PeakLife Report

Group 35

Delft University of Technology

PeakLife

TU Delft

by

Pietro Campolucci (4645979),
Laura Muntenaar (4554213),
Luke Chin A Foeng (4609972),
Mateusz Zaremba (4673603),
Matteo Rebosolan (4664973)

TU Delft

Contents

1 Introduction

The purpose of this software is to give the user insight into the quality of life within any country on the planet. The quality of life is determined by taking into account different factors from weather, pollution, crime-rates etc. The number of factors taken into account has been limited to five. These factors are weighted against each other based on a predetermined algorithm. The output of this algorithm is a score on a scale from zero to ten, with ten being the highest possible score a country could attain.
This is the core concept of the software. This software will also allow the user to customize the outcome of the algorithm using age and gender user inputs. These inputs would filter the results to limit what the user sees to what would be the most helpful to them. This would allow the user to determine the best place to live for them, as the output of the algorithm and filter are three scores. These three scores are the general score, the user tailored score and the total country score.
These scores could be applied to a variety of situations. These include helping a user decide travel plans, a new country a user would like to move to, or just to check the overall situation within a country for people of a specific demographic living within a country. A link to the site is provided below as well.

GitLab link: `https://gitlab.ewi.tudelft.nl/ti3115tu-2019/group-35`
Screencast link [2]: `https://youtu.be/mm2NiULRueA`

2

Software Development Process

For the project to move at a constant pace and keep everybody engaged, a project plan had to be set up. The project planning was done in an agile way, primarily focusing on scrum. Scrum was chosen as it has short periods of time, the "sprints", after which it releases a product increment. So results could be seen fast and progress would be build-able and dependable on previous product increments. Figure 2.1 shows the time span of our scrum cycle.

2.1. Sprints

Figure 2.1 shows the time span of our scrum cycle. The sprint duration was 1 week starting and ending on Monday. This was chosen as initially team meetings were on Monday and it was debated that it would be nice to have the meeting with the TA half way through the sprint if advise or guidance was needed.

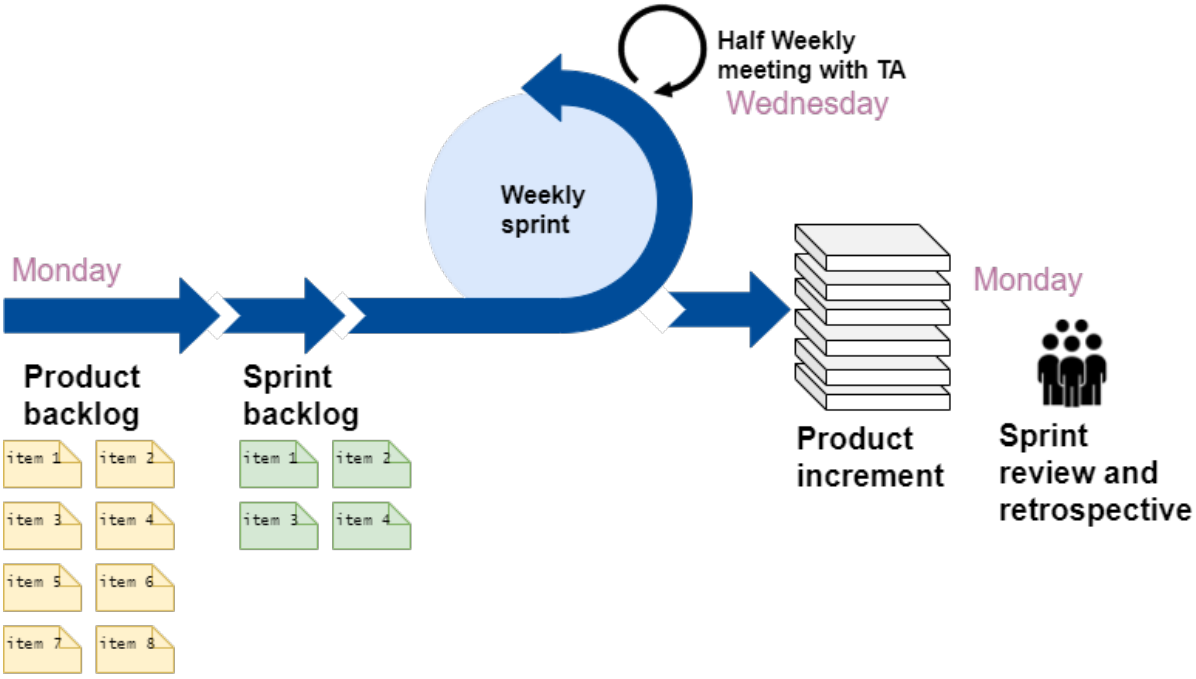

Figure 2.1: Our weekly scrum planning

Our kanban board was a digital google spreadsheet which can be seen in figure 2.2. In column A, the backlog was created per sprint, in column B and C the tasks are assigned among the team members and in column D the progress of that particular item was tracked. Column F then gave the option for team members to write comments on what they found difficult or the results of what they did. This column was also for items that needed clarification on the backlog.

