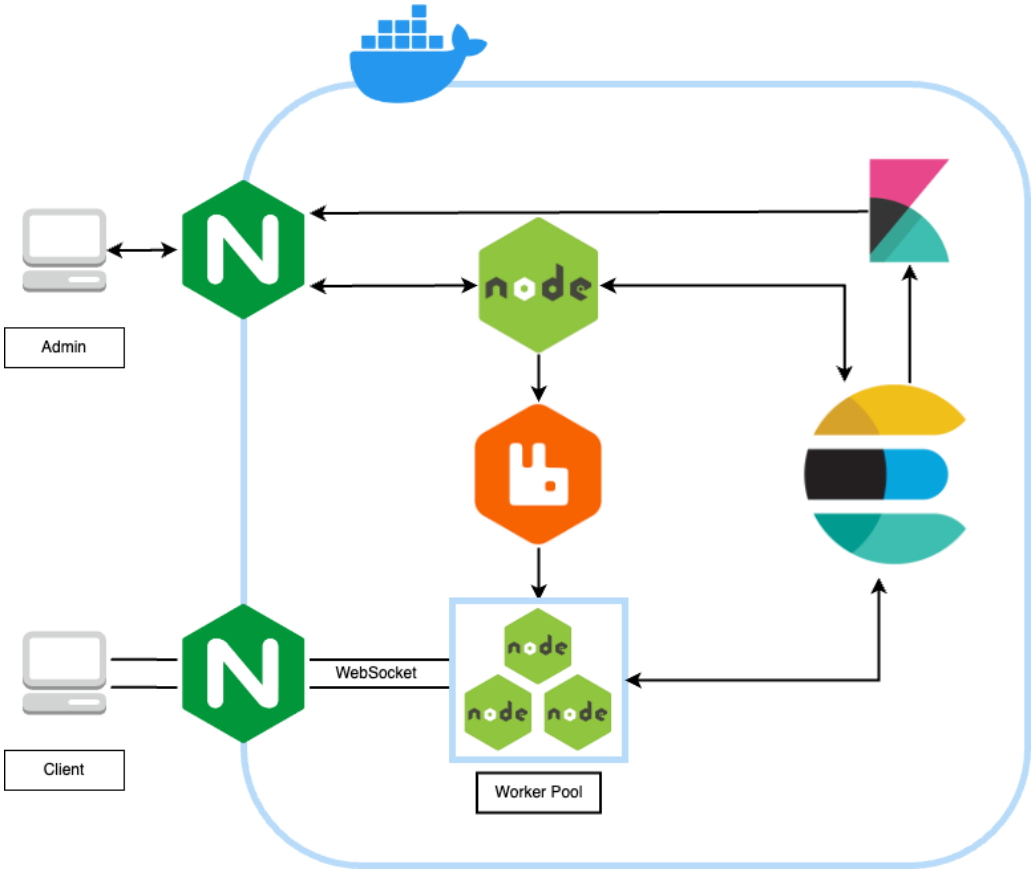
- **Docker:** to create, connect and deploy the different microservices needed for the correct working of the system. Moreover, starting from a base image, it allowed us to build personalized ones that are specific for our needs. The use of docker also simplifies cloud deployment thanks to the provided integration with AWS.
- **Elasticsearch:** as NoSQL Database. We choose it because it allows for a native interaction with another service (Kibana, see later). Being NoSQL it allows us to be flexible in how we store data without the constraints of an ER Schema.
- **Kibana:** in combination with elasticsearch, it allows us to easily create dashboards that are then shown to the admin. They provide significant information about the players and more in general about the state of the game, using data stored in the database.
- **RabbitMQ:** as a Message Broker, implemented with a Publish-Subscribe model.
- **Nginx:** we used it in several different ways depending on the type of services it needs to provide. In particular as a Web Server to provide static content to clients, as Load Balancer to evenly distribute requests among the worker and as a Reverse Proxy to handle the access to Kibana's dashboards and admin's APIs.
- **NodeJS:** to implement the different types of servers. In particular we used the express framework to create REST APIs and WebSockets.

