The Fitenss Guru

Introduction

This report presents a comprehensive overview of a service-oriented web application with a complete graphical user interface for testing. The web application is designed to cater to three different types of users, namely Athletes, Personal Trainers, and Nutritionists. The application allows Athletes to request collaborations with the other two types of users and ask them for workout plans and diets.
The aim of the web application is to provide a centralized platform for Athletes, Personal Trainers, and Nutritionists to collaborate and achieve their fitness goals. With its user-friendly interface, the web application offers a wide range of features and functionalities that help users create and manage workout plans and diet schedules.
To develop the entire project simultaneously we used github and google documents to write the report. Github link: https://github.com/spam-thunder-with-sun/SDEProject.git
In the following sections, we will discuss the technologies used, the architecture and design of the web application, and the features and functionalities offered to each type of user.

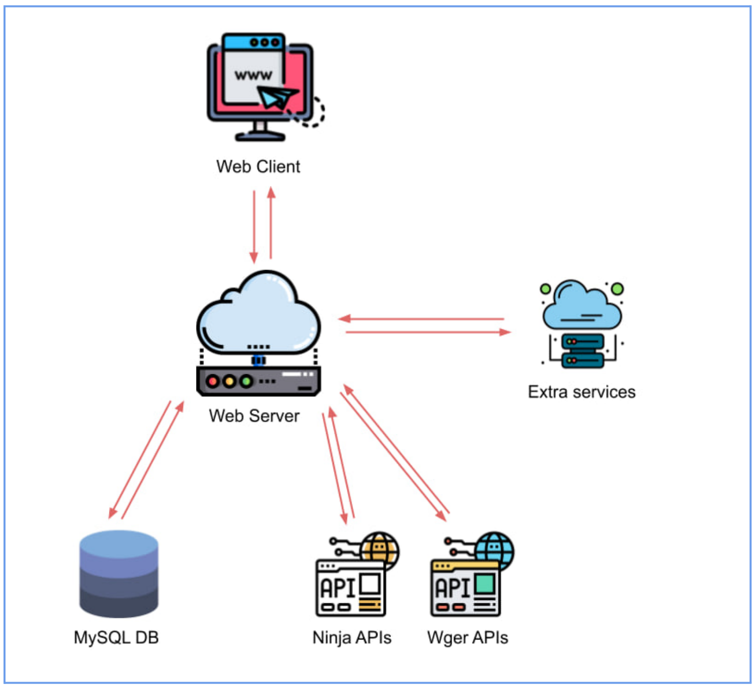
Technologies

The web application has been developed using a range of technologies to provide a robust and user-friendly platform for all our types of users.
MySQL has been used as the database for the web application. It is an open-source relational database management system that is known for its reliability, scalability, and ease of use.
The core of the web application has been developed using Servlets in Java with Web Server Tomcat. Servlets are Java classes that are used to handle requests and responses between the client and the server. Tomcat, on the other hand, is an open-source web server that is used to host Java-based web applications. The combination of Servlets and Tomcat provides a powerful and efficient framework for the web application.
For the client-side of the application, we have used HTML, JSP, CSS (importing the design of w3), and JavaScript. HTML and JSP are used for creating the structure and content of web pages, while CSS is used for styling and layout. JavaScript is used for client-side scripting and adding interactivity to the web application.
In addition, we have also imported external Java libraries to provide additional features for the web application. For example, we have used a Java library to create and manage PDF documents and a library for sending emails. This feature allows users to collaborate more effectively and efficiently, without the need for external email clients.
Finally, for the calls to the API managed by the Servlets, we have used AJAX. AJAX (Asynchronous JavaScript and XML) is a technique that allows web pages to update content without requiring a full page reload. It is commonly used in web development to create dynamic and responsive web applications.

