SEM: Project Report

Tripper: Always on time!

Group 23

TU Delft

SEM: Project Report

Tripper: Always on time!

by

Jasper van Brakel, 5323215,
Maarten Koopmans, 5186374,
Dielof van Loon, 5346894,
Roland van Dam, 5029511,
Jacob Reaves, 5348609

TU Delft

Contents

1 Introduction

GitLab link:
`https://gitlab.ewi.tudelft.nl/ti3115tu/2022-2023/Group-23/`

Nowadays, a vast amount of people have replaced their pen on paper agenda with an application on their smartphone. More specifically, the online agenda app created by Google, which is the most widely used online agenda, has over one billion downloads from the Play Store alone [2] An online agenda provides many functionalities over a traditional one. Some major ones are: Being able to store more information per agenda point; Automatic integration of multiple agendas; Sharing an agenda with multiple people; Adding reminders and alarms. Unfortunately, the Google agenda app does not make full use of its potential. A significant portion of appointments require the user to travel. However, Google Agenda does not provide any useful features to help relieve users of numerous repetitive tasks that come with planning to travel to - and from these locations. It takes time the find the location, think about and choose your travel method and then plan your itinerary in Maps or another navigation app. Google Agenda also doesn't automatically remind people of the travel time or when they need to depart, causing people to run late for their appointment.

This is where Tripper provides a solution. The app calculates the time, according to the preferred travel method, at which you need to leave your house/current location to get to the appointment location and arrive in time. It blocks the calculated time in the users Google Calendar and can also send departure reminders and add a Google Maps link to the itinerary. This app will be extra useful for TU Delft students since it will interpret faculty building codes and convert them to appointment locations.

By making use of this software, a repetitive task is taken out of hands, saving a lot of time in the long run. And it will help users to never run late again for important appointments.

2

Software Development Process

Throughout this project, the principles of the agile manifesto are kept in mind. Frequently delivering working software and increasing effectiveness throughout the project is of high importance, to be able to deliver a working end product in a relatively short period of time. That is why a scrum workflow is adopted by the team. The workflow was kept up with 2 meetings during the week. A short-term plan was made in these meetings to know what had to get done before the next meeting.

Our issue system works as follows; all issues receive a label on creation in line with the MoSCoW method. A weight and milestone number are also added. During development, a Kanban board was used to keep track of the completion of the issue, as displayed in Figure 2.1.

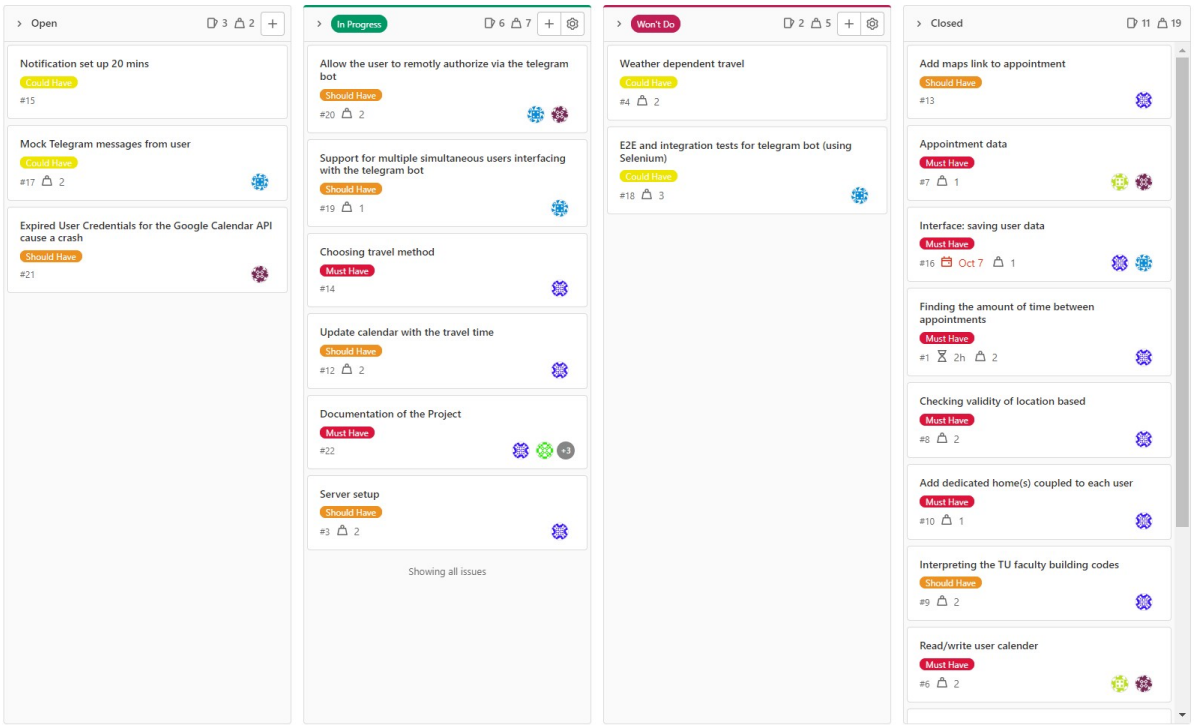

Figure 2.1: Our Kanban board during development.