We also need to briefly mention Amazon Web Services that allowed us to easily deploy on cloud the whole system. In particular we used:

- **Elastic Compute Cloud - EC2:** to create a load balancer that has a static address. In this way we are able to have a fixed entrypoint where to contact the system.
- **Elastic Container Services - ECS:** to automatically deploy the services as containers in the cloud.

Software Architecture

In the image we can see how we composed the different services together to build a more complex architecture.



We can divide the system into 3 distinct “logical subsets”: Base Services, Users Side and Admin Side.

Base Services: they are the backbone of the architecture; this subset includes all those services that are necessary for the system to function: elasticsearch, rabbitmq and nginx.

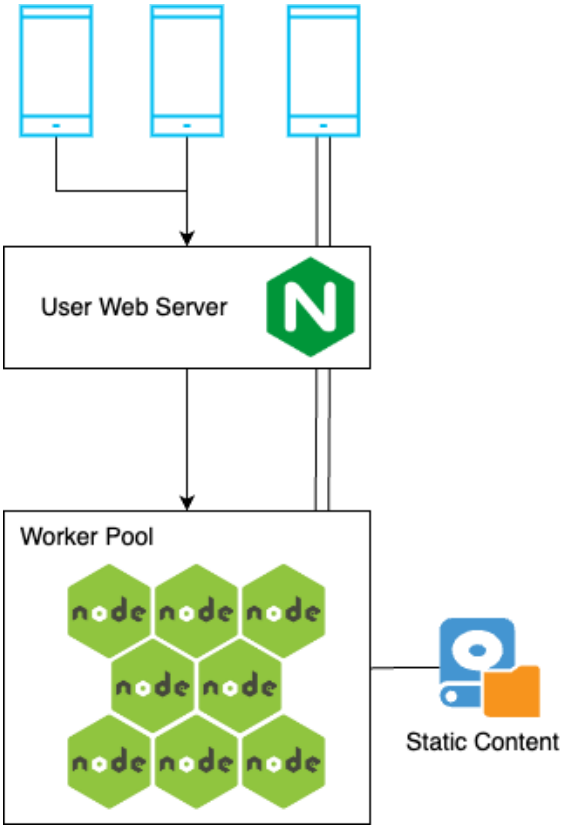
[NOTE: we will be more specific of the different instances of nginx in the discussion of the next part of the system]

Indeed without these microservices we would not be able to contact the system and the internal components would not be able to communicate within each other.

User Side: contains all those services used by users to play the game. It is composed of:

- Nginx: this instance of nginx is used as a Load Balancer for the requests. In particular, it distributes the client's request to one of the workers in the worker pool.
- Worker Pool: is a set of (almost) identical instances of a Worker Image. The role of each worker is the same and consists in being an endpoint for a websocket. Moreover it is subscribed to the message broker queues where it receives instructions from the admin such as: send a new question to all connected users or send the final score (once the game has ended). Each worker also interacts with the database to write and read information gathered during the game.

