

TruCol Company Roadmap

July 29, 2021

1 Introduction

Welcome, this document presents the roadmap for the TruCol company as well as the estimated costs of the accompanying human labour and other posts. The objective of this document is to provide some basic insight into the order of magnitude of the costs of developing a sustainable variant of the TruCol company such that it is able to generate returns for its potential investors.

Section 2 specifies the assumptions, section 3 describes how the costs are estimated, section 4 presents the projected basic planning and briefly details what each activity in the Gantt chart represents. Section 6 provides some nuance on the accuracy of the estimates, and section 7 summarises the body of information in this document.

2 Assumptions

2.1 Decentralisation Developer Wages

The hourly wage of the developers working on decentralised technology is based on a mixture of ± 3 junior developers working at €100.000,- per year, and 2 senior developers working at €200.000,- per year. This yields an average developer cost of

$$\frac{3 \cdot 50 + 2 \cdot 100}{5} = \frac{350}{5} = \text{€}70, - \quad (1)$$

The datapoints used to come to this estimate are the promoted starting wages for Junior Developers/Engineers at Optiver in Amsterdam, Think-cell in Berlin, and a third Zurich company, which all ranged between 80 to 120k at the time of inspection (Around March 2021). No proper datapoint is used to estimate the salary of the senior developers. Previous experience in co-working with senior developers led to an estimate that their hourly contributions are at least twice as valuable as that of a junior developer. Another indicator for the doubling in wage between junior and senior dev may be the hear-say high demand in solidity/decentralisation developers.

The 70,- hourly wage is interpreted as €75, – per hour to be on the conservative side of estimates.

2.2 Website/Platform Wages

The website+API+GUI development is estimated at €40 per hour. This estimate is based on a reduced hourly wage of the junior decentralised technology developers (from €50, – to €40, –). Some of the development costs for these activities may be performed at a lower hourly cost price, this platform development work also contains UX design. And excellent UX design is quite costly, hence the average hourly wage for this estimate is kept at €40, –.

2.3 Business wages

The hourly wages for the business development side of our company is estimated at €35, – per hour. This estimate is based on a reduced hourly wage of the junior platform developers (from €40, – to €35, –).

2.4 Activity durations

The estimates for the durations of the activities for both decentralised technology development as well as ecosystem development are extrapolations of our experience in developing in these disciplines. The business development activities durations are based roughly on estimating what those activities entail and how long it would take to complete them.

3 Cost Model Description

The total costs are computed based on two factors.

- The cumulative amount of human labour hours that are planned to be executed, multiplied with their respective hourly wage costs as specified in section 2.

- A combination of bounty subsidisation and buffer costs of €100.000, – are estimated to generate wide-spread adoption of the TruCol protocol.

4 Results

Figure 1 contains the Gantt chart that is generated to plan the development of the TruCol company. One can observe that several of the development-activities can be performed in parallel, these are accordingly stacked vertically. Dependencies of outputs of activities imply a "stairway" pattern in the Gantt chart.

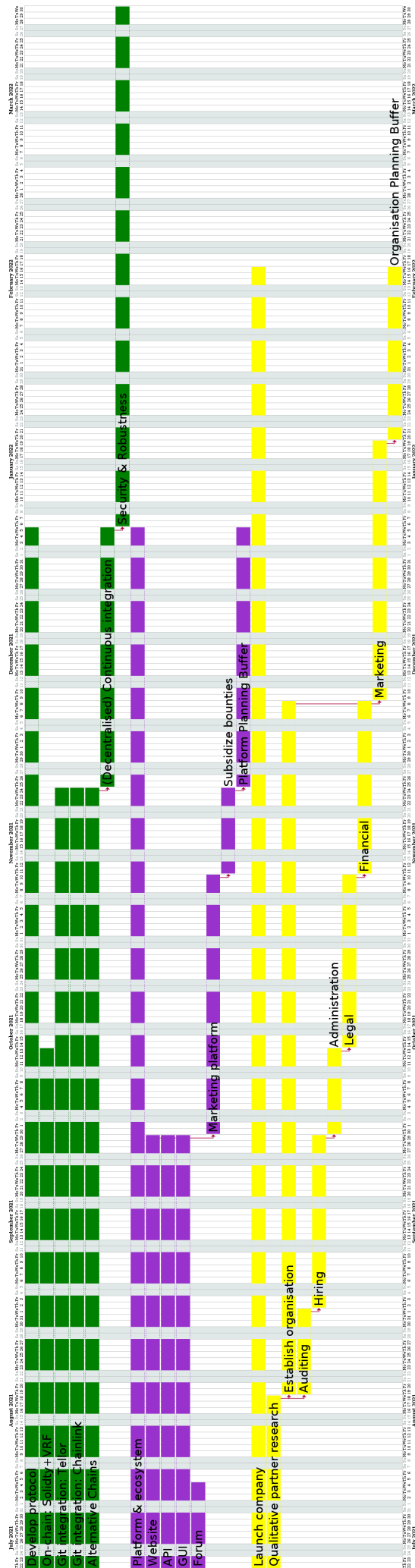


Figure 1: Gantt chart that is generated to plan the development of the TruCol company. (Source code in appendices, and on [github.com/trucol](https://github.com/trucol/trucol)/Roadmap).

The description of the activities can be given as:

4.1 Decentralised Technology Development

- **Develop protocol** - The programming work and documentation work that is required to render the TruCol protocol to a mature and robust state.
- **On-chain: Solidity+VRF** - Finalisation of the Solidity to Solidity implementation of the TruCol protocol implementation that leverages Chainlinks Verifiable Random function (VRF).
- **Git integration: Teller** - Providing a lower-cost option to the users whilst allowing the user to apply the TruCol protocol in practically any programming language using Teller oracles that query Git repository content and (build) status.
- **Git integration: Chainlink** - Same as the Teller option, except using Chainlinks oracles.
- **Alternative Chains** - Implementing the TruCol protocol in alternative chains to facilitate easy use for the users whilst possibly lowering costs and/or modulating the desired levels of decentralisation
- **(Decentralised) Continuous Integration** - Realizing a mature implementation in which the oracles can verify the build status (whether the tests in the smart contract actually passed or not) in a robust fashion. Ideally implementing support for decentralised CIs.
- **Security & Robustness** - A security audit of the TruCol protocol implementations.