At the Wednesday TA meeting, the team could discuss the current status of their item and resolve the preliminary problems with the help or approval of the TA. The next Monday, the finished items were merged into the master branch and unfinished items were put on the next backlog.

	A	B	C	D	E	F
1	Sprint 1 09/09/19-16/09/19					
2	Features backlog	Who	Who second	Progress	Deadline	Comments
3	UML requirements of lay out	Matteo			16/09/19	
4	Project layout	Laura			16/09/19	
5	Dropdown menu	Luke			16/09/19	
6	Find good indicator	Pietro	Laura		16/09/19	
7	Theorize future algorithm	Pietro			16/09/19	
8	User accounts	Matteusz			16/09/19	
9						
10	Sprint 2 16/09/19-23/09/19					
11	Feature backlog	Who	Who second	Progress	Dealine	Comments
12	Finish dropdown menu with country codes	Luke			23/09/19	
13	Connect dropdown and api's in django	Laura	Pietro		23/09/19	
14	Optimise api indicators	Pietro			23/09/19	
15	Add list in settings of the api endpoints	Pietro			23/09/19	
16	Update git with merge request	Luke			23/09/19	
17	Review the merge request	Matteusz			23/09/19	
18	Look into making a django comment	Matteusz			23/09/19	
19	Create website mockup	Matteo	Laura		23/09/19	
20	design code review				23/09/19	
21	Create Dictionary with Indicators	Pietro			23/09/19	
22	Deliver first algorithm result online	Pietro			23/09/19	
23					23/09/19	
24					23/09/19	
25	Sprint 3 23/09/19-30/09/19					
26	Feature backlog	Who	Who second	Progress	Deadline	Comments
27	Design home screen in Marvel	Matteo	Laura		30/09/19	
28	Implement home screen in Django	Matteo	Laura		30/09/19	

Figure 2.2: The initial sprints and deadlines

2.2. Sprint Planning and Sprint Review

Scrum is a precise and difficult agile technique to initially work with. As no one in the team had done it before, the roles were not clearly divided. There was no one product owner, but the entire team together created the product backlog and prioritized the items on there. Every Monday, everyone presented the result of the sprint together where after things that were considered done would be evaluated and checked if it met the acceptance criteria. This was done in person as well as in gitlab. The person that had 'finished' a sprint item, would post a merge request where after at least two other people had to agree with his/her code.

After the sprint review on Monday, the meeting would change to the next sprint planning. The team collectively decided which items needed to be put into the sprint backlog and how much could be taken on. One person would write and document the decisions made and create the sprint backlog of that week where after the sprint could begin.

2.3. Sprint Retrospectives

2.3.1. Week 1.1

The first week was the week where every team member had to get familiar with Django. The Django website provided a very thorough tutorial so every member needed to follow this.

Positive Points	Negative Points	Action Points
Everybody reached a sufficient level of django knowledge	Not everybody was able to do the entire tutorial which may cause delays in the future	

2.3.2. Week 1.2

Week two was the start of the development of the minimal viable product(mvp). For this, requirements were set up, user stories were created and flow and activity diagrams were designed. Next to this, an initial start with creating Django redirect pages and how to pass arguments was made.

Positive Points	Negative Points	Action Points
Django redirect pages and passing arguments worked, also dropdown menu worked	The UML user stories and flow/activity diagrams were not up to the desired level so they need to be redone so they are added to the backlog of the following sprint	Add UML to the backlog

2.3.3. Week 1.3

In week three, the development of the minimal viable product was continued by creating the new UML user stories/MOSCOW. Next to this, the team started to work on creating a mock design for the site using marvelapp and a simple algorithm was designed.

Positive Points	Negative Points	Action Points
The minimal viable product was as good as done, a country could be selected and after selection, a score was displayed.	The mock design of the site was unfortunately not done in time.	Add finishing mock design to the backlog

2.3.4. Week 1.4

At the beginning of week four, the mvp was done so it was decided to pace the project a bit slower and allow for more testing and prioritize code quality. In addition, the mock design was finished and ready to be actually coded.

Positive Points	Negative Points	Action Points
The minimal viable product was done and the styling of the site was decided upon	The code didn't look that good and uniform.	Code quality and refactoring need to be prioritized more.

2.3.5. Week 1.5

In week 5 it was time to start implementing the design of the web pages and think about other features that could add upon the mvp. It was decided that the algorithm would be expanded to include age ranges and genders to ensure a more personalized score.