The issues are assigned according to previous knowledge about the issue, weight, and priority.

The first working version of the software was realized at the end of sprint 2, while the requirement of a solid software setup for the whole project was also adhered to. Now part of the focus will shift to completing a draft report, by working on point 4 of A.3.1. Planning the supporting material, while still working on adding features to the software.

2.1. Reflection

2.1.1. Sprint 1 retrospective

During the work done in sprint 1.(A.1) Additional features that are needed in the future were already worked on to have a strong base for upcoming sprints. Most of the group effort was put into researching what code was most important to first write. And the actual code writing.

In the end, most of the planned tasks took longer than estimated so the time management was not on point. During upcoming sprints, this will be considered more carefully.

2.1.2. Sprint 2 retrospective

During this sprint, it already showed that the extra work done in sprint 1 had us save time during this sprint.(A.2) By creating an initial setup in sprint 1 for the software so that in the future all features could be added smoothly and with less modification to the initial code.

A basic working version of the software was realized during sprint 2. This was an important milestone that needed to be achieved. Now it is important to put more effort into the report to have a draft version by the end of sprint 3.

2.1.3. Sprint 3 retrospective

Up until the end of this sprint. One group member had been working hard on creating a working version of the telegram bot. This was an essential part of the software for the Tripper to work with specific user information, like the home location. The Telegram bot was nearly working and the report was getting completed.(A.3)

Now that a working version of the software exists and functions are being added, it is important to write tests for these functionalities. So in sprint 4, work will be put into writing tests, writing the report, and completing some of the remaining ”must have” and ”should have” issues.

2.1.4. Sprint 4 retrospective

In this sprint, all of the ”must have” issues were completed and most of the ”should have” issues were completed.(A.4) There was more focus put into writing the report. The first final version of the report was made.

Now that most of the issues were completed and the deadline of the project was coming up. The focus will be more on writing the report and implementing the feedback gotten from the TA. Also, adding all the documentation of the project, like a ReadMe. The last thing that will be done to the software, is to try and get it run on a server.

2.1.5. Sprint 5 retrospective

This last sprint was dedicated to getting the software to be run on a server.(A.5) Some issues were changed to ”won’t do” so that more work could be put into more important issues. The rest of the work was put into report writing.

3

Requirements Engineering

For this report, user stories were used for coming up with software requirements. This suited the development process better compared to use cases. Since the bigger priority was how the user is going to interact with the software, instead of how to software was going to work internally. Some problems were how a UI was going to be implemented, because of the work that already needed to be done. It couldn't be that complicated. So eventually the telegram bot was chosen because there are many people already working on that, so most of it should be easy to find.

The user stories are detailed enough so that a full set of software requirements is also described within these user stories. This was the goal from the beginning, and it was also the main challenge while writing the user stories. Namely making sure that as a whole, the user stories were detailed enough so that no important requirements were neglected or forgotten.

3.1. User stories

Title: Google API login	Priority: Must have	Weight: 3
As a new or existing user, I need a google API key to get access to google calendar. This makes it possible for the software to edit and add something to a calendar, using this API key.		
Acceptance criteria: The user retrieved an API key of their google calendar, and the Tripper software can make use of this key to get access to the calendar.		

Table 3.1: Milestone 1 issue Google API login

Title: Write user calendar	Priority: Must have	Weight: 2
For each appointment, I want to have travel time blocked before the appointment in my Google calendar. For this, it is necessary that Tripper can add and write appointments in my calendar		
Acceptance criteria: The Tripper adds an appointment in my Google calendar before every appointment that requires me to travel.		

Table 3.2: Milestone 1 issue Write user calendar

Title: Appointment data	Priority: Must have	Weight: 1
For each appointment, I want to receive the correct travel time in my Google calendar. For this, it is necessary that Tripper can read the appointments in my calendar so it can calculate travel time.		
Acceptance criteria: Tripper is able to gather data from appointments in my Google agenda and create a file with this information to be used for calculating travel time		

Table 3.3: Milestone 1 issue Appointment data

Title: Finding the amount of time between appointments	Priority: Must have	Weight: 2
I want to know the correct travel time between two subsequent appointments.		
Acceptance criteria: The Tripper can compare the time between appointments and estimate the travel time.		

Table 3.4: Milestone 2 issue Finding the amount of time between appointments

Title: Basic interface setup	Priority: Must have	Weight: 3
As a new Tripper user, I can provide Tripper with necessary information, by sharing this information during an automated conversation with a bot on Telegram.		
Acceptance criteria: I can share the maximum distance that I want to travel by foot and by bike and the minimum distance to travel by car and by train.		