4.2 Platform Development

- **Platform & Ecosystem** - Development of the online platform that provides a convenient place for users to use-, discuss- and learn about the TruCol protocol and its various implementations.
- **Website** - Completion of the company website.
- **API** - Application programming interface that allows users to submit contracts using the command line interface (CLI).
- **GUI** - Graphical user interface, that makes it easy and intuitive for new users to start using the TruCol protocol for their applications.
- **Forum** - Environment a-la stack-overflow that cultivates a knowledge base around the use of the TruCol protocol.
- **Marketing platform** - Development of the approach to realise wide-spread adoption of the TruCol ecosystem.
- **Subsidize bounties** - Subsidisation of bounties to attract new users to the platform.
- **Platform Planning Buffer** - A buffer accounting for unknown unknowns/unexpected delays.

4.3 Business Development

- **Launch company** - The administrative and non-technical aspects of growing the TruCol company.
- **Qualitative partner research** - An analysis to identify relevant partners in the growth of our company.
- **Establish Organisation** - organisational aspects of growing the company, with "Auditing, Hiring, Administration, Legal & Financial tasks as its respective subset".
- **Marketing** - Development of the approach to realise wide-spread adoption of the TruCol protocol whilst realising a steady stream of new customers.
- **Organisation Planning Buffer** - A buffer accounting for unknown unknowns/unexpected delays.

5 Cost estimates

After multiplying the amount of labour hours with their respective hourly labour costs, a total estimated labour costs of €470.400, – is generated. This is composed of:

- Decentralised Tech Development: €252.000, –
- Platform Development: €112.000, –
- Business Development €106.400, –

An additional €100.000, – are included for bounty subsidisation and as a buffer, yielding a total expected cost to create a healthy company of roughly $€470.000 + €100.000 = €570.000, –$ which is rounded to approximately .6 Meuro.

6 Discussion

The hourly wage estimates for the senior decentralised technology development, platform development and business estimates could be refined by using databases of competitive salaries for the respective tasks/work. Furthermore, a refinement of the duration estimates is proposed at the moment the tasks are started, as we expect more relevant information will be available at those times. In addition, external resources should be addressed to check whether any (critical) components are omitted in this Gantt planning.

7 Conclusion

This document presents the planning to develop the healthy TruCol company within the timeframe of roughly 9 months, and documents the assumptions and methods used to generate this planning. The main tasks in this planning are composed of decentralised technology development, TruCol platform development and business development. A total labour cost of roughly €470k is estimated, and another €100k is estimated for bounty subsidisation and as a buffer.

At the end of this planning, the TruCol company is expected to sustainably operate, generating ROI. Our aim is to gradually increase the cultivation of the diversification potential of the TruCol protocol at this point, whilst starting to develop a strategy to develop our own in-house automation/AI-engine based on the dataset that we continuously grow.

References

A Appendix __main__.py

```
1 import os
2 from .Main import Main
3 from .Compile_gantt_locally import compile_gantt_locally
4
5 print(f"Hi, I'll be running the main code, and I'll let you know when
  ↳ I'm done.")
6 project_nr = 1
7
8
9 main = Main()
10
11 # create gantt chart
12 main.create_gantt()
13
14 # Export Gantt to Latex
15 main.export_gantt_to_latex()
16
17 # compile the gantt chart locally
18 compile_gantt_locally(
19     main.relative_src_filepath, main.plantuml_java_filename, main.
  ↳ src_to_gantt_path
20 )
```

```
21
22 # export the code to latex
23 main.export_code_to_latex(project_nr)
24
25 # compile the latex report
26 main.compile_latex_report(project_nr)
27
28 print(f"Done.")
```

B Appendix Main.py

```
1 # Example code that creates plots directly in report
2 # Code is an implementation of a genetic algorithm
3 import random
4 from matplotlib import pyplot as plt
5 from matplotlib import lines
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import shutil
9
10 from .Compile_latex import Compile_latex
11 from .Gantt import Gantt
12 from .Plot_to_tex import Plot_to_tex as plt_tex
13 from .Export_code_to_latex import export_code_to_latex
14
15 # define global variables for genetic algorithm example
16 string_length = 100
17 mutation_chance = 1.0 / string_length
18 max_iterations = 1500
19
20
21 class Main:
22     def __init__(self):
23         self.project_nr = 1
24         self.plantuml_java_filename = "plantuml.jar"
25         self.src_to_gantt_image_path="Diagrams/created.png"
26         self.src_to_gantt_path = "Diagrams/created.uml"
27         # self.src_to_gantt_path="Diagrams/example_gantt.uml"
28         self.relative_src_filepath = f"code/project{self.project_nr}/
29             ↪ src/"
30         self.relative_plantuml_java_filepath = (
31             f"code/project{self.project_nr}/src/{self.
32             ↪ plantuml_java_filename}"
33         )
34
35     def create_gantt(self):
36         gantt = Gantt(f"{self.relative_src_filepath}/Diagrams/created
37             ↪ .uml")
38
39     def export_gantt_to_latex(self):
40         try:
41             #shutil.move(src_to_gantt_image_path, f'latex/project{self.
42             ↪ project_nr}/Images/')
43             print(f'{self.relative_src_filepath}{self.
44             ↪ src_to_gantt_image_path}')
45             print(f'to:latex/project{self.project_nr}/Images/')
46             shutil.copy(f'{self.relative_src_filepath}{self.
47             ↪ src_to_gantt_image_path}', f'latex/project{self.
48             ↪ project_nr}/Images/')
49             #shutil.move(f'{self.relative_src_filepath}{self.
50             ↪ src_to_gantt_image_path}', f'latex/project{self.
51             ↪ project_nr}/Images/')
52             #shutil.copy(f'{self.relative_src_filepath}{self.
53             ↪ src_to_gantt_image_path}', f'latex/project{self.
54             ↪ project_nr}/Images/')
55         except:
56             print("Error while moving file ", self.
57             ↪ src_to_gantt_image_path)
58             # exit()
59
60     def export_code_to_latex(self, project_nr):
```

```

49         export_code_to_latex("main.tex", project_nr)
50
51     def compile_latex_report(self, project_nr):
52         """compiles latex code to pdf"""
53         compile_latex = Compile_latex(project_nr, "main.tex")
54
55     def addTwo(self, x):
56         """adds two to the incoming integer and returns the result of
57             ↪ the computation."""
58         return x + 2
59
60 if __name__ == "__main__":
61     # initialize main class
62     main = Main()