Positive Points	Negative Points	Action Points
Decisions were made on what the next big step was.	Styling the webpage took a lot longer than expected.	Styling needs to be put onto the backlog for next weeks sprint

2.3.6. Week 1.6

The algorithm was done at the beginning of week six, the design of the web pages was done in week six. As well as the creating an about page as well as drop down menu's for additional user input.

Positive Points	Negative Points	Action Points
The styling of the site was done.	Testing still needed to be done and proved to be difficult	Testing is put onto next weeks backlog.

2.3.7. Week 1.7

In week seven, testing was done. The algorithm and the fetching of the API's was done. Next to this the styling of the site and the contents of the about page were tweaked. Next to this, the report plan was introduced.

Positive Points	Negative Points	Action Points
The site was fully functional and tests seemed promising.	There was not enough progress with the report.	Writing the report was prioritized more on the backlog.

2.3.8. Week 1.8

Almost everything was finished, but at the end of week 8 our site did not seem to work. Turned out the world health organisation had changed their query logic so the fetching code needed to be changed at the last minute.

Positive Points	Negative Points	Action Points
Everything was done and the report was close to being done.	The WHO changed their query logic so this needed to be changed asap.	Report is prioritized.

2.4. Retrospective

Over the course of the project, the team got better in deciding how much work can fit into a sprint and what exactly was considered done. Where at first the items that where done functionally wise where immediately merged, at the end, different branches where only merged if the items were done functionally and if they were reviewed or even refactored.

3

Requirements Engineering

The most important requirements come directly from the web app's concept of a service that helps the user see which countries have the conditions to live a healthy life. These are the requirements in the "Must Have" category in the next sessions. More non-necessary requirements came from the implementation of user stories. These requirements, while not directly related to the core purpose of the web app, should provide the user with a more complete experience (e.g. feedback based on the user's characteristics). The requirements were also chosen on the basis of the software development skills of the group, for instance which software/programming languages were to be used.

The requirements changed during the course of development, as requirement in the could have/should have category of the Moscow list were decided to be implemented. These new requirements were added to the weekly Scrum backlogs as the team went ahead in the development process.

3.1. Showcase Material

The team used various methods to list the functionalities desired for the software, starting from MOSCOW and ending with activity diagrams.

3.1.1. The MOSCOW method

The team made use of the MOSCOW technique to better identify the most important features of the app and organize in a more efficient way the development process to follow. The list is therefore reported below:

- **Must Have:**
  - GUI that allows the user to select any country.
  - A wide storage of health related information about any selected country.
  - A page showing the current score of the country.
  - An algorithm that translates the values from the storage in a more understandable 10 points grade.
- **Should Have:**
  - GUI that allows the user to insert his gender and age details.
  - A wide storage of information tailored for each combination of age and gender of the user.
  - An algorithm that weights each country information respect to its importance.
  - An algorithm that gathers general information and user tailored information separately and translated them to a 10 points score.

- **Could Have:**
  - An interactive map instead of a drop-down menu as country selection location.
  - A page that compares the scores of the selected country with the scores of the current country where the user is located.
- **Won't have:**
  - An automatic detection of the best country for the user.
  - The possibility of printing the score or saving it to PDF file.

A more schematic visualization of the list is presented in Appendix A.

3.1.2. User stories

- User wants to check which country has the best living conditions for a 34 year old man that wants to relocate. User enters the web app and clicks on the "Do the test" button. The user is redirected to the country selection page where he inputs the country he wants to look up, his gender and his age and then clicks on "Get your country score!". After that the user is shown the result page, which includes a score based on the user's age and gender, an overall score for the country and a third score that represents a weighted average of the first two scores. The scores are shown on circular diagrams. The user can then click on either "about" or "country test" on the nav bar to go back to the home page or repeat the test, respectively.
- User wants to see the overall living conditions of a country, irrespective of age or gender. The user enters the web app and clicks on "Do the test". Then, in the country selection page, the user chooses a country while leaving the age and gender fields intact and then clicks on "Get your country score!". The result page will then only show the country general score. Again, the user can go back to either the home page or the country selection screen by clicking on the buttons in the nav bar.

3.1.3. State and Activity Diagram

As the reader can see from Figure 3.1, the app will bring the user through three different pages during the usage. Once the app is initiated, the user is redirected to the main menu, where the he can read about the app and its functionalities. In addition, he will be able to start the test to see which country will suit him better. During the test, the user will be redirected through two pages: in the first one he will insert his details and the country name, while in the second one he can visualize the results obtained. In both pages is possible to go back to the main menu.

