University of Trento - Introduction to Service Design and Engineering

Course Assignment: *Virtual Lifestyle Coach*

Gianvito Taneburgo (182569)

Introduction

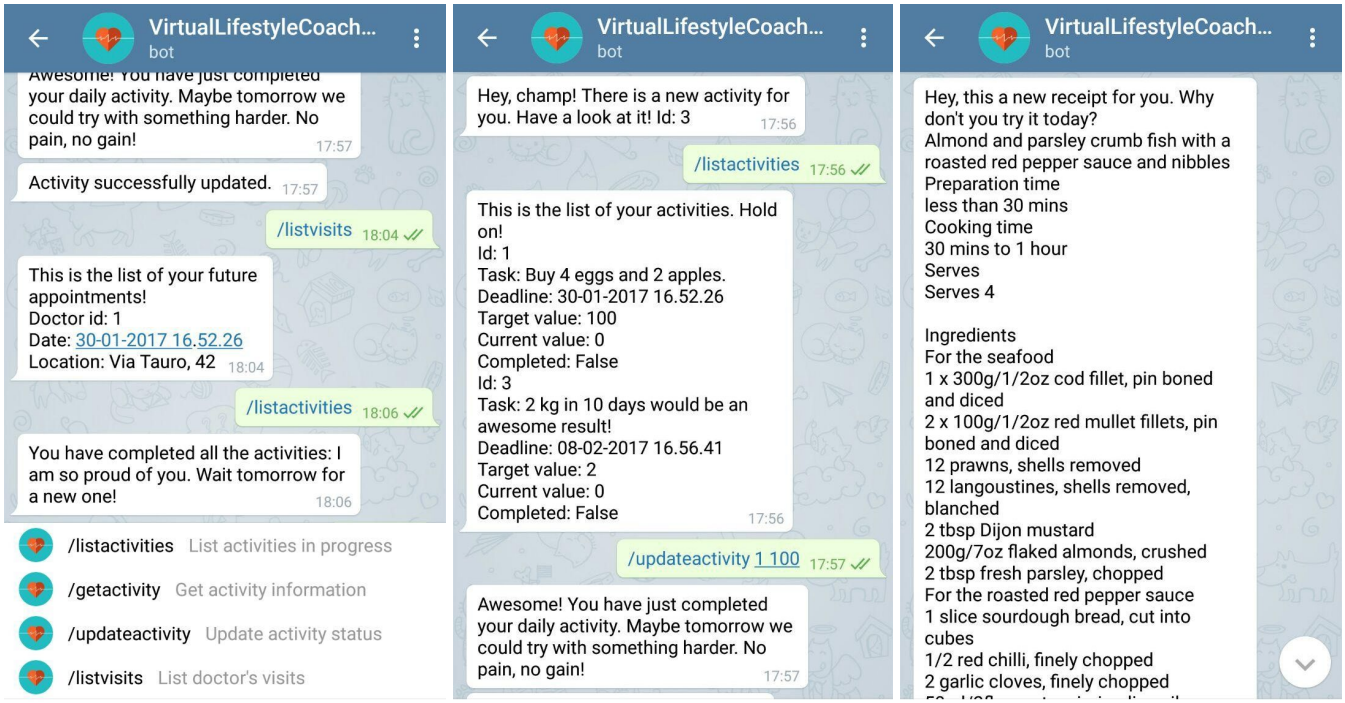
This report describes *Virtual Lifestyle Coach*, an application designed to help people attaining a healthier lifestyle. The following paragraphs describe both the functionalities of the application and its architecture, with a great emphasis on how components communicate and cooperate.
The application source code is hosted on GitHub (https://github.com/virtual-life-coach).

Application architecture

The application has a client-server distributed structure. The current version provides only one kind of client: a bot for Telegram, the free open-source instant messaging service. However, the application server is designed in such a way that other clients could be easily integrated. A new web application, for instance, could be developed without modifying the back end, only relying on the existing provided services. Actually, *Virtual Lifestyle Coach* can be used also by computer desktop, since Telegram offers both a web application and desktop clients for every major operating system.
The server is composed by different highly-decoupled layers cooperating to serve client requests. Each of them provides a specific functionality and is executed in a dedicated virtual machine offered by some cloud computing service (Google App Engine [GAE] or Heroku). A specific server layer exposes a RESTful web services for clients.