```

C Appendix Activity.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 import numpy as np
4 import copy
5
6 from .Plot_to_tex import Plot_to_tex as plt_tex
7
8
9 class Activity:
10     def __init__(
11         self,
12         description,
13         duration,
14         new_tag,
15         colour=None,
16         parent=None,
17         starts_at_child_nr_start=None,
18         starts_at_child_nr_end=None,
19         font_size=None,
20         hourly_wage=None,
21         hours_per_day=None
22     ):
23         self.parent = parent
24         self.description = description
25         self.duration = duration
26         self.children = []
27         self.font_size=None
28         self.starts_at_child_nr_start=starts_at_child_nr_start
29         self.starts_at_child_nr_end=starts_at_child_nr_end
30         self.hourly_wage = hourly_wage
31         if hours_per_day is None:
32             self.hours_per_day=8
33         else:
34             self.hours_per_day=hours_per_day
35         if self.parent is None:
36             if colour is None:
37                 raise Exception("Parent activity needs a colour")
38             self.colour = colour
39             self.tag = []
40         else:
41             if (not self.parent.hourly_wage is None) and (hourly_wage
42                 ↪ is None):
43                 self.hourly_wage=self.parent.hourly_wage
44             self.colour = parent.colour
45             self.tag = copy.deepcopy(self.parent.tag)
46
47         # create tag
48         self.tag.append(new_tag)
49
50     def add_children(self, children):
51         self.children = children
52
53     def get_tag(self):
54         return "_".join(list(map(lambda x: str(x), self.tag)))
55
56     def addTwo(self, x):
57         """adds two to the incoming integer and returns the result of
58             ↪ the computation."""
59         return x + 2
```

D Appendix Compile_gantt_locally.py

```
1 # pip install plantuml
2
3 # to compile locally:
4 # export PLANTUML_LIMIT_SIZE=8192
5 # java -jar plantuml.jar -verbose sequenceDiagram.txt
6
7 # To compile with server:
8 from plantuml import PlantUML
9 import os
10 import subprocess
11 from os.path import abspath
12 from shutil import copyfile
13 import requests
14
15
16 def compile_gantt_locally(
17     relative_src_filepath, plant_uml_java_filename, src_to_gantt_path
18 ):
19     relative_plant_uml_java_filepath = (
20         f"{relative_src_filepath}{plant_uml_java_filename}"
21     )
22     print(f"relative_plant_uml_java_filepath={
23         ↪ relative_plant_uml_java_filepath}")
24     if got_java_file(relative_plant_uml_java_filepath):
25         print("FOUND")
26         os.environ["PLANTUML_LIMIT_SIZE"] = "8192"
27         run_bash_command(
28             f"java -jar {relative_plant_uml_java_filepath} -verbose {
29                 ↪ relative_src_filepath}{src_to_gantt_path}"
30         )
31         # export_gantt_to_latex(filename)
32     pass
33
34 def check_if_java_file_exists(relative_filepath):
35     if os.path.isfile(relative_filepath):
36         return True
37     else:
38         return False
39
40 def got_java_file(relative_filepath):
41     # Check if the jar file exists, curl it if not.
42     if not check_if_java_file_exists(relative_filepath):
43         # The java file is not found, curl it
44         request_file("https://sourceforge.net/projects/plantuml/files
45             ↪ /plantuml.jar/download", relative_filepath)
46     # Check if the jar file exists after curling it. Raise Exception
47     ↪ if it is not found after curling.
48     if not check_if_java_file_exists(relative_filepath):
49         raise Exception(f"File:{relative_filepath} is not accessible"
50             ↪ )
51     print(f'Got the Java file')
52     return True
53
54 def request_file(url, output_filepath):
55     import requests
56     # Request the file in the url
57     response = requests.get(url)
58     with open(output_filepath, "wb") as f:
59         f.write(response.content)
```

```
56
57 def run_bash_command(bashCommand):
58     subprocess.Popen(bashCommand, shell=True)
59
60
61 def export_gantt_to_latex(filename):
62     if filename.endswith(".png"):
63         copyfile(
64             abspath(f"./Diagrams/{filename}"),
65             abspath(f"../../latex/researchplan/Images/{filename}"),
66         )
```

E Appendix Compile_latex.py