Table 3.5: Milestone 2 Basic interface setup

Title: Add dedicated homes to each user	Priority: Must have	Weight: 1
I can provide the Telegram bot with a home address, which the Tripper can set as default departure address		
Acceptance criteria: The Tripper saves and stores the home address that I provide. The home location can be changed.		

Table 3.6: Milestone 2 Add dedicated homes to each user

Title: Interpreting the TU faculty building codes	Priority: Should have	Weight: 2
As a TU Delft student or employee, Tripper needs to be able to parse a TU Delft faculty building code from an appointment in my agenda, and return the corresponding Google maps location		
Acceptance criteria: Tripper can translate TU Delft faculty building codes to Google maps destinations		

Table 3.7: Milestone 2 Interpreting the TU faculty building codes

Title: Checking the validity of location based	Priority: Must have	Weight: 2
I only want to see travel times blocked in my calendar for appointments that include TU Delft faculty building codes or a valid Google maps location		
Acceptance criteria: Shows the travel time when the location is a valid google maps location. Shows the location when the location is a valid TU Delft building code/Faculty descriptor. The validation should fail when multiple options are presented		

Table 3.8: Milestone 2 Checking the validity of location

Title: Choosing the travel method	Priority: Must have	Weight: 2
Each of the travel times I can find in my Google calendar is based on the travel method that is best suited for the appointment that it belongs to.		
Acceptance criteria: The Tripper will automatically choose a travel method for each appointment, based on my travel preferences. I can manually change the travel method for each appointment.		

Table 3.9: Milestone 2 Choosing the travel method

Title: Estimate travel time using the Google maps API	Priority: Must have	Weight: 2
I need Tripper to make an accurate estimation of travel times for each appointment, by using Google maps..		
Acceptance criteria: Tripper uses either my home address or the location of a preceding appointment as departure location. Tripper determines the appointment location as destination. Tripper estimates all possible travel times using Google maps and chooses the method that best suits the preferences To each travel time appointment in my Google calendar, Tripper adds the corresponding estimated travel time.		

Table 3.10: Milestone 2 Estimate travel time using Google Maps

title: Interface: saving the user data	Priority: Must have	Weight: 1
After providing the Telegram bot with my information. I need the tripper to store the data so it can use it for all appointments in the future.		
Acceptance criteria: The user data entered in the Telegram bot gets stored in a file and can be accessed by the Tripper		

Table 3.11: Milestone 3 Saving the user data

Title: Interface: Update calendar with the travel time	Priority: Should have	Weight: 2
As a user, after I the time or location of an appointment in my agenda gets changed, I need the Tripper to automatically update the corresponding travel time.		
Acceptance criteria: The Tripper rereads and recalculates the travel time after the appointment is changed		

Table 3.12: Milestone 3 Interface: Update calendar with the travel time

Title: Add maps link to appointment	Priority: Should have	Weight: 1
A link is included in the travel time event in my Google calendar. This link will direct me to the correct itinerary on Google Maps		
Acceptance criteria: Even before the time of departure, a link to Google Maps will be available. In google maps, the itinerary is pre-selected and ready to be used for directions.		

Table 3.13: Milestone 2 Add maps link to appointment

Title: Notification set up	Priority: Could have	Weight: 1
If I wish to be reminded of my departure time in advance. I am able to have to Tripper set a notification/ alarm in the Google calendar travel time event		
Acceptance criteria: If switched on, a notification will be set that triggers a specified amount of time before the time of departure		

Table 3.14: Milestone 3 Notification setup

Title: Server setup	Priority: Should have	Weight: 2
I am able to have the Tripper run continuously to always have the Google calendar be up to date.		
Acceptance criteria: The Tripper runs 24/7 remotely from a server		

Table 3.15: Milestone 4 Server setup

Title: Mock telegram messages from user	Priority: Could have	Weight: 2
As a developer, I can enter messages in the Tripper in python code style, which act as real Telegram messages, to allow reliability testing		
Acceptance criteria: Telegram bot registers fake texts as user texts. Fake texts work like user texts regarding commands and answers.		

Table 3.16: Milestone 4 Mock telegram messages

Title: Weather dependent travel	Priority: Won't do	Weight: 2
I can have Tripper select a travel method based on the weather.		
Acceptance criteria: Tripper makes the travel methods walking and cycling less favourable during its travel method selection if the weather forecast predicts rain.		

Table 3.17: Milestone 5 Weather dependent travel choice

3.2. The future of the product

When more time would be available to further develop Tripper, we would first go on a thorough bug hunt to make the software more stable. We'd also like to make it possible to serve multiple client calendars

at the same time, which is not that far out of reach if we had a little more time. There are some features that would be a good addition to the working software, but these didn't make it in the priority queue. Such as weather depended filters, public transport information, logical transport methods (that you need to drive back while you came with public transport) and some features that will be thought off in a possible next quarter.

