Service Design and Engineering 2019-20 University of Trento, 10 Feb. 20

# PROJECT REPORT

## ArenaKeys

a layer of abstraction over official and unofficial videogame keys reseller

**Team Members**:

- Davide Tessarolo (211459)
- Devis Dal Moro (211457)
- Mouslim Fatnassi (211421)

**Repository**: https://github.com/texdade/SDE_project

**API documentation**: https://gamekeysarena.docs.apiary.io/#reference

**Website gamekeys-arena**: http://ec2-34-245-97-253.eu-west-1.compute.amazonaws.com:3000/ (if for some reason it's not up, please contact us)

## Introduction

This document comes to help in order to summarize what's going on behind the system we created, but without giving any actual detail: we're happy to describe them and clarify any doubts during the presentation.

We wanted to build a system that could help users to find games of their interest at the cheapest price. Using the means offered by Google and Steam SSO, we were able to authenticate the user, so that he can build (or import from Steam) his "wishlist(s)" to check over time the prices of items of his interest and eventually be notified when one or more of the games between them have a reseller which is able to provide it for a price below the one prompted by the user.

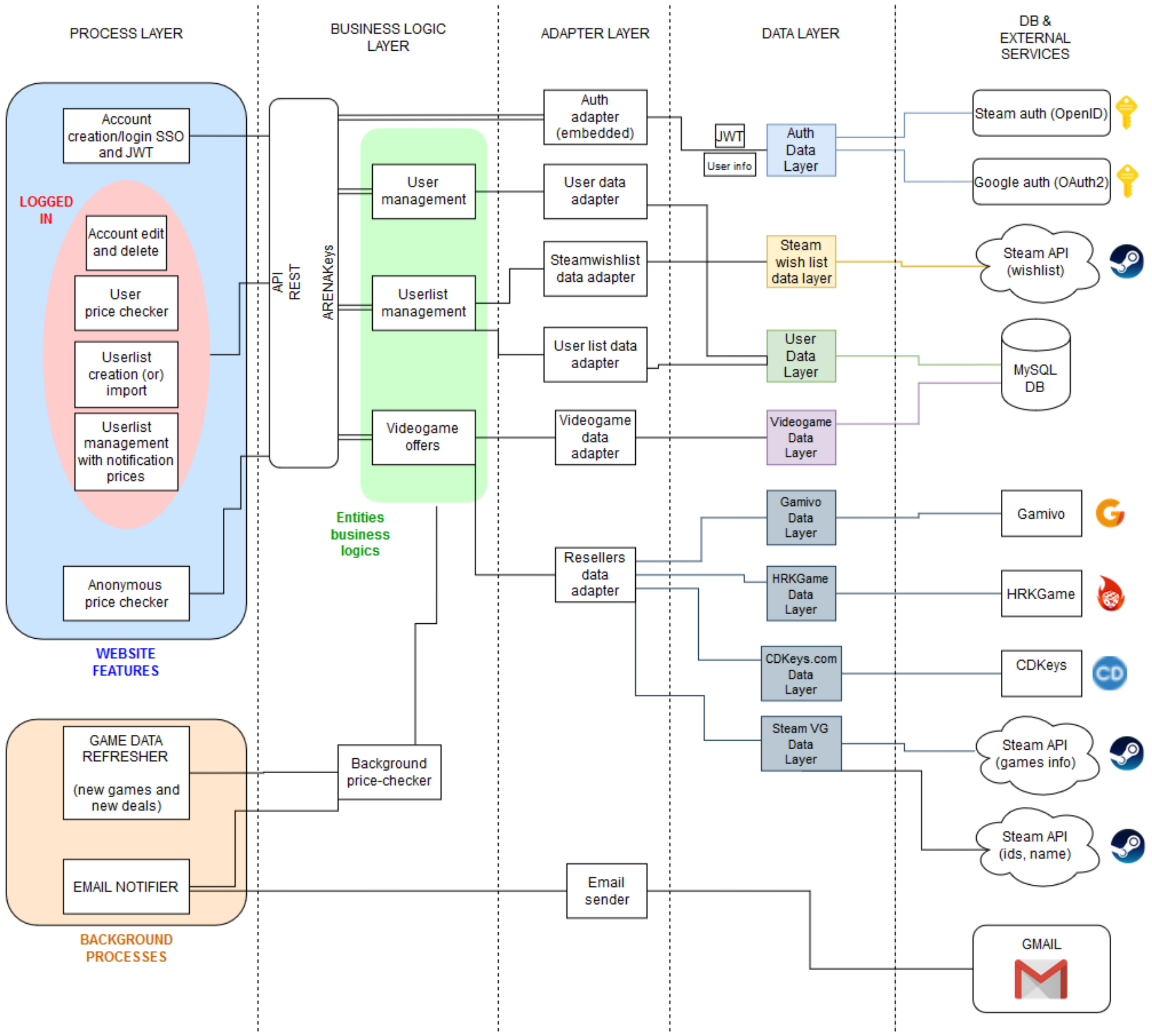
The system is supposed to work also for unregistered users (e.g. not-logged), just without the part of wishlist and notification.

