TQS: Product specification report

Bruno Páscoa [107418], David Cobileac [102409], Guilherme Lopes [103896]
V2024-05-28

Índice

1 Introduction

1.1 Overview of the project

With this project, we aimed not just to create a viable MVP of a product, but also to follow a complete, professional, workflow (including, but not limited to, the use of project management tools, the automation of CI/CD and code quality standards).

To do this, we decided to develop Apollo Care, a website and mobile app combo that facilitates the scheduling and management of consultations for both clinic workers and patients alike.

1.2 Limitations

As our product is still in the MVP phase, there are still planned features we have yet to implement. These features are:

- Staff page.

- Consultation queues (for demo purposes, this has been temporarily mocked using a service that randomly generates consultations).
- Notification system for the user page.
- Role-based access restrictions using RLS (so that patients can´t access other patient´s data).
- Designing a separate workflow for staff and doctor registration to enforce the above restrictions (some proposals were requiring manual registration directly in Supabase or to be added manually by another staff member/doctor).
- Show available timeslots for the selected area.
- Optimizing query time by caching/indexing frequently used results.

2 Product concept and requirements

2.1 Vision statement

ApolloCare is mainly divided into 3 groups:

First, the app and website allow the patient to schedule new consultations as well as showing available timeslots for the selected area. However, we do not force the patients to use the app, as the staff themselves can schedule the consultations.

Secondly, we also allow the patient to see both future and past consultations and allow the user to set notifications, allowing users to not lose track of them.

Finally, we also integrate the call screen/consultation queue functionality into our app, automating the process of choosing and displaying the next people to be received.

2.2 Personas and scenarios

Comentado [IO1]: or, in alternative, actors and use cases

<Uma Persona é uma personagem utilizada para contar histórias representativas da futura utilização do sistema. Uma Persona é um Actor instanciado, à qual se dá um conjunto de caraterísticas para a humanizar e definir o contexto em que usará o sistema e as suas motivações.

"Personas are fictional people. They have names, likenesses, clothes, occupations, families, friends, pets, possessions, and so forth. They have age, gender, ethnicity, educational achievement, and socioeconomic status. They have life stories, goals and tasks. Scenarios can be constructed around personas, but the personas come first. They are not 'agents' or 'actors' in a script, they are people. Photographs of the personas and their workplaces are created and displayed. […] It is to obtain a more powerful level of identification and engagement that enable design, development, and testing to move forward more effectively". Adapted from Grudin, J. and Pruitt, J., 2002, June. Personas, participatory design and product development: An infrastructure for engagement. In Proc. PDC (Vol. 2).

Exemplo: ver secção 4.1, neste artigo (open access)] >

<You don't need to include all possible details. Pick the main scenarios, related to the core value of the system.>
<The scenarios tell the story of the Personas in their lives, doing their daily/professional activities that are relevant to find the points of contact with the system under specification.
Scenarios are somewhat similar to use cases (they have a goal and tell a story), but, unlike use cases, they capture a larger process, with activities that may not use the software. Scenarios don't required a "template", like the usual use cases description.>

Exemplo: ver secção 4.2 neste artigo (open access)] >

2.3 Project epics and priorities

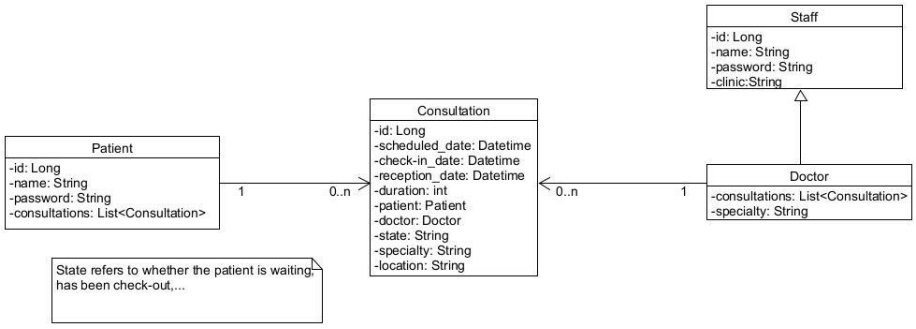
Besides the 2 sprints dedicated to the conception phase, we had a total of 3 sprints (including the current one).