4

Software Architecture

4.1. General architecture

As with all software, we start in main. All of the functions and classes described further in this chapter come together in the main script. The main script starts with initializing the ngrok tunnel. This tunnel is used to make our program reachable by the Google webhooks (without opening any ports). After that we come to user selection. If a new user is desired, we start up the Telegram bot and authenticate a new user. The Telegram bot is started as a subprocess and runs the whole time in the background. This enables the user to always change their preferences during operation.

After that, or if an already existing user is selected, a request is send to Google, requesting the webhooks, which are pointed to the ngrok URL. When all is ready, the whole Tripper calendar is cleared, to start with a clean slate. After that all events from the Tripper calendar and all calendars tagged with [Tripper] in the des cription are arranged. Once all the events have been arranged in the proper order, we continue to the `update_tripper` function.

The `update_tripper` function checks all the Tripper events already present if all properties are still up to date (start, end, location, duration). If not, it will call `update_tripper_event`, which will update all outdated properties. In some cases it will decide to completely delete the event and add a new one.

During it's traversal of the events, it also checks if there are any new events without a Tripper event and if it is needed to add one. Adding a new event is done with `update_tripper_event`. This function calls `mode_selector` with the destination event and the previous event to determine which method of transport is most suited for the trip. Once this has been decided, an event is made with the title containing the minutes of time travelled and the method of transport. The description contains the start and end address of the trip and a Google Maps link, to immediately start navigating to the next event.

Once the startup run is complete, we wait and listen for any webhooks send from Google. This will trigger an update run, which causes an almost instant update of the changed Tripper event!

4.2. Google Calendar API interface

The Google Calendar API for 'Events' or Appointments is wrapped in the `CalendarEvent` class. This class has usages and problems to solve:

- Provide a simple and clear interface to work with the existing Google Calendar 'Events' from multiple 'Calendars'. With the main task being to sort them based on starting time and check the location.
- An easy way to create new Google Calendar 'Events' in a structured and simplified way. This was needed because the Google Calendar Events API contains a lot of non-functioning, depreciated and read-only fields. Furthermore some data fields have non-intuitive names. The 'Title' of a Google Calendar Event is referred to as a 'summary'.

In order to achieve these things, the `CalendarEvent` class was made. The first thing is that there are three ways to instantiate an `CalendarEvent` Object:

__init__ This is the standard Python `init`-function. It takes 4 parameters:

- a Google Calendar API service
- the `calendarID` of the calendar in which the event is located. (Or will be located).
- An dict in which the preexisting calendar data is stored.
- an Boolean variable to indicate if the event is a preexisting event of not. It defaults to `false`.

new This function is an easier way to make a `CalendarEvent` for an event that does not exist yet. It takes the following parameters:

- a Google Calendar API service
- the `calendarID` of the calendar in which the event will be located.

get_event This constructing function sets retrieves the Event data from the calendar directly. It takes the following parameters:

- a Google Calendar API service
- the `calendarID` of the calendar in which the event is located.
- the `eventID` of the Event, that you want the `CalendarEvent` representation of.

During construction the relevant data of the Google Calendar Event is stored in attributes of the `CalendarEvent`. These are exposed as properties with only a getter defined.

This is done in order to ensure the chainable setters are used. These functions (named like `set_FIELD`) do three things. First it notes stores a flag of which field was changed, applies the change and returns the object (`self`). The returning is done to create a fluent builder-like design pattern for these fields, which allows chaining of methods.

Some fields can also be locked from the Google Calendar API, the appropriate setter functions can deal with that and raise a custom error (`ReadOnlyError`).

When all the desired fields are set, the `execute` function should be called. This will construct a dict and create a new Google Calendar Event or update it if it already exists.'

All this has been done to eliminate error due to interfacing with the Google Calendar API, which requires you to check the following errors yourself (Or you get an error response back):

- Are the minimal fields given?
- Are the field names spelled correctly?
- Do the fields correspond to the intended data? (e.g. the title-like visual thing is actually called)
- Is this field allowed to be changed? (Not locked or Read-Only)

Checking these things by hand is difficult in such a large API, like the Google Calendar API, which also contain allot of deprecated fields for backwards-compatibility with preexisting Google Calendar events.

4.3. Google Maps API interface

The Google Maps API was quite simple to integrate, as we really only needed the duration of different travel methods. The Google Maps Distance Matrix API would satisfy all our needs. We made a wrapper function with some network checks and caching, as the calls use our Google Cloud trial balance of $300.

4.4. Users

For all user related business we use the class `User`. The class is quite simple but very modular. Our users are identified by their Telegram ID and are stored in the `telegram_id` attribute. When running the program, all users are initialized by using the `load_users` class method. This method reads all users from our database (a `users.json`) and stores all attributes in a user data dictionary. This user data dictionary is readable and writable through the data method, which takes a `user_property` name and a value, if writing is desired. Quite a simple system, but thus far future proof and effective.