```
1 # runs a jupyter notebook and converts it to pdf
2
3 import os
4 import shutil
5 import nbformat
6 from nbconvert.preprocessors import ExecutePreprocessor
7
8
9 class Compile_latex:
10     def __init__(self, project_nr, latex_filename):
11         self.script_dir = self.get_script_dir()
12         relative_dir = f"latex/project{project_nr}/"
13         self.compile_latex(relative_dir, latex_filename)
14         self.clean_up_after_compilation(latex_filename)
15         self.move_pdf_into_latex_dir(relative_dir, latex_filename)
16
17     # runs jupyter notebook
18     def compile_latex(self, relative_dir, latex_filename):
19         os.system(f"pdflatex {relative_dir}{latex_filename}")
20
21     def clean_up_after_compilation(self, latex_filename):
22         latex_filename_without_extention = latex_filename[:-4]
23         print(f"latex_filename_without_extention={
24             ↪ latex_filename_without_extention}")
25         self.delete_file_if_exists(f"{
26             ↪ latex_filename_without_extention}.aux")
27         self.delete_file_if_exists(f"{
28             ↪ latex_filename_without_extention}.log")
29         self.delete_file_if_exists(f"texput.log")
30
31     def move_pdf_into_latex_dir(self, relative_dir, latex_filename):
32         pdf_filename = f"{latex_filename[:-4]}.pdf"
33         destination = f"{self.get_script_dir()}/../../../{
34             ↪ relative_dir}{pdf_filename}"
35
36         try:
37             shutil.move(pdf_filename, destination)
38         except:
39             print("Error while moving file ", pdf_filename)
40
41     def delete_file_if_exists(self, filename):
42         try:
43             os.remove(filename)
44         except:
45             print(
46                 f"Error while deleting file: {filename} but that is
47                 ↪ not too bad because the intention is for it to
48                 ↪ not be there."
49             )
50
51     def get_script_dir(self):
52         """returns the directory of this script regardless of from
53             ↪ which level the code is executed"""
54         return os.path.dirname(__file__)
55
56 if __name__ == "__main__":
57     main = Compile_latex()
```

F Appendix Create_python_gantt.py

```
1 import numpy as np
2
3 from .Activity import Activity
4
5
6 def create_python_gantt():
7     parents = []
8     # parent one
9     parent_one = Activity("description", duration=5, new_tag=0,
10         ↪ colour="Green")
11     # children
12     child_one = Activity(
13         ↪ "child one description", duration=2, new_tag=0, parent=
14         ↪ parent_one
15     )
16     child_two = Activity(
17         ↪ "child two description", duration=2, new_tag=1, parent=
18         ↪ parent_one
19     )
20     # merge family
21     parent_one.add_children([child_one, child_two])
22     #parents.append(parent_one)
23
24     parent_two = Activity("description two", duration=5, new_tag=1,
25         ↪ colour="DarkOrchid")
26     # children
27     child_one = Activity(
28         ↪ "child one description two", duration=2, new_tag=0, parent=
29         ↪ parent_two
30     )
31     child_two = Activity(
32         ↪ "child two description two", duration=2, new_tag=1, parent=
33         ↪ parent_two
34     )
35     # merge family
36     parent_two.add_children([child_one, child_two])
37     #parents.append(parent_two)
38
39     ## parent
40     # parent one
41     protocol = Activity(description="Develop protocol", duration=120,
42         ↪ new_tag=0, colour="Green", hourly_wage=0)
43     # children
44     onchain = Activity(description="On-chain: Solidty+VRF", duration
45         ↪ =60, new_tag=0, parent=protocol, hourly_wage=75)
46     git_tellor = Activity(description="Git integration: Tellor",
47         ↪ duration=90, new_tag=1, parent=protocol,
48         ↪ starts_at_child_nr_start=0, hourly_wage=75)
49     git_chainlink = Activity(description="Git integration: Chainlink"
50         ↪ , duration=90, new_tag=2, parent=protocol,
51         ↪ starts_at_child_nr_start=0, hourly_wage=75)
52     alt_chains = Activity(description="Alternative Chains", duration
53         ↪ =90, new_tag=3, parent=protocol, starts_at_child_nr_start=0,
54         ↪ hourly_wage=75)
55
56     # grandchildren
57     ci = Activity(description="(Decentralised) Continuous integration
58         ↪ ", new_tag=0, duration=30, parent=git_chainlink)
59     #git_chainlink.add_children([ci])
```

```

45 security = Activity(description="Security & Robustness", duration
    ↳ =60, new_tag=1, parent=git_chainlink)
46 #ci.add_children([security])
47
48 # merge
49 protocol.add_children([onchain,git_tellor,git_chainlink,
    ↳ alt_chains, ci, security])
50 parents.append(protocol)
51
52
53 # parent_two
54 platform_eco = Activity(description="Platform & ecosystem",
    ↳ duration=120, new_tag=1, colour="DarkOrchid",
    ↳ starts_at_child_nr_start=0, hourly_wage=40)
55 # children
56 website = Activity(description="Website", duration=50, new_tag=0,
    ↳ parent=platform_eco)
57 marketing_platf = Activity(description="Marketing platform",
    ↳ duration=30, new_tag=1, parent=platform_eco)
58 bounties = Activity(description="Subsidize bounties", duration
    ↳ =10, new_tag=2, parent=platform_eco)
59 platform_buffer = Activity(description="Platform Planning Buffer"
    ↳ , duration=30, new_tag=3, parent=platform_eco)
60
61 # Grandchildren
62 api = Activity(description="API", duration=50, new_tag=0, parent=
    ↳ website)
63 gui = Activity(description="GUI", duration=50, new_tag=1, parent=
    ↳ website, starts_at_child_nr_start=0)
64 forum = Activity(description="Forum", duration=10, new_tag=2,
    ↳ parent=website, starts_at_child_nr_start=0)
65 website.add_children([api, gui, forum])
66
67 platform_eco.add_children([website,marketing_platf,bounties,
    ↳ platform_buffer])
68 parents.append(platform_eco)
69
70 # parent_three
71 company = Activity(description="Launch company", duration=150,
    ↳ new_tag=2, colour="Yellow", starts_at_child_nr_start=0,
    ↳ hourly_wage=35)
72 # children
73 partners = Activity(description="Qualitative partner research",
    ↳ duration=20, new_tag=0, parent=company)
74 organisation = Activity(description="Establish organisation",
    ↳ duration=80, new_tag=1, parent=company)
75 marketing_company = Activity(description="Marketing", duration
    ↳ =30, new_tag=2, parent=company)
76 company_buffer = Activity(description="Organisation Planning
    ↳ Buffer ", duration=20, new_tag=3, parent=company)
77
78 # Grandchildren
79 auditing = Activity(description="Auditing", duration=10, new_tag
    ↳ =0, parent=organisation)
80 hiring = Activity(description="Hiring", duration=20, new_tag=1,
    ↳ parent=organisation)
81 administration = Activity(description="Administration", duration
    ↳ =10, new_tag=2, parent=organisation)
82 legal = Activity(description="Legal", duration=20, new_tag=3,
    ↳ parent=organisation)
83 financial = Activity(description="Financial", duration=20,
    ↳ new_tag=4, parent=organisation)

```