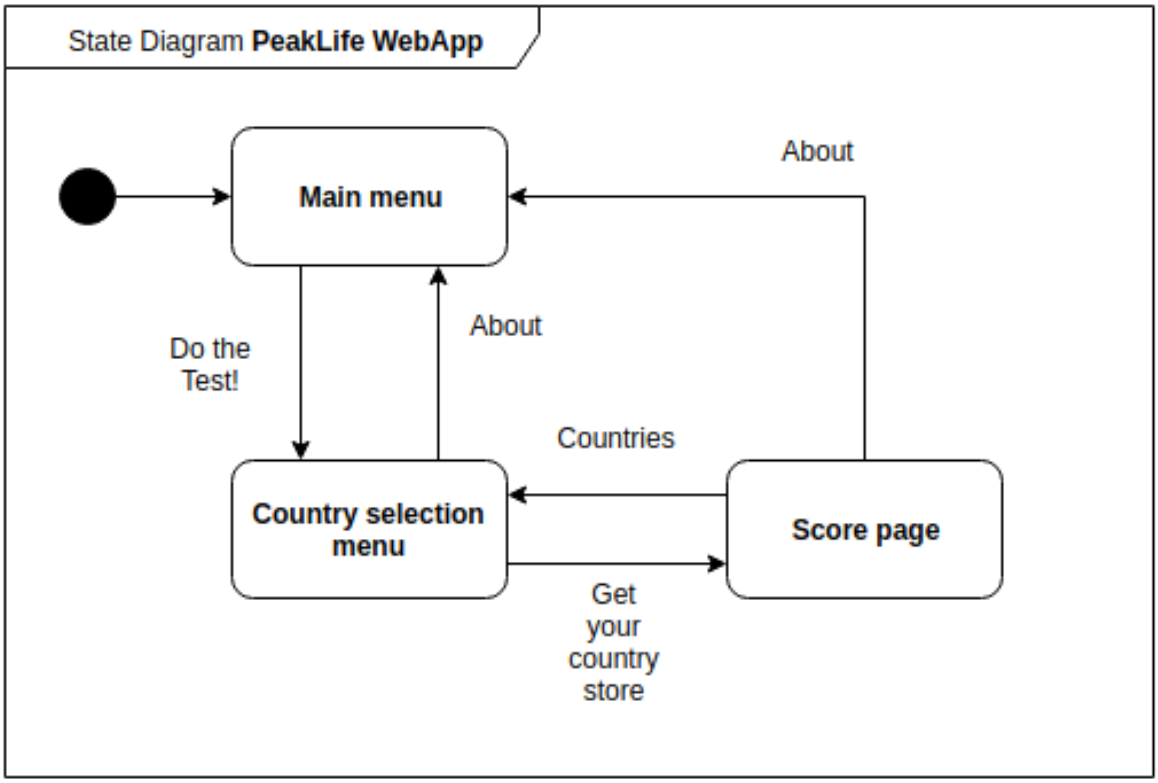

Figure 3.1: State Diagram of the Web App

The whole app rotates around the algorithm that, once the user inserts his details, will retrieve the required information about the country by means of online API's. The script will also be able to translate the values obtained into a 10 points grade, in order to be able to generate single scores representing more different topics. The process can be observed in form of an activity diagram in Figure 3.2