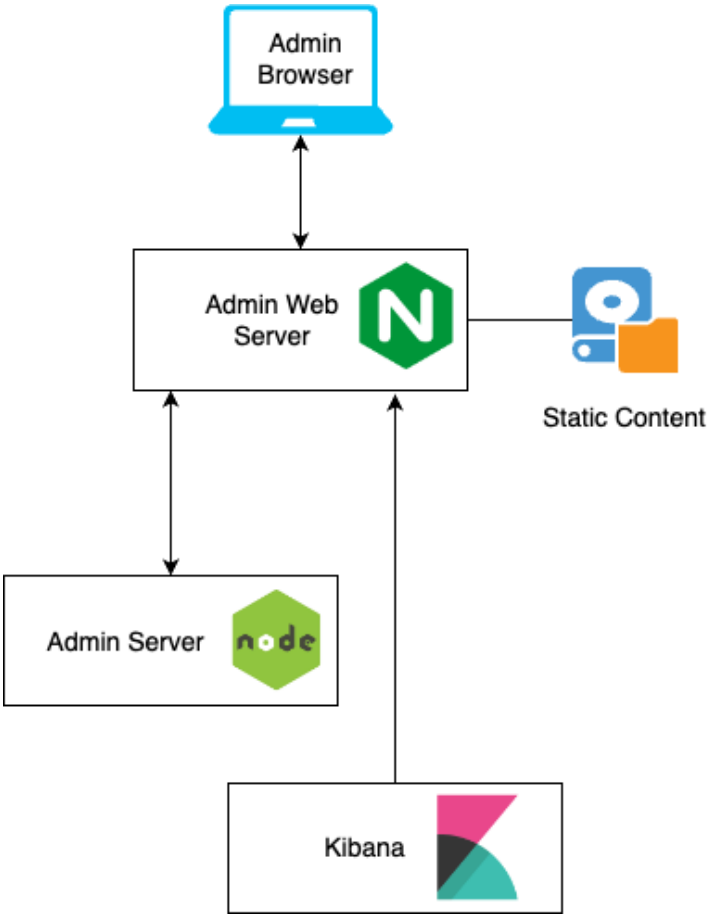
[NOTE: the instances of the worker pool are not identical due to how we deploy them. Indeed, our architecture is deployed using Docker on a single machine so we cannot use the same port for two different containers. Because of this, we are forced to specify as environment variables a different port for each worker, making them slightly different.]



(the arrows means that the client is doing a new request instead the "channel" is an established websocket connection)

Admin Side: contains all those services used by the admin to interact with the system. It is composed of:

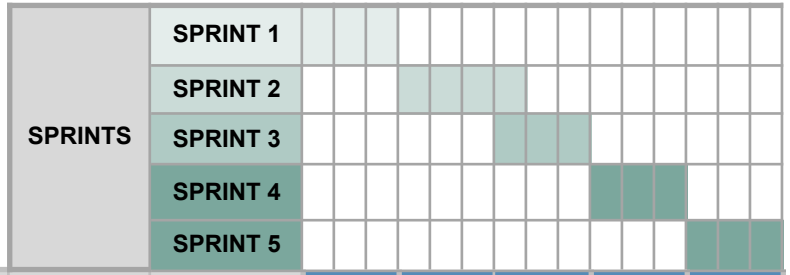
- Nginx: the instance of nginx used for the Admin Side works as a Web Server providing static content and a SSL Connection to the user. Moreover it also works as Reverse Proxy which allows us to offer an SSL Connection to Kibana and to the APIs.
- NodeJS Server - Admin Server: that provides the REST APIs needed to start/end a game and send a new question. It interacts with the message broker as a publisher to notify all the workers in the system. Moreover it also interacts with elasticsearch to clean the database of old “game instances”.
- Kibana: provides to the Admin Web Interface two dashboards. The first shows the list of players with their id, nickname and score ordered by score. The second one shows the number of answers for each option with regard to the last question.



Scrum

GANTT CHART

We developed the system in 5 sprints. The first sprint was dedicated to the general infrastructure. In the second one we developed the admin side of the system. In the third one we developed the user side. We used the fourth sprint to perform some tests and to fix bugs. In the last sprint we deployed the system in the cloud.