```
84     organisation.add_children([auditing, hiring, administration,  
    ↪     legal, financial])  
85  
86  
87     company.add_children([partners, organisation, marketing_company,  
    ↪     company_buffer])  
88     parents.append(company)  
89  
90     return parents  
91  
92  
93 def addTwo(self, x):  
94     """adds two to the incoming integer and returns the result of the  
    ↪     computation."""  
95     return x + 2
```

G Appendix Export_code_to_latex.py

```
1 # runs a jupyter notebook and converts it to pdf
2 import os
3 import shutil
4 import nbformat
5 from nbconvert.preprocessors import ExecutePreprocessor
6
7
8 def export_code_to_latex(main_latex_filename, project_nr):
9     """This function exports the python files and compiled pdfs of
10         ↪ jupyter notebooks into the
11         latex of the same project number. First it scans which appendices
12         ↪ (without code, without
13         notebooks) are already manually included in the main latex code.
14         ↪ Next, all appendices
15         that contain the python code are either found or created in the
16         ↪ following order:
17         First, the __main__.py file is included, followed by the main.py
18         ↪ file, followed by all
19         python code files in alphabetic order. After this, all the pdfs
20         ↪ of the compiled notebooks
21         are added in alphabetic order of filename. This order of
22         ↪ appendices is overwritten in the
23         main tex file.
24
25     :param main_latex_filename: Name of the main latex document of
26         ↪ this project number
27     :param project_nr: The number indicating which project this code
28         ↪ pertains to.
29     """
30     script_dir = get_script_dir()
31     relative_dir = f"latex/project{project_nr}/"
32     appendix_dir = script_dir + "../..../" + relative_dir + "
33         ↪ Appendices/"
34     path_to_main_latex_file = (
35         f"{script_dir}/../..../{relative_dir}/{main_latex_filename}"
36     )
37     root_dir = script_dir[0 : script_dir.rfind(f"code/project{
38         ↪ project_nr}")]
39
40     # Get paths to files containing python code.
41     python_filepaths = get_filenames_in_dir("py", script_dir, ["
42         ↪ __init__.py"])
43     compiled_notebook_pdf_filepaths = get_compiled_notebook_paths(
44         ↪ script_dir)
45
46     # Check which files are already included in the latex appendices
47     ↪ .
48     python_files_already_included_in_appendices = (
49         get_code_files_already_included_in_appendices(
50             python_filepaths, appendix_dir, ".py", project_nr,
51             ↪ root_dir
52         )
53     )
54     notebook_pdf_files_already_included_in_appendices = (
55         get_code_files_already_included_in_appendices(
56             compiled_notebook_pdf_filepaths,
57             appendix_dir,
58             ".ipynb",
59             project_nr,
60             root_dir,
```



```

46     )
47 )
48
49 # Get which appendices are still missing.
50 missing_python_files_in_appendices =
51     ↪ get_code_files_not_yet_included_in_appendices(
52         python_filepaths, python_files_already_included_in_appendices
53         ↪ , ".py"
54     )
55 missing_notebook_files_in_appendices = (
56     get_code_files_not_yet_included_in_appendices(
57         compiled_notebook_pdf_filepaths,
58         notebook_pdf_files_already_included_in_appendices,
59         ↪ ".pdf",
60     )
61 )
62
63 # Create the missing appendices.
64 created_python_appendix_filenames = create_appendices_with_code(
65     appendix_dir, missing_python_files_in_appendices, ".py",
66     ↪ project_nr, root_dir
67 )
68
69 created_notebook_appendix_filenames = create_appendices_with_code
70 ↪ (
71     appendix_dir,
72     missing_notebook_files_in_appendices,
73     ↪ ".ipynb",
74     project_nr,
75     ↪ root_dir,
76 )
77
78 appendices = get_list_of_appendix_files(
79     appendix_dir, compiled_notebook_pdf_filepaths,
80     ↪ python_filepaths
81 )
82
83 main_tex_code, start_index, end_index, appendix_tex_code =
84 ↪ get_appendix_tex_code(
85     path_to_main_latex_file
86 )
87
88 # assumes non-included non-code appendices should not be included
89 ↪ :
90 # overwrite the existing appendix lists with the current appendix
91 ↪ list.
92
93 (
94     non_code_appendices,
95     main_non_code_appendix_inclusion_lines,
96 ) = get_order_of_non_code_appendices_in_main(appendices,
97     ↪ appendix_tex_code)
98
99 python_appendix_filenames = list(
100     map(
101         ↪ lambda x: x.appendix_filename,
102         filter_appendices_by_type(appendices, "python"),
103     )
104 )
105 sorted_created_python_appendices = sort_python_appendices(
106     filter_appendices_by_type(appendices, "python")
107 )
108 sorted_python_appendix_filenames = list(

```