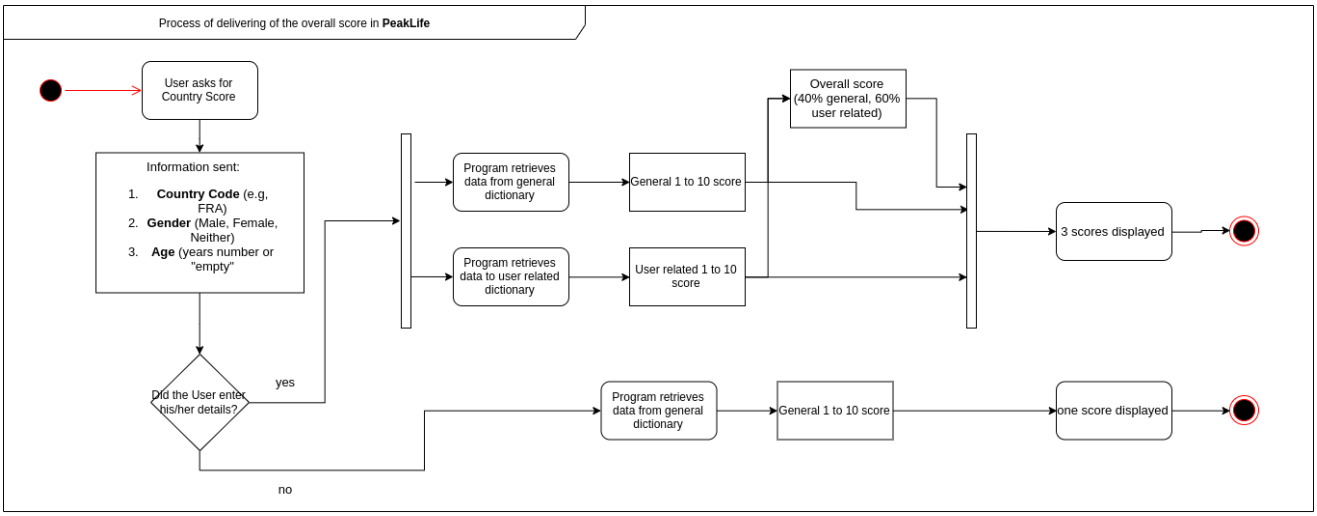

Figure 3.2: Activity Diagram of the main algorithm user to provide the country scores

4

Software Architecture

4.1. Design choices

In order to develop a web app, there are quite many options available only for Python. Since it was not considered to have separate Front-end and Back-end server, because project was supposed to be written in (mostly) Python, the only option was the Server Side Rendering (SSR). In order to choose a package, a few architectural decisions have been made:

- **Server Gateway Interface**. For this app, asynchronous behaviour is not required, since there are no chats, no websockets and any other need for server to push the data. Also, considering team experience and simplicity, Web Server Gateway Interface (WSGI) was chosen instead of Asynchronous Server Gateway Interface (ASGI), therefore also communication protocol was set to HTTP.
- **Static or dynamic**. In order to allow dynamic behaviour, JavaScript code is required, therefore the webpage will be static.
- **API**. Multiple data sources are available online for free. We have chosen the World Health Organisation API [**?**], since its database is large enough to handle all data related requirements, while still being a verified source.
- **Database and deployment**. For such application, the database is not necessarily required, since the data we operate on is fetched from online sources. To improve the user experience, we have decided to store some of the data in our own database with prefetched the results. For database, PostgreSQL was preferred due to seamless connection with a deployment tool - Heroku. Heroku was chosen for deployment, as it operates with git and has a free tier plan.
- **Web development framework**. Based on previous points, it was concluded that Django [1] is the best fit. While it is not the lightest framework (as opposed to i.e. Flask), it has great Object Relational Mapper which handles the logic with database. Moreover, Django provides the SSR options, and it has been proven successful in similar projects.
- **Layout framework.** Bootstrap was chosen as it provides great amount of components that can be customized to the specification.
- **Environment.** The programming stack consist of Git for the version control, Black for the autoformatting, Flak8 for the compliance with PEP8 (ignoring E203, E266, E501, W503, F403, F401) and Pipenv for the environment management.

4.2. Model-Template-View+Controller

Django is a framework which does not follow the standard logic of Model-View-Controller but Model-Template-View + Controller. In case of the Peaklife app, the only model is Country that stores prefetched data. This

model also handles API calls to WHO API via requests library. There are 3 different templates written in HTML, which are controlled by the views (so called View+Controller).

- **Home View.** It is a fully static view, whose only purpose is to inform user about purpose and redirect to Country Selection View. It has to handle only GET method.
- **Country Selection View.** Purpose of this view is to let the user input data by selecting country and optionally adding personal information. Therefore, this view has to handle GET and POST requests and be capable of redirecting the user to the right indicator view, while forwarding his input data.
- **Indicator View.** This view is only informative but has to handle which information it needs to show and fetch required information. It only handles GET method with query parameters and country ID.

In general, the code flow is as follows:

1. User goes to the webpage (sends GET requests).
2. urls.py selects view based on url.
3. Selected view controls and displays the template with data based on request method and provided information.

4.3. Internal structure

Structure of project is following (mentioning only most important files for brevity):

- **Main directory**
  - Config files. Contains project configuration (git, gitignore, black, flake8 config files, env files and so on)
  - **src.** Contains source code of app
    - ◇ **indicators** Stores files specific features.
      - · **templates.** Contains HTML files
      - · **admin.py** Handles the model registration for admin page
      - · **models.py** Contains python representations of data stored in database
      - · **urls.py** Used for directing requests to views, specific for features
      - · **views.py** Stores views
    - ◇ **peaklife** stores setup files
      - · **settings.py** Contains general, app-wise settings
      - · **urls.py** Used for directing requests to views, app-wise.
    - ◇ **static, staticfiles** directory to store static assets for deployment
    - ◇ **manage.py** file that handles running web server

4.4. Relation to WHO API

We have taken a risk to assume that World Health Organization will provide almost constant uptime of their servers. We do know that in the future, a solution should not rely on one single point of failure, especially when this point of failure is an outside our scope (we do not have any influence over WHO API). Recently we have discovered that WHO is moving their API to the different service and/or endpoint. This might cause some unexpected downtime or slow response when moving from country selection to the indicator view. We are aware of that. As of now, the endpoint we are using usually works. In the event of migration to the newer system and dropping support to old one, our code was designed to be modular enough to switch to new API by changing only one setting.