The objective of the first sprint was to setup the groundwork. Our goals (besides setting up Sonarcloud and Jira) were to set up the frontend, setting up authentication, setting up Supabase and its tables (and streamlining its connection by creating a "JPA-like" abstraction) and setting up base scheduling functionalities (as future functionalities were dependent on most of these features), as well as the call screen integration (as it wasn´t dependent on anything and we wanted to get it done as soon as possible). As this was the first sprint developing actual code, we were a bit overeager and ended up having to extend the sprint for 2 more days but, overall, we were satisfied with the results given the available time.

For the second sprint, rather than try to do as many functionalities as possible, we focused on expanding the existing functionalities by displaying future/past consultations, as well as adding check-in functionality and setting up CD. As we managed to meet our deadlines, we are satisfied with the results.

For the third sprint, as it is the last week available, we are focusing on generating/writing documentation, as well as doing integration and load tests in order to test the ApolloCare as a whole (as we previously only had the opportunity to unit test specific components in isolation).

3 Domain model



4 Architecture notebook

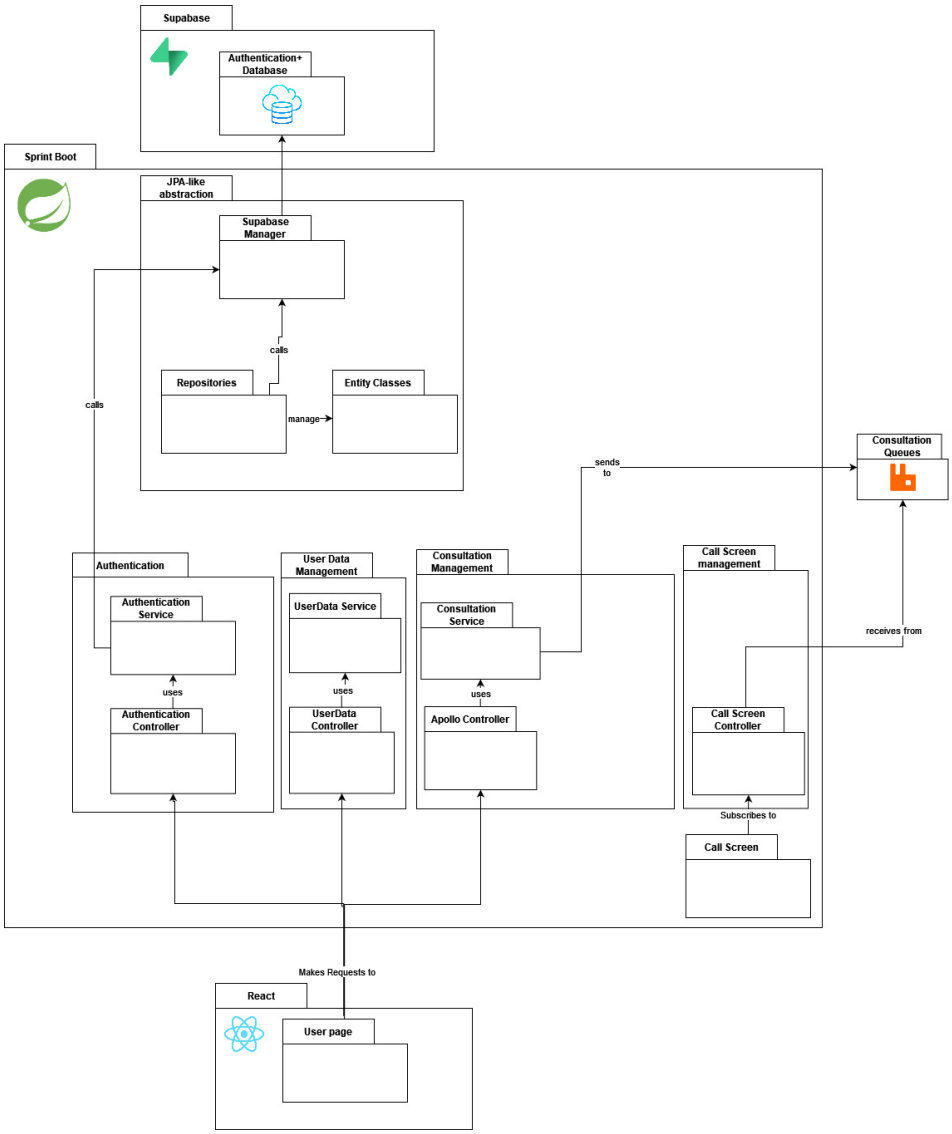
4.1 Key requirements and constrains

The following constraints had an impact on the choice of architecture:

- The call screen must dynamically display new information in less than 2 minutes -> Call screen is updated using web sockets.
- The user page must be separate from the remaining system (to allow the latter to scale independently) and, if possible, must be available in Android -> React was chosen for the frontend.
- The queue system must be decentralized to allow each clinic to run (and scale) its own queues, if desired. -> Queues are made using RabbitMQ (rather than just use a Java queue).
- While each clinic's queues may be separated, the authentication and storage features must be shared between all clinics using the system. -> Database and authentication use Supabase (as it is an external service, it can be shared even if running multiple server instances).
- Using Supabase (because it is an external service) prevents the use of JPA and there's no (official) Java client. -> Create a class to streamline the REST request (by pre-setting repetitive values). Use

repository classes to abstract the interaction with Supabase while providing a “JPA-like” interface for ease of use.

4.2 Architecture view

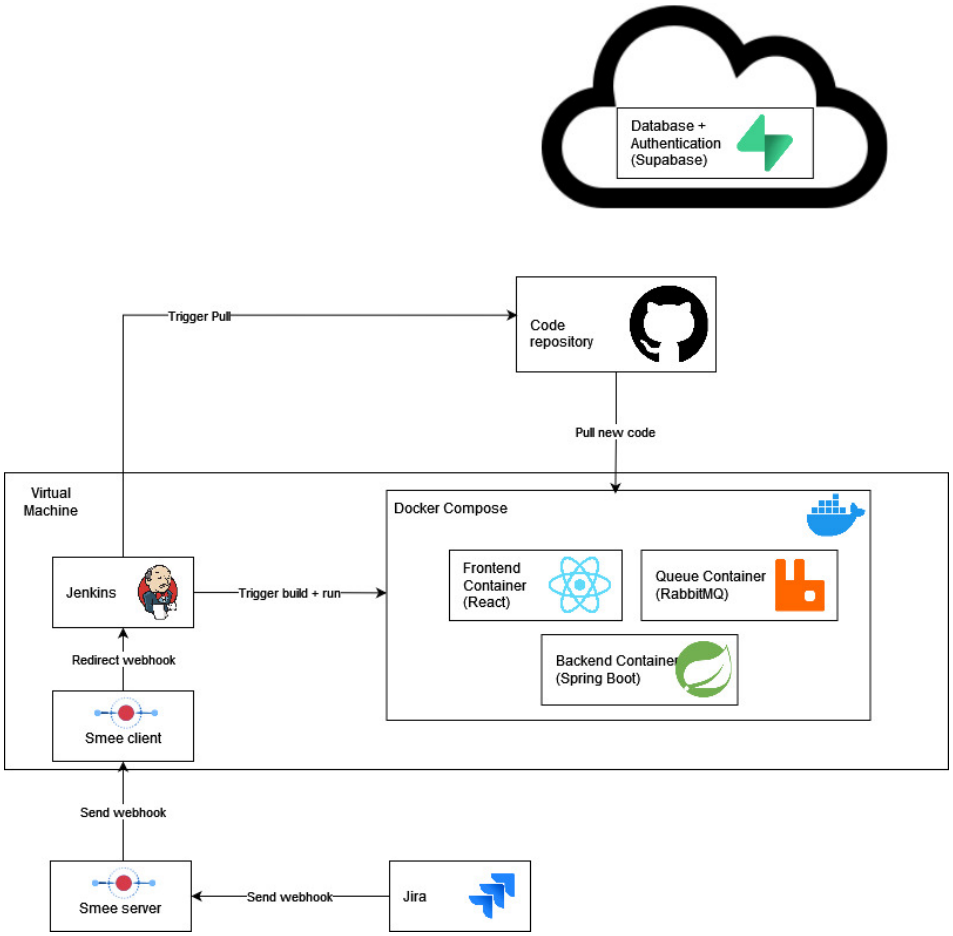


Note: for better readability, only some interactions between modules are represented in the diagram. For the full list of interactions, please consult the table below.