4.5. Telegram bot

The interface used for Tripper is a Telegram bot, which consists of two python files, the `telegram_bot.py` file and the `telegram_response.py` file. The former provides the majority of the functionality as this is the part that listens to any texts the user sends. This is setup using an `Updater`, which fetches updates for the bot and queues them for handling, and a `Dispatcher` which dispatches the the updates to various different handlers.

The interface makes use of three different handlers: a `CommandHandler`, a `MessageHandler` and a `CallbackQueryHandler`. Depending on the type of update, the dispatcher runs a user defined function, which, at first, is always a command. From there the user can continue the conversation with the bot to, for example, change their home location. In order to keep the users separate the bot stores how far the user is in the conversation for each user in a dictionary. This dictionary gets passed to each function via the `CallbackContext` and is persistent as long as the bot is not terminated.

After a user reaches the final input field their reply will be stored in a telegram response object. This object handles the saving of the preferences to a JSON file and checks what the data is that needs to be saved and if it is in the right format. After is has confirmed that, the data is saved using methods from the `User` class for consistent saving and reading.

The reason the architecture was written this way is because it is quite readable and not difficult to understand what code is being run when and for what purpose to the point where it is possible to read the code along as one is going through the interface on telegram.

An overview of the architecture is shown in Figure 4.1.

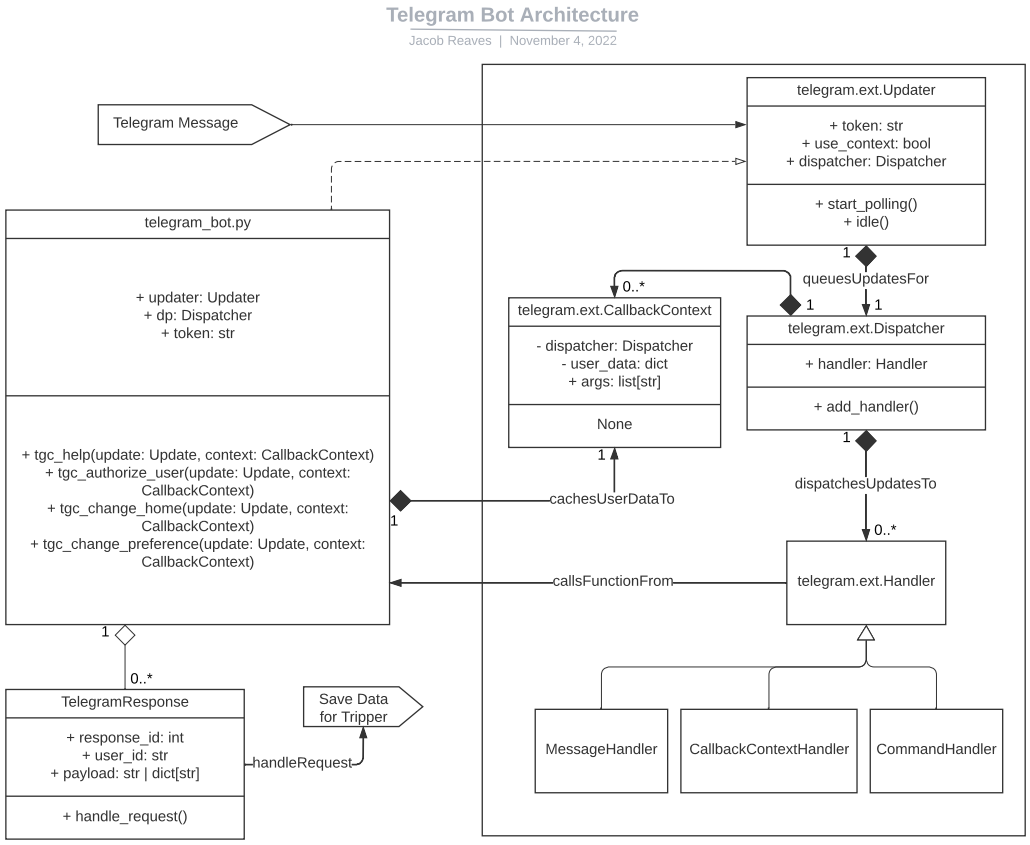

Figure 4.1: Telegram Bot Architecture

5

Software Testing

5.1. Tests

Lot's of testing has been done in the creation of Tripper, both automated and non-automated tests. Some of these non-automated test are still available by running scripts files, that aren't supposed to work on their own, for example `utils.py`, tries to get valid credentials for a test user when run.

The automated (unit)tests are all in the test folder, as is usual when using `pytest`. These are separated out by module, and they test various small functions. More details will be described in the following

5.1.1. Test Google calendar

A google account was made for the group to test our project. In this, calendar tests were made with random times and random locations to test the software.

In sprint one, the first test was done if a user story 'google API login' would work. This was done by getting an API key and a quick start guide from the Google Calendar API to get the first login with the Google Calendar API and the first information from an event. This first test was successful after a couple of tries.