5

Software Testing

The current software pipeline is mostly written in Django, which removed the need of performing system testing due to the already precise debugging window offered by the package. The team was however capable of recognizing and isolate individual functions and classes that did not require to be implemented in a Django environment to work, and design unit tests for them. The main example is the testing environment developed around the class of functions that retrieves data through API's and combines them to generate the overall score that the user is demanding.

The tests have been set up to show if the code actually fails (or it returns what is expected), if 90-100% of line and branch coverage is achieved and last if the code is actually written well or needs some refactoring. Figure 5.1 shows the structure applied for example to the code used grade estimation.

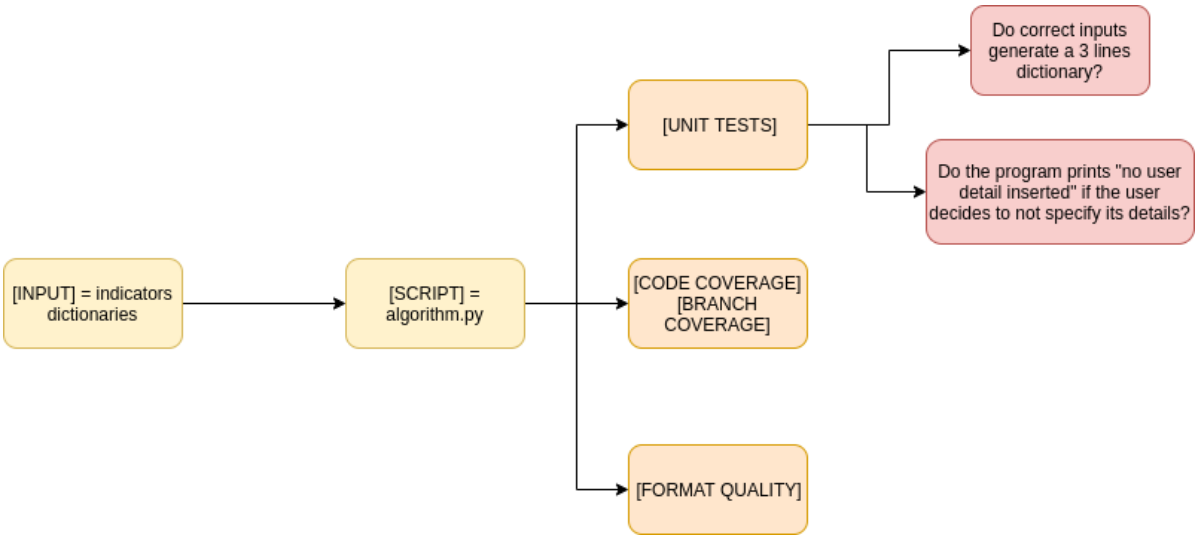

Figure 5.1: Test plan for Algorithm.py file

5.1. Coverage

Figure 5.2 presents a line and branch coverage report of the two files that contribute the most to data retrieval and score publication. As can be seen from the report, a coverage of 94% is accomplished, which is considered highly satisfactory.

Coverage report: 94%

Module ↓	statements	missing	excluded	branches	partial	coverage
src/indicators/algorithm.py	58	3	0	26	1	95%
src/indicators/range_detection/range_function.py	26	1	0	4	1	93%
Total	**168**	**86**	**0**	**42**	**2**	**94%**

coverage.py v4.5.4, created at 2019-10-21 10:20

Figure 5.2: Coverage report for algorithm

5.2. Overview of Tests

The script `algorithm.py` has been chosen as the only script worth to be tested because of the relative testing speed and, more importantly, its relevance in the score generation. Since it is easy to misinterpret the query for data retrieval, unit tests were performed in order to check that indeed each category (gender, age...) is delivering the right value. Table 5.1 gives a more complete overview of which tests are run.

Table 5.1: Overview of all tests, automated or manual, written and/or executed by the team

Short Test Description	Type	Team Member
Assert that the code returns a 3 lines dictionary	automated	Pietro Campolucci
Assert that the code returns as "Tailor Made Score" a "no user details inserted" when the user does not provide and personal details	automated	Pietro Campolucci
Assert that the code returns as "Total Country Score" a "no user details inserted" when the user does not provide and personal details	automated	Pietro Campolucci
Assert that the code returns different user tailored scores for same country and age but different gender	automated	Pietro Campolucci
Assert that the code returns different user tailored scores for same country and gender but different age	automated	Pietro Campolucci

6

Reflection and Adaptation

6.1. Pietro Campolucci

My way of thinking before joining this course was the following: "Software development does not require a set up of all that planning structure that is instead a must when you build , together with a team, a wing or a landing gear". Immediately after a few lectures, I realized I was wrong! A structured plan in software development is quintessential for a functioning, high quality product: the situation can get very messy very quickly and huge losses of time can occur if not cautious enough.