## Flow of operations

We'll try now to explain briefly how the main processes which are behind the working of the system operate. Steam is the official reseller which mainly offers two calls to explore its database of videogames: one which just gives you a dump of pairs (*steamID*, *name*) which has brutally persisted in our database in the first place and it's constantly checked for new games and another which requires the *steamID* of a game and gives you information and the price as well. For what concerns the third-party resellers which we integrate in the system (specifically Gamivo, HRKGame and CDKeys), they offer you their whole dump (with games and best offers) which is not that massive as the one offered by Steam, hence it's possible to cache them in memory at the very first request and refresh it once every 3 hours or so (requiring more time just for that one).

Every time a game is requested, the videogame data layer checks on the database the DateTime value of when that game was last updated: if two or three days have been passed or the game has never been requested before by anyone we use the resellers module to contact (or explore the recently memory-cached data) their different web APIs and find interesting offers which might interest wrt the request made in the first place. Unfortunately, the resellers don't always provide the steamID in their database, so most of the time we have to be happy with a heuristic matching (by checking on the similarity between the names of the listed games) when the exact one cannot be performed.

A more detailed architecture of the system can be seen here (with a few additional comments following it):



A JSON Web Token is going to be produced after the authentication and has to be encapsulated in the header of the request in order to get responses as "logged" user. The user data, which can be recovered through the authentication process (except for the internal id given by ArenaKeys itself), as well as the userlist ones, are persisted in the database and recover through apposite data layers.

The user list can be created as empty or directly imported having the steam id of a user, offering a sort of "commodity" to directly check all the prices for a subset of selected games. One of the background processes constantly checks the prices of the games the user had previously "ticked" by tagging them with a price. More specifically, if one or more of the offers for the game will be discovered below the price prompted by the user, the system will use Gmail to notify the user (assuming there is a valid email within the data of its account).

The other background process checks for new games and for updating the prices of the ones already present in the database, but that haven't been asked in a few days.

By the means of adapters, everything is filtered down and formatted to our object(s) of interest[1], hiding some underlying differences, while the business logic handles the whole flow of operations to manage a single request (e.g. getting the offers for a game could imply multiple fetching of data from the databases and from the resellers' API as well which has to be orchestrated in a specific manner by the central engine).

The higher-level processes just use and combine some business logic – level operations which are also exposed through web REST API.

## Used technologies

Mainly the app is built using nodejs for the backend services with express as the main backend framework, using middlewares like body-parser, cookie-parser to help us create the endpoints (wrt. to all the different and available http methods: GET, POST, PUT, DELETE, …) and elaborate the different requests.

The resellers have been mainly contacted through their web APIs and this has implied to parse XML and JSON documents with proper libraries found on npm. The data provided by them and of our users' interest has been persisted on a mysql database, which is constantly refreshed with updated offers and info for what concerns new available games[2]. To boost up performances a little bit, by avoiding making multiple times the same API call, when possible, we cache data also in memory using apposite node modules to perform these kinds of operations. In any case, it's possible to see all the libraries and external modules we fetched and install using npm, just by exploring the "dependencies" in the package.json file which can be found in the root of our repository.

Passport modules have been useful for dealing with Google OAuth2 and Steam OpenID, returning a JSON Web Token to authenticate the user and handle its "session".

Finally, we use ejs as render engine for express to dynamically generate some webpages for the website which uses our APIs. Part of the high-level logic which does the latter is directly embedded in the client-side scripts (mostly jQuery, hence Javascript is the main character of our project).

The APIs documentation has been written in Swagger2.0 and posted on Apiary, while the code versioning has been performed using git and Github.

[1] Which can be found in the linked documentation
[2] Honestly, that was our target and "code-speaking" we reached it, but we have to pay to have a hosting service which can constantly (day and night) run the procedures to do so, thus we're going to show you in our presentation that the background services for refreshing games (and sending you notifications) effectively work, but we're not going to actually put them to "regime" for the time being