At the end of sprint two, the first complete test of the working software was done. There was a basic version of the total software, with very few features. some calendar events were put in the test calendar and tested if it would get the travel time to the given events. The travel time was then placed in the calendar, right before the event. This test was the first concept of the full software that worked. Although it was a start and it only took the first event of one day. It did give the first full result.

During sprints, there were multiple small tests done to check if the code did what it had to do. These tests were also put into the automated unittests. For example, if the title of a `CalendarEvent` could be changed. Or if the start time of a `CalendarEvent` was able to be changed. These were all successful the first time of testing them. This made the group get the confidence to continue coding and not worry about if the code was wrong.

Later on these tests where expanded by checking more internal changed values. An other test was also made to check if the custom `ReadOnlyError` was raised correctly if the `CalendarEvent` was locked. There was also a test made to test if a `NotImplementedError` is raised when an unexpected change was added to the `updated_fields` field at creation of the payload for the API call to update or create a Google Calendar Event.

5.1.2. Test Google Maps

With the same google account, that was made for the calendar tests, tests were done for the software made for the maps API.

In sprint one, there was a similar test done for the maps API as the calendar API. A quick start code was used to get an understanding of the API. Exploring the code with this an understanding of the output was acquired from the software.

At the end of sprint two using the same test as our first-time software concept. The first version of the maps API was working in combination with the calendar API. Now the focus was on features that had to be added.

During sprints, many small tests were done to check small additions of code. This came down to testing the software with different inputs. Examples of these are: changing the travel method and getting the travel time from one place to another place. The maps API worked quite fast and there were not a lot of problems during this process.

At some point the possibility to read TU Delft building codes was added, this was also tested with a non-automated test at the time of creation.

5.1.3. Test telegram bot

The telegram bot was tested manually at first, but towards the end of the project unit tests covering around 75% of the code (of the telegram bot) were written.

Sprint one was very challenging. There are a lot of wrappers for Telegram, so there were a lot of choices. However, after researching a couple it was clear that, due to the variety, there were a lot of resources for various different wrappers, but none of them went in-depth. Having to rely solely on the documentation [1] made figuring out a way to automatically test difficult. A logging system was set up to view the incoming user messages on Telegram and how they got processed in order to see how the software could be improved.

The lack of automated testing made the overall testing process slower than usual, however, it was still faster than setting up automated tests. That was the case, because automation would require the mocking of user messages, which entails that numerous responses from Telegram would have to be mocked, as well as a refactor of the code since the bot would attempt to reply to a non-existent user, which would not work. However, this in the end did not prove to be a big issue overall. By the end of sprint two most of the features had been laid out or implemented and the flow of the program is quite easy, following the flow of the conversation with the user.

A recent deep dive in the source code of the Telegram wrapper revealed an easy way to clean up the code and implement functionality for multiple customers at the same time. It also inspired a way to mock user messages without too much refactoring. Later on automatic tests were written despite previous difficulties. These tests tested the `conversation_status` after each of the functions. This variable, unique to each user, is crucial since it signals the bot what code to run in a given function.

5.2. Coverage

The 38 automated tests that are in the software cover 52% of the software. This had a big increase because we got it to test the telegram bot. The coverage still wasn't that high, because a lot of IO functions were hard to get tested. It would have needed to be done with mocking. The time for mocking was simply not there. The coverage over time was really stable without the telegram bot, but with it you see it raise by 20% in the last week. In Figure 5.1 below you see the coverage over time.