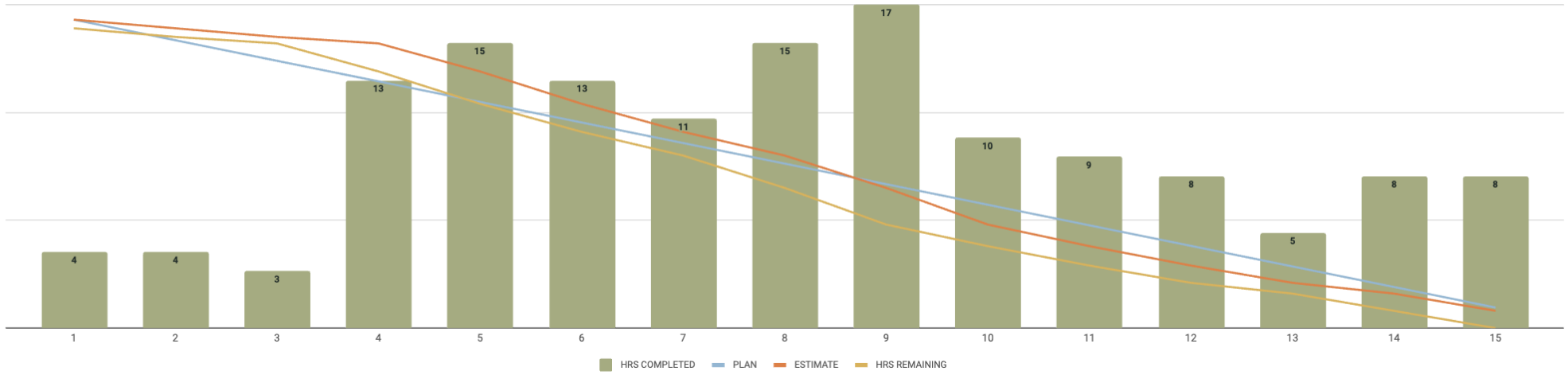
WORK BREAKDOWN STRUCTURE	TASK TITLE	TASK OWNER	AMOUNT OF WORK IN HOURS			SPRINT	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	WEEK 1			WEEK 2			WEEK 3			WEEK 4			WEEK 5		
			ESTIMATE	COMPLETED	REMAINING						M	T	W	M	T	W	M	T	W	M	T	W	M	T	W
1	General infrastructure		11	11	0					100%															
1.1	Create Docker infrastructure	Andrea M.	11	11	0	1	8/1/2022	8/3/2022	3	100%															
2	Admin infrastructure implementation		41	41	0					100%															
2.1	Address Issues	ALL	3	3	0	2	8/8/2022	8/8/2022	1	100%															
2.2	Configure Admin Nginx	Andrea M.	5	5	0	2	8/8/2022	8/8/2022	1	100%															
2.3	Create Admin Server	Giorgia R.	10	10	0	2	8/8/2022	8/9/2022	2	100%															
2.4	Create Admin Interface	Andrea R.	10	10	0	2	8/9/2022	8/10/2022	2	100%															
2.5	Create DB Interaction Function	Alessandro M.	7	7	0	2	8/9/2022	8/10/2022	2	100%															
2.6	Create Connection to Message Broker	Giorgia R.	6	6	0	2	8/9/2022	8/10/2022	2	100%															
3	User infrastructure implementation		43	43	0					100%															

3.1	Address Issues	ALL	2	2	0	2	8/22/2022	8/22/2022	1	**100%**
3.2	Fix the Infrastructure	Andrea M.	3	3	0	3	8/22/2022	8/23/2022	2	**100%**
3.3	Configure User Nginx	Andrea M.	4	4	0	3	8/22/2022	8/23/2022	2	**100%**
3.4	Create User Server	Andrea R.	13	13	0	3	8/22/2022	8/24/2022	3	**100%**
3.5	Create User Interface	Alessandro M.	15	15	0	3	8/23/2022	8/24/2022	2	**100%**
3.6	Create Connection to Message Broker	Giorgia R.	6	6	0	3	8/23/2022	8/24/2022	2	**100%**
4	Final touches		27	27	0					**100%**
4.1	Address Issues	ALL	10	10	0	4	8/29/2022	8/31/2022	3	**100%**
4.2	Bug Fixes	ALL	17	17	0	4	8/29/2022	8/31/2022	3	**100%**
5	Cloud Deployment		21	21	0					**100%**
5.1	Understand Feasibility	Andrea R., Andrea M.	3	3	0	5	9/5/2022	9/5/2022	1	**100%**
5.2	Cloud Adjustments	Andrea R., Andrea M.	7	7	0	5	9/5/2022	9/6/2022	2	**100%**
5.3	Issue Solving	Giorgia R.	6	6	0	5	9/6/2022	9/7/2022	2	**100%**
5.4	Bug Fixes	Alessandro M.	5	5	0	5	9/7/2022	9/7/2022	1	**100%**