This project allowed me to fully realize that what the lectures communicated were essential tools for the realization of a good product. After two months, I am already reasoning in an agile way and the results are remarkable also in my side projects.

I am willing to meet new people in the future with such background to amplify my skills and hopefully get more efficient, especially in debugging.

6.2. Matteo Rebosolan

Before starting the minor program I had no idea that computer science projects could be so structured. Like Pietro, I was surprised by the depth that software engineering/architecture could reach and its importance in the project. My favorite part of the course were the development methodologies, such as Scrum and Agile. I really enjoyed learning how to work like a developer, also because I feel like these work philosophies can also be applied outside of the computer science domain. On that node, my goal for the future is to keep working on my Scrum/Agile skills, since I personally think that those methodologies have great potential for increased effectiveness.

6.3. Luke Chin A Foeng

This course has helped me better understand how code is structured and the whole process as to how a piece of functional software is created. Before the start of this course, I had a very minimal understanding of how git worked. Now that I have had to use this program throughout this course, I am now extremely confident in how git works.

Since we used the methods and software development processes explained to us in the Software Engineering Methods class, I was able to not only understand how to use Scrum and Agile, but I was also able to understand why they are used in software development. These methods actually helped us efficiently create our software.

The next steps I will take in order to become a great software engineer would be to continue to implement the work flows used during this course into any other projects that would require coding. I will also continue

to learn new python modules and maybe develop some software on my own using these methods in the near future.

6.4. Laura Muntenaar

Before this course, in projects where git was used for code structure, there were always a lot of merge conflicts. There was never really a structure and planning went horribly. But this course gave me an insight in how code projects should go. I never had a project that went this smoothly, there was a clear structure, create an issue, create a branch and work on your own part. The progress is steady and consistent and breaking the project up in sprints where I get to decide how much I can take on was a breath of fresh air. Another thing that was really helpful was the MOSCOW. Bare boning your project and ensuring quality at the lowest features before adding more features ensured precise work and early on code checks by other people.

6.5. Mateusz Zaremba

I already knew about Agile and Scrum before but this project let me realize an important factor - the team. Before I was mostly coding on my own, at most my development team consisted of 2 people, usually focused on completely separated tasks. This project and lecture material showed me how different is it to work with medium sized group of talented people, and apply Scrum techniques in real life and problems. I have learned how to manage development, split big problems into smaller ones, coordinate connecting separate solutions into one feature. What I was missing from the project was the simulation of full Scrum but for obvious reasons that was not possible. It was a great experience, a true eye-opener how different it is to work with the team.

7

Conclusion

PeakLife has great potential to be much more than it is right now. Currently the site is in its bare bones and had we had more time, one of the first thing we would have liked to implement is a regional scored based on regional data from within individual countries. This would have required several extra APIs per country, as the ones that we used simply had data per country.

Another thing that could have been implemented with more time available would be a broader breakdown of the various factors affecting the score. The user then could actually understand better what the score means and where exactly the particular country scores high or low. Building onto this would then be a filtering system where the user could select or deselect factors he/she/they do not value as important. Also, a comparison feature could be implemented in order to let users measure two or more countries of their choice against each other. This can also be combined with the filtering feature to compare only selected parameters. Finally, a ranking system could be implemented that shows which countries perform best on selected categories.

The layout of the site should could have been redesigned using JavaScript, had we been able to use that language. This would have allowed for a more natural look on all devices, as bootstrap was not enough to allow for parts such as the footer to stay at the bottom of the screen when scrolling through the website on a computer. This footer, which contained a link to the homepage as well as a copyright, was also troublesome on mobile devices as it would also stay in place when scrolling through the site.

Another part which could have been done better was designing a much more complex algorithm which takes into account current affairs or weighs more factors against each other. This would require more data processing power but the result would perhaps be more accurate.

After 9 weeks of working on the project, the result is something that we are proud of. We all learned something that is really helpful in the future and python-based namely Django, which is what we initially wanted to get out of this project.