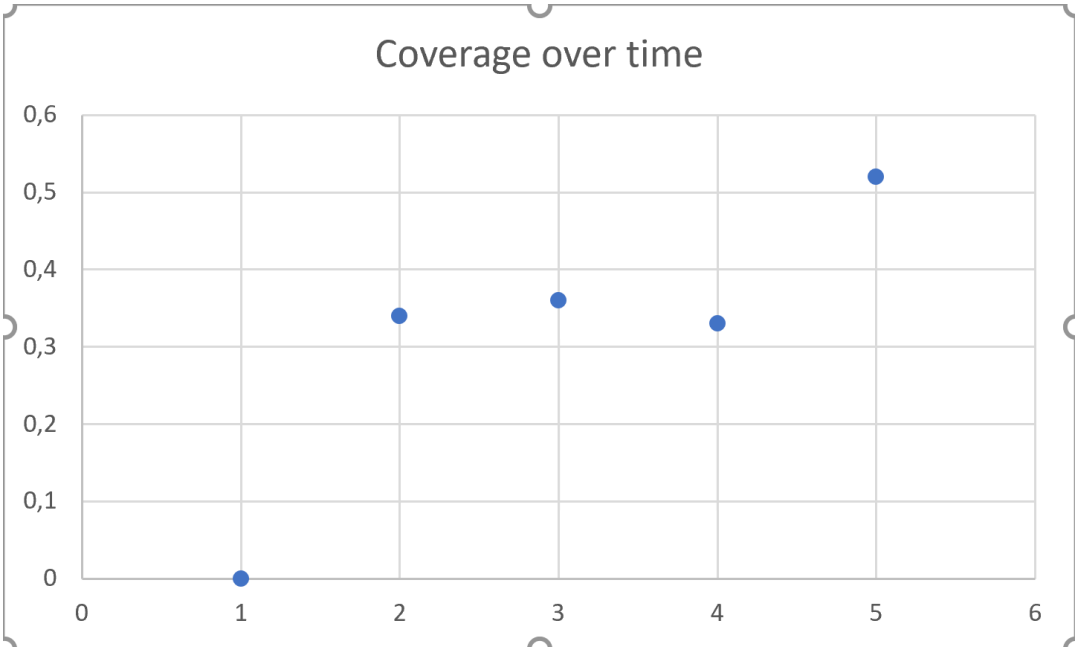

Figure 5.1: coverage over time.

6

Reflection and Adaptation

6.1. Jasper van Brakel

Working on this project was a nice experience of which I learned a lot. This is because I had worked on some small software projects prior to this course, but on those I mostly worked alone. So the experience of working together with a team consisting of people with different backgrounds and programming experiences was positively interesting. Then designing a software architecture together in a way, that the work could easily be dived, went smoother than expected, but the planning of the integration could have been better. In total this course learned me how to work together on software projects in a structured and organized way. It also reminded me of how easy things can get, when everybody is communicating clearly with each other, by means of day to day communication and written comments and documentation.

In the future I want to learn how to write software which is better testable, since this course has also showed me how the reliability increases a lot, when software is tested properly.

6.2. Maarten Koopmans

This project made it clear that everyone is on a different level of programming or other skills. Because of that, you need to communicate well to get the best out of everyone and use their abilities as well as possible. Also that a great plan makes for easier execution. A logical step for me to take as a software engineer is to get better at programming and learn how to write in a group. And I hope I can take that step in my next project.

6.3. Dielof van Loon

I think the biggest success during this project was picking our concept. We picked a product that was doable with clear functionality and not too ambitious. Because all the services we needed to use where from Google, I knew upfront that enough documentation was available, which also made the implementation go quite smoothly. While the workload wasn't always equally distributed, I still am quite happy with this team, as we have three developers with prior experience and two other developers who tried to help where they could. This way we where able to make, in my opinion, a nice and (almost) complete product!

This project taught me primarily the infrastructure used and the unwritten rules as a developer. It also showed me that I might want to reconsider becoming a Mechanical Engineer.

6.4. Roland van Dam

In my mind, our team has been collaborating really well during this project. At the beginning, it was clear that the degree of knowledge differed quite a bit between group members. I think that one key aspect that made our group work well together and what makes a software engineering team work well in general was that everybody made their expectations of himself and each other very clear. This made us aware of what we wanted to achieve and what we needed to do to make that happen. Furthermore,

this course made me understand how software developers can collaborate during code writing and code engineering in general, since that was still unknown to me.

In the future, I would like to take more initiative with the software architecture and code writing. Up until now, I have let my teammates take this initiative because I feel like this required certain prior knowledge and skills that I did not yet possess.

6.5. Jacob Reaves

I was fortunate enough to have decent prior coding experience which made helping to develop the program a lot easier. This biggest learning point for me was the coding style and the collaboration aspect. It was crucial that your code could be read and interpreted easily by the others to keep a steady workflow. Since I have thus far mostly worked on the telegram bot, which does not have tests yet, there was little point in me setting up automated testing. I did manage to write some tests for the Telegram bot testing the flow of the application. This was not super useful since the code at that point had been tested manually quite some times, but it was a valuable learning experience nonetheless. For example, I had no clue how to patch functions using mocking prior to writing the tests.

One aspect I think we could have implemented better, is the deadlines. We had a plan of what to do and in what order (namely MoSCoW), but we did not set strict deadlines for our issues, though I do think we spread the workload relatively evenly across the timeline despite the lack of hard deadlines. Additionally, even though it is difficult to criticize, the weights we assigned to issues often were not representative of how much effort they actually took. Most of the time, they took more...

7

Conclusion

The purpose of this project was to make a product, with programming. But also to learn how to work in a group while programming and getting to use programs that are commonly used during these projects.

Our project was a software program where it would automatically add travel time to your calendar, to get people a more organized life. Our project was split into three different parts, Google Calendar, Google maps, and the telegram bot. This made it better to split up the work and work more efficiently. The work was separated by multiple issues, to keep track of what was important. This made it so the end product at least had the must-have parts in it.

The software that was made during this project, has most of the features added to it, that we wanted. But there are some things that could be added to the software like weather-dependent adaptation. In the time that was available, it was not possible to get the complete software that was entirely complete. But that can always still be worked on.