	ESTIMATE	COMPLETED	REMAINING	DAYS	EST/DAYS
TOTAL HOURS	143	143	0	15	9.53

DAY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PLAN	143	133	124	114	105	95	86	76	67	57	48	38	29	19	10
ESTIMATE	143	139	135	132	119	104	91	80	65	48	38	29	21	16	8
HRS COMPLETED	4	4	3	13	15	13	11	15	17	10	9	8	5	8	8
HRS REMAINING	139	135	132	119	104	91	80	65	48	38	29	21	16	8	0

BURNDOWN CHART



RELEASE BACKLOG

PRIORITY	SPRINT	FUNCTIONALITY	TASK TITLE	TASK DESCRIPTION	TASK OWNER	WORK ESTIMATE IN HOURS	STATUS	NOTES
1	1	General Infrastructure	Create Docker Infrastructure	Build the rough structure of the docker compose file and the base configuration of the main services	Andrea M.	**11**	Completed	
1	2	General	Address Issues	Address the issues/doubts discovered in the previous sprint	ALL	**3**	Completed	
2	2	General Infrastructure	Configure Admin Nginx	Define the configuration to access to the admin panel	Andrea M.	**5**	Completed	
2	2	Admin - Side	Create Admin Server	Define the basic API of the Admin Server	Giorgia R.	**10**	Completed	

2	2	Admin - Side	Create Admin Interface	Implement the basic web interface for the admin	Andrea R.	**10**	Completed	SSL missing
2	2	General Infrastructure	Create DB Interaction Function	Create the necessaries functions to interact with the choosen database	Alessandro M.	**7**	Completed	Still need to manage concurrency
3	2	General Infrastructure	Create Connection to Message Broker	Make the admin server write on the message broker	Giorgia R.	**6**	Completed	
1	3	General	Address Issues	Address the issues/doubts discovered in the previous sprint	ALL	**2**	Completed	
1	3	General Infrasturcture	Fix the Infrastructure	Refine the infrastructure build, adjusting the parameters according to how the development evolved	Andrea M.	**3**	Completed	
2	3	General Infrastructure	Configure User Nginx	Define the configuration enabling load balancing for the user	Andrea M.	**4**	Completed	
2	3	User - Side	Create User Server	Implement the template for the implementation of the websocket of the worker	Andrea R.	**13**	Completed	
2	3	User - Side	Create User Interface	Implement the basic web interface for the user	Alessandro M.	**15**	Completed	
3	3	General Infrastructure	Create Connection to Message Broker	Make the worker receive the messages published on the message broker	Giorgia R.	**6**	Completed	
1	4	General	Address Issues	Address the issues/doubts discovered in the previous sprint	ALL	**10**	Completed	
1	4	Test	Bug Fixes	Try to discover and fix bugs	ALL	**17**	Completed	
1	5	Cloud Deployment	Understand Feasability	Define the feasability of a cloud deployment	Andrea R., Andrea M.	**3**	Completed	
1	5	Cloud Deployment	Cloud Adjustments	Adjust the existing files to make them cloud compatible	Andrea R., Andrea M.	**7**	Completed	
1	5	Cloud Deployment	Issue Solving	Fix the issues that rose up	Giorgia R.	**6**	Completed	
1	5	Test	Bug Fixes	Try to discover and fix bugs	Alessandro M.	**5**	Completed	

Code

The code to run the project can be found at:

https://github.com/andreamazzitelli/quizTime

To run the application, clone the repository and launch the command “docker-compose up” inside the main folder.

To run the project on AWS *Elastic Container Service*, set the proper context in Docker and launch the command “docker-compose up -f docker-compose_cloud.yml”.