Architectures

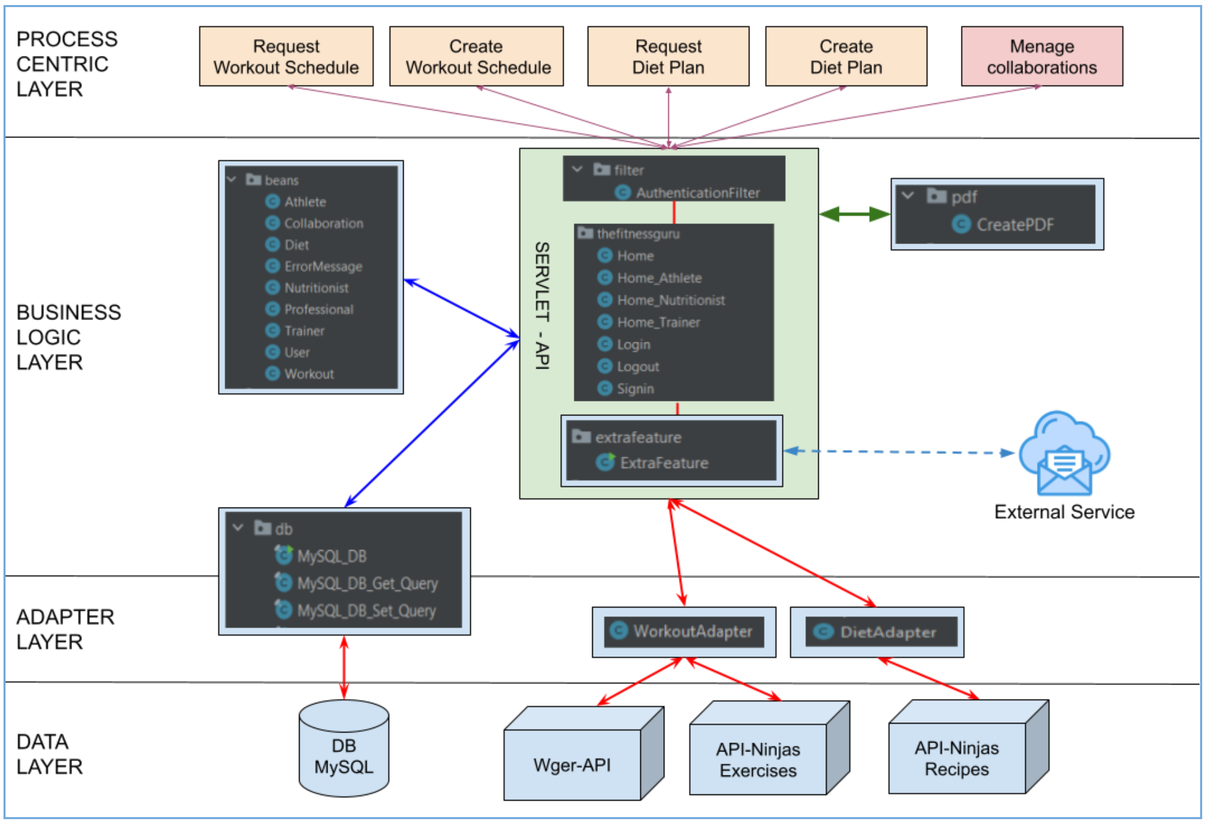
In this section we will first show the general structure of everything about the project, all the elements and the basic iterations between them. After that we will show more specifically the structure of the core of our project and the logical levels on which it is distributed.

General architecture



As you can see in the image, we developed not only the Web Server that contains the main Business Logic and all the APIs but an external web server too. This was done to contain all the possible extra services and to emulate an alternative communications way service-oriented and not only data-oriented through APIs. Right now on this external server is present a service that is able to send confirmation and notification through email. There are also 2 different APIs sources and a MySQL DB for the data storage and retrieving.

Web server architecture



In that image is shown the structure of the main Web Server with all the logical layer subdivision.
The Data Layer contains the mySQL database and all the APIs sources. Note that API-ninja is duplicated because it is a large platform and we preferred to maintain the two endpoints used separately.
The Adapter layer has two different adapters. The first one, the WorkoutAdapter, is able to handle exercise retrieval requests fetching two different APIs on two different sources and

merging both the response in an unique JSON response. The second one, the DietAdapter works in a different way: since we want more than the 10 recipes provided by the API this adapter is able to do 2 different requests to the same endpoint but with an offset. The two JSONs are then merged together and provided at the top level.
In the Business Logic layer we have a lot of components able to manage the APIs and the interface simultaneously. Starting from the filter to check if the login permission are presents, we developed all the servlets that contain the APIs. There is also the ExtraFeature class where all the calls to the External Service are handled. Of course there are the classes to communicate with the DB and all the beans to manage local data. In conclusion, in this layer there is also an internal additional service provided by the createPDF class to create well formed PDFs for diet plans and workout schedules.
In the highest layer, the Process Centric Layer, it presents all the main processes feasible through the BL using the provided APIs that we see as main features in the next section.

Functionalities

There are three types of users: Athletes, Personal Trainers and Nutritionists. Regardless of the user type, the first thing to do is login. Of course, there is also the registration step which is specific to the user type.
Once logged in, users can access the features mentioned earlier in the Process Centric Layer.
The athlete can search among the available professionals and request them to cooperate. Personal trainers and Nutritionists, at this point can accept or decline the collaboration. In the future, both parties can terminate the collaboration.
Once a collaboration is established, the athlete can execute a request for a diet plan or a training schedule specifying within it the parameters that the professionals require. Nutritionist and Personal Trainer, in their home, will then have a view of all requests with the specific requirements and can choose to reject it or to continue with the creation of the training schedule/diet plan. Calls to external APIs are used for the creation and the internal PDF service is used for the final view.
Once created, via the external service we developed, an e-mail notification is sent to the athlete of successful delivery and the result will be available in their web app home.
Here are all the links to the APIs documentation (our and external).
APIARY LINK: https://thefitnessguru.docs.apiary.io
NINJA-API exercises: https://api-ninjas.com/api/exercises
NINJA-API recipes: https://api-ninjas.com/api/nutrition
WGER-API: https://wger.de/en/software/api

Future developments & Conclusion

It is possible to expand the functionalities provided by our Web Application. Firstly, to make the collaboration more efficient we should add an internal chat or by integrating an external API service, as for instance Zoom, to do cooperative meetings with all the users.
There would be many other possible developments, but in conclusion we believe we have created a solid, API-based, easily expandable product. Now that we have concluded it, after due consideration, if we were to do it again we would use different technologies (i.e. NodeJS) that are more suitable and effective instead of using Tomcat and Servlet that we already know well.

Group members

- Stefano Faccio {email: stefano.faccio@studenti.unitn.it - Matricola: 241373}
- Giovanni Rigotti {email: giovanni.rigotti@studenti.unitn.it - Matricola: 239109}