Appendices

A

Supporting Material

A.1. Sprint 1

running cycle from 14-9-2022 to 28-9-2022

A.1.1. Planning

Milestone 1				
Order	Issues	Assignees	Due date	Completion date
1.	Google API login	everyone	30-9-2022	22-9-2022
2.	Read/write user calendar	Jasper van Brakel and Maarten Koopmans	30-9-2022	29-9-2022
3.	Appointment data	Jasper van Brakel and Maarten Koopmans	30-9-2022	-
4.	Telegram bot	Jacob Reaves	30-9-2022	-
5.	Google maps API	Dielof van Loon	30-9-2022	-

Table A.1: sprint 1 Planning

A.1.2. Review

Telegram bot is more work than expected and not a lot of information can be found for our setup. There are multiple wrappers that can be used and only small snippets of information are available. So it is hard to merge everything. User class all features needed are done. Everything needed more work than expected. Also, some were more experienced with python than others so it was the plan to change how much programming was done by less experienced programmers and let them focus on the non-programming part.

A.2. Sprint 2

running cycle from 28-9-2022 to 14-10-2022

A.2.1. Planning

Milestone 2				
Order	Issues	Assignees	Due date	Completion date
1.	Basic interface setup	Jacob Reaves	14-10-2022	22-9-2022
2.	Interpreting TU delft faculty codes	Dielof van Loon	14-10-2022	30-9-2022
3.	Add dedicated home to each user	Jacob Reaves	14-10-2022	7-10-2022
4.	Estimate travel time using Google Maps	Dielof van Loon	14-10-2022	30-9-2022
5.	Work on chapter 2 and 3 of the report	Roland van dam	14-10-2022	—

Table A.2: Sprint 2 planning

A.2.2. Review

A basic working version of the software was completed. It is now possible to scan all calendars and add travel times for the first appointment of the day that has a specified location, based on cycling as a travel method. The telegram bot is not being used yet.

There was a start made to writing the report because the assessment was made that it had to be kept up during the project.

A.3. Sprint 3

running cycle from 14-10-2022 to 21-10-2022

A.3.1. Planning

Milestone 3				
Order	Issues	Assignees	Due date	Completion date
1.	Saving user data	Jacob Reaves and Jasper van Brakel	21-10-2022	21-10-2022
2.	Checking validity of location	Dielof van Loon	21-10-2022	21-10-2022
3.	Add maps link to appointment	Dielof van loon	21-10-2022	28-10-2022
4.	Work on report chapters 1, 2, 3, 4	Roland van Dam and Maarten Koopmans	21-10-2022	—

Table A.3: Sprint 3 planning

A.3.2. Review

The tripper can now read user data that is stored in a file. Also, the tripper checks whether the appointment location is a valid location on Google Maps. If there are multiple, it returns the first one. Adding a Maps link to the appointment will be continued to be worked on in sprint 4. Also, work on the report is continuous until the deadline of the report.

Because the report was more work than thought beforehand. from now on two people were working on the report. Because the software engineering process was going so well, it was a good choice to put less focus on the software itself.

A.4. Sprint 4

running cycle from 21-10-2022 to 28-10-2022

A.4.1. Planning

Milestone 4				
Order	Issues	Assignees	Due date	Completion date
1.	Adding link to appointment	Dielof van Loon	28-10-2022	28-10-2022
2.	Support for multiple simultaneous users interfacing with the telegram bot	Jacob Reaves and Jasper van Brakel	4-11-2022	2-11-2022
3.	Choosing travel method	Dielof van Loon	28-10-2022	28-10-2022
4.	Working on first version of a report	Roland van Dam and Maarten Koopmans	28-10-2022	28-10-2022

Table A.4: Sprint 4 Planning

A.4.2. Review

In A calendar event, there is now a Maps link available. The travel method that is used is now automatically chosen using a max length of walking or biking. As expected setting up simultaneous users was hard to do, so it hasn't been finished. The first version of the report was finished to get feedback.

Everything went as planned. The report was handed in in time for the deadline. And there were no other surprises this sprint.

A.5. Sprint 5

running cycle from 28-10-2022 to 04-11-2022

A.5.1. Planning

Milestone 5				
Order	Issues	Assignees	Due date	Completion date
1.	Server setup	Dielof van Loon	4-11-2022	3-11-2022
2.	Support for multiple simultaneous users interfacing with the telegram bot	Jacob Reaves and Jasper van Brakel	4-11-2022	2-11-2022
3.	documentation of the product	Everyone	28-10-2022	4-11-2022
4.	Finalizing the report	Roland van Dam and Maarten Koopmans	28-10-2022	4-11-2022

Table A.5: Sprint 5 Planning

A.5.2. Review

Tripper is now able to be run on a server. Also, the telegram bot is now fully completed. The documentation has been updated. And the report has been finished with the given feedback from the teaching assistant.

Bibliography

[1] Docs.python-Telegram-Bot.org. python-telegram-bot v20.0a4, 21-10-2022. URL https://docs.python-telegram-bot.org/en/v20.0a4/.

[2] Google. Google calendar app, 16-4-2006. URL https://play.google.com/store/apps/details?id=com.google.android.calendar&gl=US.