UNIVERSITÀ DEGLI STUDI DI BERGAMO

Software engineering project

Matias Negro 1065808, Daniele Greco 1065570

1 Introduction

The project goal is to build a software system that helps the Steam user-base in buying the desired products at the desired price. We though about this as an external application that saves the user preferences as couples with the scheme $[product, desiredPrice]$. When the product price is under the $desiredPrice$ threshold, the application notifies the user with an OS notification, supposing that the machine running the application is connected to an internet provider. Since the Steam user base uses a lot the personal computer for checking the site (Steam is a games market for PC), we chose to let the application run on Windows, MacOS and Linux platforms. The project components are: Matias Negro (matr. 1065808) and Daniele Greco (matr. 1065570)

2 Process Model

The project life-cycle is agile. We used the Rapid Application Development (RAD) technique for the development of the software with time boxes of one week. A common technique used with RAD is MoSCoW, where we decided that:

- Must have:
  - The ability to retrieve the Steam product list
  - The ability to select the products from the retrieved list
  - The ability to store internally in the machine the list of selected products
  - A notification multi-platform (Win, Mac, Linux) system
- Should have:
  - A reactive Graphic User Interface (GUI)
  - A logo
  - The tray management for all the compatible OS
  - The window management
  - The ability to keep the software running in background
  - A search system for the Steam product list
- Could have:
  - A good-looking reactive Graphics User Interface (GUI)
  - A registration system to keep the data not just in the machine but also online
  - A complementary application for mobile (this one is related to the registration system)
  - A system to retrieve more info from the web about the products
- Won't have:
  - A review system for the products

  - A suggestion system for the user
  - Data collection from the user

The start of the development starts the 10/11/2022 and assume to finish during the first days of December. For the requirements we used the MoSCoW prioritization model.

3 Project organization

For the team organization in the project we used the "Chief Programmer Team" model.

- Chief Programmer: Matias Negro
- Assistant: Daniele Greco
- Librarian: Daniele Greco, Matias Negro

As end users we use the development team itself and external people that use, for a short period of time, the system as time boxes passed and new features are implemented. Some knowledge will probably miss and will be retrieved by web resources (as the DB implementation in dart/flutter).

4 Standards, guidelines e procedure

The programming language mostly used for the project will be Dart, integrated into the Flutter framework. This framework compiles Dart code in native language for the system OS before the build is done, this means that other programming languages will be highlighted in the repository. The second main language used is SQL, for the internal DB interaction. We will follow the Dart/Flutter standards. The data retrieved from the SteamAPI is in JSON format. The code editor used by the team is Visual Studio Code. For the repository interaction we will use the "git" command line.

5 Management activities

For every time box the chief programmer will send the assistant the list of to-do's done and will explain him how the new features were implemented. Other than that, a new working build will be shared with the external people used as sample of the Steam user base. After a short time usage there will be an informal interview with these external people to get a review of the current version of the system.

6 Risks

The main risks of the project are changes in the steamAPI, especially regarding the information retrieve. This means that there must be a periodic check of the API documentation to get noticed of incoming changes in formats, url's ecc.. Another relevant risk is related to performance, since the system can run in background errors can consume resources more than what is actually needed.

7 Staffing

The staff components are Matias Negro (matr. 1065808) and Daniele Greco (matr. 1065570).

8 Methods and Techniques

The requirements specification method is the IEEE830 standard. For the testing techniques we will use the Flutter internal testing system. Since there is no actual "customer" we will use a COTS methodology for the component selection and definition. For the documentation we will use the "dartDoc" package.
Software components:

- Visual Studio Code
- GIT
- Trello.com

- SQLITE2
- Draw.io
- StarUML

Dart/Flutter packages:

- cupertino icons: v.1.0.2
- test: v.1.21.1
- dio: v.4.0.6
- multi split view: v.2.1.0
- local notifier: v.0.1.5
- path provider: v.2.0.11
- sqflite: v.2.1.0
- provider: v.6.0.3
- html: v.0.15.0
- cron: v.0.5.0
- sqflite common ffi: v.2.2.0+1
- window manager: v.0.2.7
- tray manager: v.0.2.0
- dartdoc: v.6.1.1
- mockito: v.5.3.2
- connectivity plus: v.3.0.2

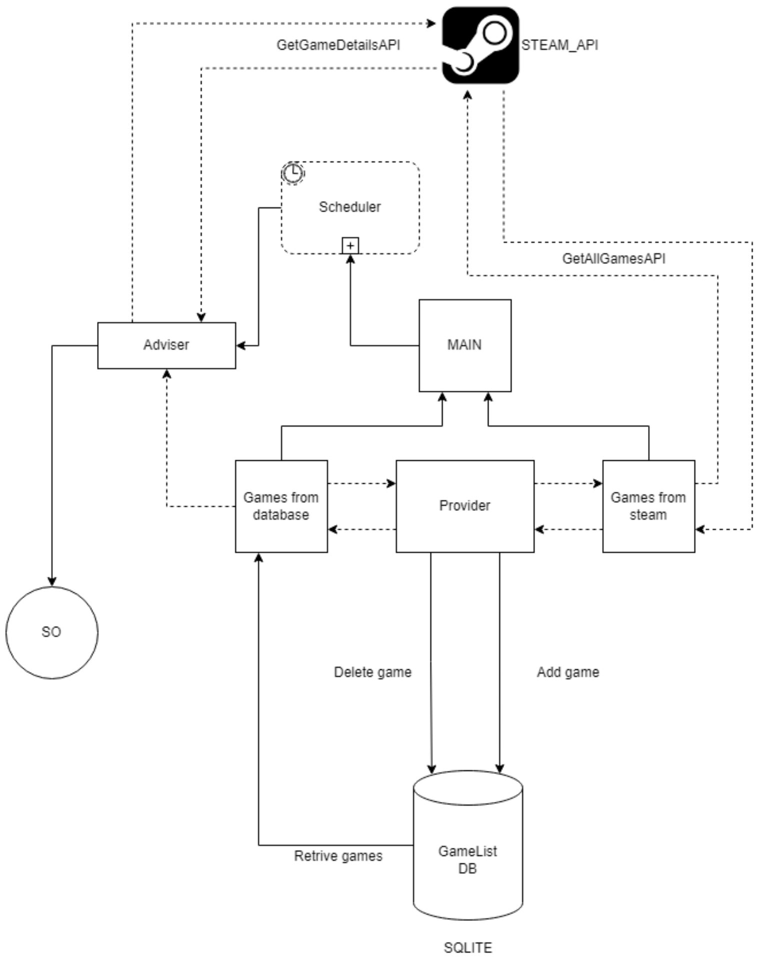
The class diagram will be generated by the dart package dcdg v.4.1.0.

9 Quality assurance

For the quality assurance we will follow a limited version of the IEEE9001.

10 Work packages

For the project modules we used the following scheme:



11 Resources

The resources needed are a compatible Operative System:

- Windows 10 / Windows 11
- MacOS Big Sur / MacOS Monterey / MacOS Ventura
- Linux

and a personal computer to run the application. The application will not be hard to run, so even a low-end personal computer is enough.

12 Budget and schedule

The main constraint is time. The project should occupy 70 hours per staff member. For this project there are just two of us, so this time guideline will probably not be enough. The main focus will be to deliver in time (first days of December 2022) a working system with good documentation.

13 Changes

Since we use an agile development, changes will be present for every time-box. Probably some changes will be pushed even in the mid of time boxes during development if the added feature is complete, due to time optimization. For the update management we use GitHub with different branches that needs the chief programmer approve for merge with the Master branch.

14 Delivery

The final delivery date will be in the first days of December. We are not able to say the precise day yet.