Component name	Technology	Purpose	Main interactions with other modules

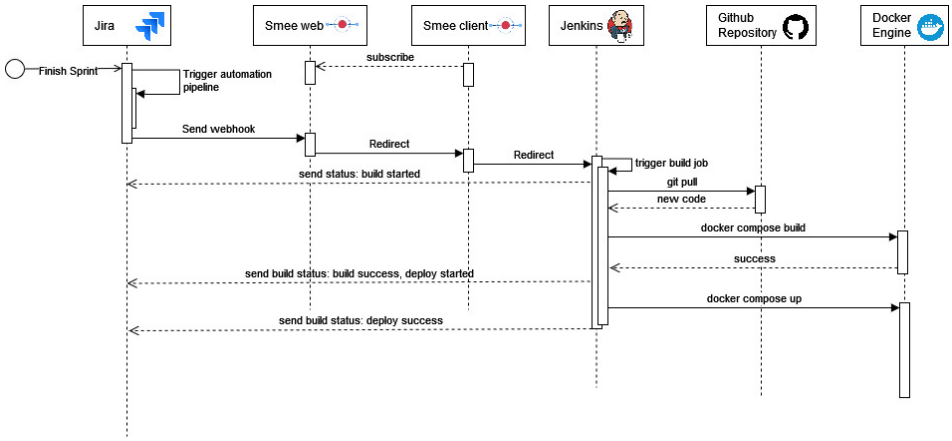
	used		Interacts with	Reason
User page	React (Ionic)	Interface for the user (patient) to use	Authentication, User Data and Apollo Controllers	Gets/Posts data
Call Screen	Spring Boot	Displays recently called consultations	Call Screen Controller	Subscribes to in order to receive updates
Authentication Service + Controller	Spring Boot	Registers and logs in the user	Supabase Manager	Calls to use Supabase's auth functionalities
			User Repository	Inserts new user on register, fetches on login.
User Data Service + Controller	Sprint Boot	Fetches user details (consultation history, for example).	Consultation Repository	Fetches consultations by patient and state (SCHEDULED or CHECKED_OUT)
Apollo Controller + Consultation Controller	Sprint Boot	Deals with consultation-related operations (schedule, check-in…)	Consultation Repository	Fetches/inserts consultations
			Consultation queues	Adds checked in consultation
Call Screen Controller	Spring Boot	API for the call screen to subscribe to	Consultation queues	Subscribes to be notified when a new consultation is called
Consultation Queues	Rabbit MQ	Queues that set call order of consultations (one per clinic per area of specialty)		
Entity classes	Sprint Boot	Each class represents a table in the database (made to mirror @Entity classes in JPA).		
Repositories	Spring Boot	Provide functions to perform database operations (made to mirror @Repository classes in JPA).	Entity classes	Each repository has a corresponding entity they insert/fetch from the database
Supabase Manager	Spring Boot	Wrapper from Spring's WebClient class. Streamlines request sending and response processing.	Supabase	Mediates requests to Supabase.

4.3 Deployment architecture



Note: the use of smee is used to “bypass” the firewall restrictions on inbound connection by having the client subscribe to the server (turning an inbound connection to an outbound connection). Supabase exists in the cloud and doesn’t require any actions during deployment.

4.3.1 Deployment architecture



5 API for developers

A API segue o seguinte formato:

- Endpoints relacionados a autenticação devem utilizar o prefixo "/auth/{versão}"
- Os restantes endpoints da api devem utilizar o prefixo "/api/{versão}"
- Urls que não possuam nenhum destes prefixos serão, numa versão futura, redirecionados para o frontend.
- Endpoints que pertençam à mesma categoria "lógica" deverão ter o mesmo prefixo (por exemplo, endpoiints relacionados aos dados do utilizador têm o prefixo "/api/{versão}/user"

A documentação completa pode ser encontrada no Swagger.

6 References and resources

Apache JMeter - User's Manual. (n.d.). Retrieved June 2, 2024, from https://jmeter.apache.org/usermanual/index.html
Introduction to Spring REST Docs | Baeldung. (n.d.). Retrieved June 2, 2024, from https://www.baeldung.com/spring-rest-docs
Supabase CLI reference - Generate bash script. (n.d.). Retrieved June 2, 2024, from https://supabase.com/docs/reference/cli/supabase-completion-bash