```

99         map(lambda x: x.appendix_filename,
100             ↪ sorted_created_python_appendices)
101     )
102     notebook_appendix_filenames = list(
103         map(
104             lambda x: x.appendix_filename,
105             filter_appendices_by_type(appendices, "notebook"),
106         )
107     )
108     sorted_created_notebook_appendices =
109         ↪ sort_notebook_appendices_alphabetically(
110             filter_appendices_by_type(appendices, "notebook")
111         )
112     sorted_notebook_appendix_filenames = list(
113         map(lambda x: x.appendix_filename,
114             ↪ sorted_created_notebook_appendices)
115     )
116     appendix_latex_code = create_appendices_latex_code(
117         main_non_code_appendix_inclusion_lines,
118         sorted_created_notebook_appendices,
119         project_nr,
120         sorted_created_python_appendices,
121     )
122     updated_main_tex_code = substitute_appendix_code(
123         end_index, main_tex_code, start_index, appendix_latex_code
124     )
125     print(f"\n\n")
126     print(f"updated_main_tex_code={updated_main_tex_code}")
127
128     overwrite_content_to_file(updated_main_tex_code,
129         ↪ path_to_main_latex_file)
130
131     def create_appendices_latex_code(
132         main_non_code_appendix_inclusion_lines,
133         notebook_appendices,
134         project_nr,
135         python_appendices,
136     ):
137         """Creates the latex code that includeds the appendices in the
138             ↪ main latex file.
139
140         :param main_non_code_appendix_inclusion_lines: latex code that
141             ↪ includes the appendices that do not contain python code nor
142             ↪ notebooks
143         :param notebook_appendices: List of Appendix objects representing
144             ↪ appendices that include the pdf files of compiled Jupiter
145             ↪ notebooks
146         :param project_nr: The number indicating which project this code
147             ↪ pertains to.
148         :param python_appendices: List of Appendix objects representing
149             ↪ appendices that include the python code files.
150         """
151         main_appendix_inclusion_lines =
152             ↪ main_non_code_appendix_inclusion_lines
153         print(f"main_appendix_inclusion_lines={
154             ↪ main_appendix_inclusion_lines}")
155

```

```

147     appendices_of_all_types = [python_appendices, notebook_appendices
148         ↪ ]
149
150     print(f"\n\n")
151     main_appendix_inclusion_lines.append(
152         f"\IfFileExists{{latex/project{project_nr}/main.tex}}{{{"
153     )
154     main_appendix_inclusion_lines = append_latex_inclusion_command(
155         appendices_of_all_types,
156         True,
157         main_appendix_inclusion_lines,
158         project_nr,
159     )
160     main_appendix_inclusion_lines.append(f"}}{{{"")
161     main_appendix_inclusion_lines = append_latex_inclusion_command(
162         appendices_of_all_types,
163         False,
164         main_appendix_inclusion_lines,
165         project_nr,
166     )
167     # main_appendix_inclusion_lines.append(f"}}")
168     print(f"main_appendix_inclusion_lines={
169         ↪ main_appendix_inclusion_lines}")
170     return main_appendix_inclusion_lines
171
172 def append_latex_inclusion_command(
173     appendices_of_all_types, is_from_root_dir,
174     ↪ main_appendix_inclusion_lines, project_nr
175 ):
176     for appendix_type in appendices_of_all_types:
177         for appendix in appendix_type:
178             line = update_appendix_tex_code(
179                 appendix.appendix_filename, is_from_root_dir,
180                 ↪ project_nr
181             )
182             print(f"appendix.appendix_filename={appendix.
183                 ↪ appendix_filename}")
184             main_appendix_inclusion_lines.append(line)
185     return main_appendix_inclusion_lines
186
187 def filter_appendices_by_type(appendices, appendix_type):
188     """Returns the list of all appendices of a certain appendix type,
189     ↪ from the incoming list of Appendix objects.
190
191     :param appendices: List of Appendix objects
192     :param appendix_type: Can consist of "no_code", "python", or "
193     ↪ notebook" and indicates different appendix types
194     """
195     return_appendices = []
196     for appendix in appendices:
197         if appendix.appendix_type == appendix_type:
198             return_appendices.append(appendix)
199     return return_appendices
200
201 def sort_python_appendices(appendices):
202     """First puts __main__.py, followed by main.py followed by a-z
203     ↪ code files.
204
205     :param appendices: List of Appendix objects

```

```

201 """
202 return_appendices = []
203 for appendix in appendices: # first get appendix containing
    ↳ __main__.py
204     if (appendix.code_filename == "__main__.py") or (
205         appendix.code_filename == "__Main__.py"
206     ):
207         return_appendices.append(appendix)
208         appendices.remove(appendix)
209 for appendix in appendices: # second get appendix containing
    ↳ main.py
210     if (appendix.code_filename == "main.py") or (
211         appendix.code_filename == "Main.py"
212     ):
213         return_appendices.append(appendix)
214         appendices.remove(appendix)
215 return_appendices
216
217 # Filter remaining appendices in order of a-z
218 filtered_remaining_appendices = [
219     i for i in appendices if i.code_filename is not None
220 ]
221 appendices_sorted_a_z = sort_appendices_on_code_filename(
222     filtered_remaining_appendices
223 )
224 return return_appendices + appendices_sorted_a_z
225
226
227 def sort_notebook_appendices_alphabetically(appendices):
228     """Sorts notebook appendix objects alphabetic order of their pdf
    ↳ filenames.
229
230     :param appendices: List of Appendix objects
231     """
232     return_appendices = []
233     filtered_remaining_appendices = [
234         i for i in appendices if i.code_filename is not None
235     ]
236     appendices_sorted_a_z = sort_appendices_on_code_filename(
237         filtered_remaining_appendices
238     )
239     return return_appendices + appendices_sorted_a_z
240
241
242 def sort_appendices_on_code_filename(appendices):
243     """Returns a list of Appendix objects that are sorted and based
    ↳ on the property: code_filename.
244     Assumes the incoming appendices only contain python files.
245
246     :param appendices: List of Appendix objects
247     """
248     attributes = list(map(lambda x: x.code_filename, appendices))
249     sorted_indices = sorted(range(len(attributes)), key=lambda k:
    ↳ attributes[k])
250     sorted_list = []
251     for i in sorted_indices:
252         sorted_list.append(appendices[i])
253     return sorted_list
254
255
256 def get_order_of_non_code_appendices_in_main(appendices,
    ↳ appendix_tex_code):

```