Appendices

A

Supporting Material

A.1. Screenshots from the PeakLife App

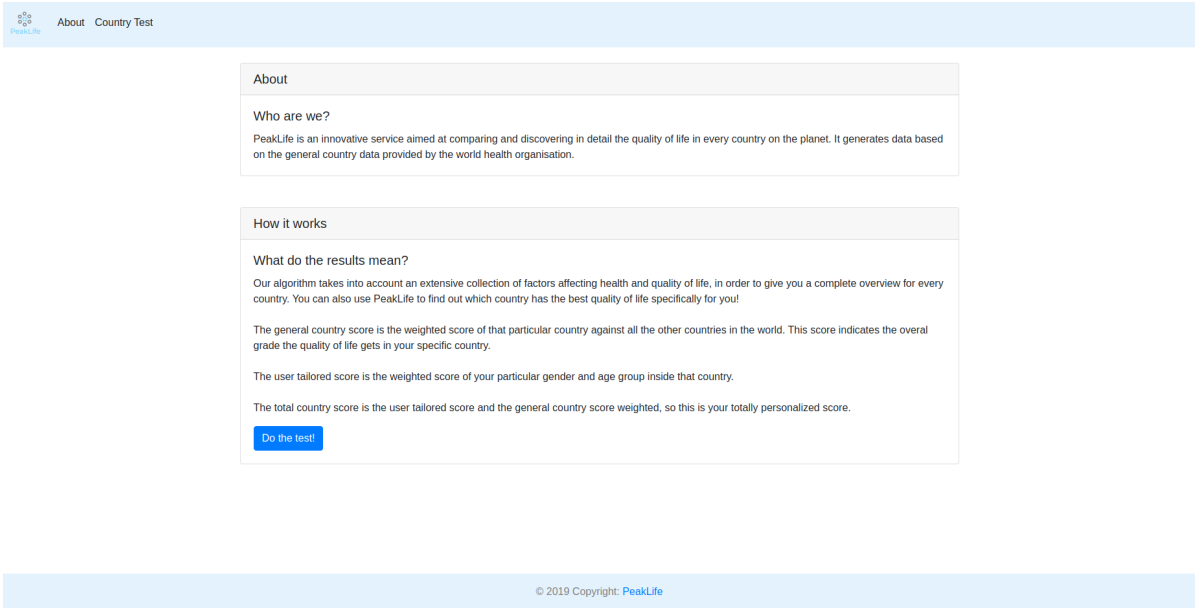

Figure A.1: Main menu page

Figure A.2: Country selection and user insertion page

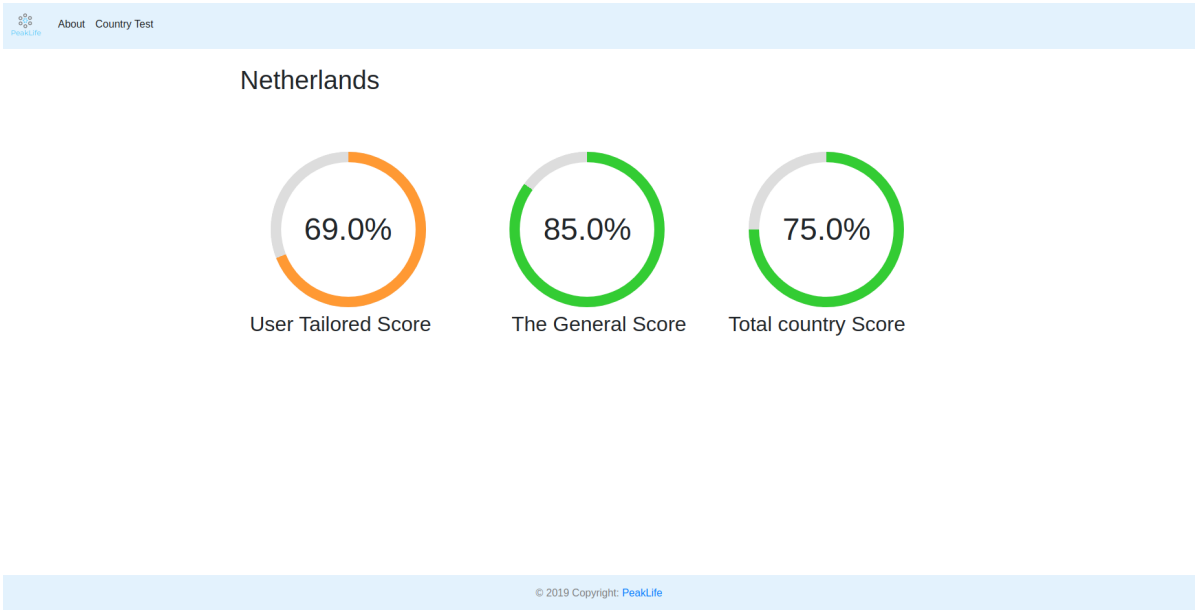

Figure A.3: Data visualization page

Bibliography

[1] Simon Willison Adrian Holovaty. Django: web framework for Python. 2005–. URL https://www.djangoproject.com/.

[2] Adobe Systems. Adobe Spark: video development platform. 2016–. URL https://spark.adobe.com/.