Client

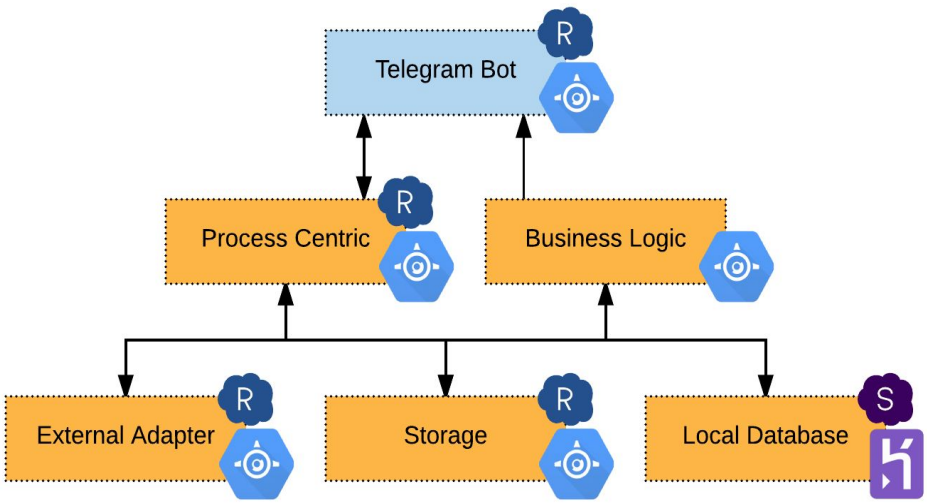
A Telegram bot provides a flexible and convenient way to interact with the application since it supports both the pull-based and the push-based interaction mechanism required by the use cases. In fact, users can actively contact the server by invoking bot commands or passively receive communications in the form of push notifications on their devices.
The bot is developed in Python by using the python-telegram-bot library and deployed on GAE. It also exposes a RESTful web services to let the server send messages to users.
Every day users receive a new activity at 8:00. Activities are user-tailored and designed by specialists. Users can manually update the activity progress or let other devices update it automatically. When the activity is completed users receive a congratulation message.
A fresh recipe by BBC is delivered every day at 11:45 to encourage users to eat an healthy lunch and, perhaps, have a walk to the local supermarket to buy the ingredients.
Finally, a quote is automatically sent at 16:00 for moral support in the afternoon.
The screenshots below show these functionalities, together with some command usages, such as the one to list future appointments with doctors.



Command examples *Activity interaction* *Random daily recipe*

Server

The application back end is developed in Java 7 and is composed by five layers, each one completely independent from the others, as it can be noticed by looking in the source code at the Maven pom files describing the project structure. In order to avoid code duplication, shared classes have been grouped in a common package. It includes, for instance, the classes defining the transfer objects [TO] that layers exchange to communicate. TOs are automatically un/marshalled from/to JSON by Jersey.



The diagram shows an overview of the application architecture. As it is possible to see, the local database layer is deployed on Heroku, while the others on GAE. The environment provided by Heroku is more flexible than Google's one, since it can be easily customized via buildpacks. For such reason, Heroku is the best choice for a component designed to persist data on a RDBMS via JPA and to expose a web service via SOAP. On the other side, GAE is tightly integrated with other Google Cloud Platform technologies, such as Datastore, a NoSQL datastore, and Cloud Endpoints, an API management system. The latter one is extremely useful to generate automatically REST APIs by annotating code and test them in a dedicated APIs Explorer. These features, along with other capabilities and technologies (application versioning, local development server, cron jobs, etc.), make GAE the best platform for the application.
Each layer is now described in details. Components will be analyzed bottom-up.

1. Local Database Layer

Platform: Heroku. **URL**: https://vlc-server-local-database.herokuapp.com/
Web Service API: SOAP. **WSDL**: https://vlc-server-local-database.herokuapp.com/ws/ldb?wsdl

This layer stores information about users, doctors, appointments, health measurements and activities into a RDBMS. Apache Ivy is used as dependency manager and build tool and EclipseLink as JPA technology. CRUD operations are executed on all the objects with a single generic class implementing the Data Access Object pattern. The result is a concise, elegant and flexible code that allows to easily expose all CRUD operations via SOAP. The client code has been generated from the WSDL endpoint directly into the common module to let every layer perform requests to this layer in a convenient way. An Ivy task is started after every deployment on Heroku to register the WSDL endpoint.

2. Storage Layer

Platform: Google App Engine. **URL**: https://vlc-server-storage.appspot.com/
Web Service API: REST. **API Explorer**: https://vlc-server-storage.appspot.com/_ah/api/explorer

This layer is designed to store entities of miscellaneous types that can be somehow considered external knowledge for the application. For instance, some BBC recipes have been stored into Datastore, GAE's NoSQL document database built for automatic scaling, high performance and ease of application development. CRUD operations are performed via Objectify and a RESTful API is automatically generated via Cloud Endpoints to let other components retrieve random recipes.

3. External Adapter Layer

Platform: Google App Engine. **URL**: https://vlc-server-external-adapter.appspot.com/
Web Service API: REST. **API Explorer**: https://vlc-server-external-adapter.appspot.com/_ah/api/explorer

Entities required by the application that are not persisted in any component are retrieved on-the-fly by this layer. In a broad sense, it is designed to be a toolkit to dynamically extract information from other services and web pages. Every external data source can expose its entities in a different format. The layer is expected to have an appropriate adapter for each of them. At the moment, the only external service involved is Forismatic, a collection of the most inspiring expressions of mankind. The website offers a RESTful API that is used by the layer to retrieve random quotes and serve them with its own RESTful API. More complex tasks, such as crawling, could be easily added by using GAE task queues or GCE.

4. Business Logic Layer

Platform: Google App Engine. **URL**: https://vlc-server-business-logic.appspot.com/

The application logic is encoded in this layer. At the moment the component is responsible of generating daily user-specific activities and to perform recurring operations such as sending new recipes or quotes to users. Cron jobs have been used to periodically trigger code execution. None of the aforementioned entities are stored in this layer, so they are all requested to the underlying components, processed and sent to the clients via push notifications, bypassing the process centric layer.

5. Process Centric Layer

Platform: Google App Engine. **URL**: https://vlc-server-process-centric.appspot.com/
Web Service API: REST. **API Explorer**: https://vlc-server-process-centric.appspot.com/_ah/api/explorer

This layer is responsible of handling user-initiated interactions, in fact there is a one-to-one relationship between the functionalities requested by the Telegram bot and those offered by this component. Simple client requests are immediately handled and dispatched by this layer. More complex operations, on the contrary, require this component to orchestrate multiple requests to the other server layers to gather all the necessary entities.