```

257 """Scans the lines of appendices in the main code, and returns
258     ↳ the lines
259 of the appendices that do not contain code, in the order in which
260     ↳ they were
261 included in the main latex file.
262
263 :param appendices: List of Appendix objects
264 :param appendix_tex_code: latex code from the main latex file
265     ↳ that includes the appendices
266 """
267 non_code_appendices = []
268 non_code_appendix_lines = []
269 appendix_tex_code = list(dict.fromkeys(appendix_tex_code))
270 for line in appendix_tex_code:
271     appendix_filename = get_filename_from_latex_appendix_line(
272         ↳ appendices, line)
273
274     # Check if line is not commented
275     if not appendix_filename is None:
276         if not line_is_commented(line, appendix_filename):
277             appendix = get_appendix_from_filename(appendices,
278                 ↳ appendix_filename)
279             if appendix.appendix_type == "no_code":
280                 non_code_appendices.append(appendix)
281                 non_code_appendix_lines.append(line)
282 return non_code_appendices, non_code_appendix_lines
283
284 def get_filename_from_latex_appendix_line(appendices, appendix_line):
285     """Returns the first filename from a list of incoming filenames
286     ↳ that
287 occurs in a latex code line.
288
289 :param appendices: List of Appendix objects
290 :param appendix_line: latex code (in particular expected to be
291     ↳ the code from main that is used to include appendix latex
292     ↳ files.)
293 """
294 for filename in list(map(lambda appendix: appendix.
295     ↳ appendix_filename, appendices)):
296     if filename in appendix_line:
297         if not line_is_commented(appendix_line, filename):
298             return filename
299
300 def get_appendix_from_filename(appendices, appendix_filename):
301     """Returns the first Appendix object with an appendix filename
302     ↳ that matches the incoming appendix_filename.
303 The Appendix objects are selected from an incoming list of
304     ↳ Appendix objects.
305
306 :param appendices: List of Appendix objects
307 :param appendix_filename: name of a latex appendix file, ends in
308     ↳ .tex,
309 """
310 for appendix in appendices:
311     if appendix_filename == appendix.appendix_filename:
312         return appendix
313
314 def get_compiled_notebook_paths(script_dir):

```

```

306 """Returns the list of jupyter notebook filepaths that were
307     ↳ compiled successfully and that are
308     included in the same dias this script (the src directory).
309
310 :param script_dir: absolute path of this file.
311 """
312 notebook_filepaths = get_filenames_in_dir(".ipynb", script_dir)
313 compiled_notebook_filepaths = []
314
315 # check if the jupyter notebooks were compiled
316 for notebook_filepath in notebook_filepaths:
317     # swap file extension
318     notebook_filepath = notebook_filepath.replace(".ipynb", ".pdf"
319     ↳ ")
320
321     # check if file exists
322     if os.path.isfile(notebook_filepath):
323         compiled_notebook_filepaths.append(notebook_filepath)
324
325 return compiled_notebook_filepaths
326
327 def get_list_of_appendix_files(
328     appendix_dir, absolute_notebook_filepaths,
329     ↳ absolute_python_filepaths
330 ):
331     """Returns a list of Appendix objects that contain all the
332     ↳ appendix files with .tex extension.
333
334     :param appendix_dir: Absolute path that contains the appendix .
335     ↳ tex files.
336     :param absolute_notebook_filepaths: List of absolute paths to the
337     ↳ compiled notebook pdf files.
338     :param absolute_python_filepaths: List of absolute paths to the
339     ↳ python files.
340     """
341     appendices = []
342     appendices_paths = get_filenames_in_dir(".tex", appendix_dir)
343
344     for appendix_filepath in appendices_paths:
345         appendix_type = "no_code"
346         appendix_filecontent = read_file(appending_filepath)
347         line_nr_python_file_inclusion = get_line_of_latex_command(
348             appendix_filecontent, "\pythonexternal{"
349         )
350         line_nr_notebook_file_inclusion = get_line_of_latex_command(
351             appendix_filecontent, "\includepdf[pages="
352         )
353         if line_nr_python_file_inclusion > -1:
354             appendix_type = "python"
355             # get python filename
356             line = appendix_filecontent[line_nr_python_file_inclusion
357             ↳ ]
358             filename = get_filename_from_latex_inclusion_command(
359                 line, ".py", "\pythonexternal{"

```

```

360         line,
361     )
362 )
363 if line_nr_notebook_file_inclusion > -1:
364     appendix_type = "notebook"
365     line = appendix_filecontent[
366         ↪ line_nr_notebook_file_inclusion]
367     filename = get_filename_from_latex_inclusion_command(
368         line, ".pdf", "\includepdf[pages="
369     )
370     appendices.append(
371         Appendix(
372             appendix_filepath,
373             appendix_filecontent,
374             appendix_type,
375             filename,
376             line,
377         )
378     )
379 else:
380     appendices.append(
381         Appendix(appending_filepath, appendix_filecontent,
382             ↪ appendix_type)
383     )
384 return appendices
385
386 def get_filename_from_latex_inclusion_command(
387     appendix_line, extension, start_substring
388 ):
389     """returns the code/notebook filename in a latex command which
390     ↪ includes that code in an appendix.
391     The inclusion command includes a python code or jupyter notebook
392     ↪ pdf.
393
394     :param appendix_line: :Line of latex code (in particular expected
395     ↪ to be the latex code from an appendix.).
396     :param extension: The file extension of the file that is sought
397     ↪ in the appendix line. Either ".py" or ".pdf".
398     :param start_substring: The substring that characterises the
399     ↪ latex inclusion command.
400     """
401     start_index = appendix_line.index(start_substring)
402     end_index = appendix_line.index(extension)
403     return get_filename_from_dir(
404         appendix_line[start_index : end_index + len(extension)]
405     )
406
407 def get_filenames_in_dir(extension, path, excluded_files=None):
408     """Returns a list of the relative paths to all files within the
409     ↪ some path that match
410     the given file extension.
411
412     :param extension: The file extension of the file that is sought
413     ↪ in the appendix line. Either ".py" or ".pdf".
414     :param path: Absolute filepath in which files are being sought.
415     :param excluded_files: (Default value = None) Files that will not
416     ↪ be included even if they are found.
417     """
418     filepaths = []
419     for r, d, f in os.walk(path):

```

```

412     for file in f:
413         if file.endswith(extension):
414             if (excluded_files is None) or (
415                 (not excluded_files is None) and (not file in
416                     ↪ excluded_files)
417             ):
418                 filepaths.append(r + "/" + file)
419
420     return filepaths
421
422 def get_code_files_already_included_in_appendices(
423     absolute_code_filepaths, appendix_dir, extension, project_nr,
424     ↪ root_dir
425 ):
426     """Returns a list of code filepaths that are already properly
427     ↪ included the latex appendix files of this project.
428
429     :param absolute_code_filepaths: List of absolute paths to the
430     ↪ code files (either python files or compiled jupyter
431     ↪ notebook pdfs).
432     :param appendix_dir: Absolute path that contains the appendix .
433     ↪ tex files.
434     :param extension: The file extension of the file that is sought
435     ↪ in the appendix line. Either ".py" or ".pdf".
436     :param project_nr: The number indicating which project this code
437     ↪ pertains to.
438     :param root_dir: The root directory of this repository.
439     """
440     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
441     contained_codes = []
442     for code_filepath in absolute_code_filepaths:
443         for appendix_filepath in appendix_files:
444             appendix_filecontent = read_file(appendix_filepath)
445             line_nr = check_if_appendix_contains_file(
446                 appendix_filecontent, code_filepath, extension,
447                 ↪ project_nr, root_dir
448             )
449             if line_nr > -1:
450                 # add filepath to list of files that are already in
451                 ↪ the appendices
452                 contained_codes.append(
453                     Appendix_with_code(
454                         code_filepath,
455                         appendix_filepath,
456                         appendix_filecontent,
457                         line_nr,
458                         ".py",
459                     )
460                 )
461     return contained_codes
462
463 def check_if_appendix_contains_file(
464     appendix_content, code_filepath, extension, project_nr, root_dir
465 ):
466     """Scans an appendix content to determine whether it contains a
467     ↪ substring that
468     includes a code file (of either python or compiled notebook=pdf
469     ↪ extension).
470
471     :param appendix_content: content in an appendix latex file.

```



```

461 :param code_filepath: Absolute path to a code file (either python
    ↪ files or compiled jupyter notebook pdfs).
462 :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".
463 :param project_nr: The number indicating which project this code
    ↪ pertains to.
464 :param root_dir: The root directory of this repository.
465 """
466 # convert code_filepath to the inclusion format in latex format
467 latex_relative_filepath = (
468     f"latex/project{project_nr}/../../{code_filepath[len(root_dir)
    ↪ ):]}"
469 )
470 latex_command = get_latex_inclusion_command(extension,
    ↪ latex_relative_filepath)
471 return get_line_of_latex_command(appendix_content, latex_command)
472
473 def get_line_of_latex_command(appendix_content, latex_command):
474     """Returns the line number of a latex command if it is found.
475     ↪ Returns -1 otherwise.
476
477     :param appendix_content: content in an appendix latex file.
478     :param latex_command: A line of latex code. (Expected to come
    ↪ from some appendix)
479     """
480     # check if the file is in the latex code
481     line_nr = 0
482     for line in appendix_content:
483         if latex_command in line:
484             if line_is_commented(line, latex_command):
485                 commented = True
486             else:
487                 return line_nr
488             line_nr = line_nr + 1
489     return -1
490
491 def line_is_commented(line, target_substring):
492     """Returns True if a latex code line is commented, returns False
    ↪ otherwise
493
494     :param line: A line of latex code that contains a relevant
    ↪ command (target substring).
495     :param target_substring: Used to determine whether the command
    ↪ that is found is commented or not.
496     """
497     left_of_command = line[: line.rfind(target_substring)]
498     if "%" in left_of_command:
499         return True
500     return False
501
502 def get_latex_inclusion_command(extension,
    ↪ latex_relative_filepath_to_codefile):
503     """Creates and returns a latex command that includes either a
    ↪ python file or a compiled jupyter
504     notebook pdf (wherever the command is placed). The command is
    ↪ intended to be placed in the appendix.
505
506     :param extension: The file extension of the file that is sought
    ↪ in the appendix line. Either ".py" or ".pdf".

```

```

509 :param latex_relative_filepath_to_codefile: The latex compilation
510 ↪ requires a relative path towards code files
511 that are included. Therefore, a relative path towards the code is
512 ↪ given.
513 """
514 if extension == ".py":
515     left = "\pythonexternal{"
516     right = "}"
517     latex_command = f"{left}{latex_relative_filepath_to_codefile}
518 ↪ {right}"
519 elif extension == ".ipynb":
520     left = "\includepdf[pages=-]"
521     right = "}"
522     latex_command = f"{left}{latex_relative_filepath_to_codefile}
523 ↪ {right}"
524 return latex_command
525
526 def read_file(filepath):
527     """Reads content of a file and returns it as a list of strings,
528     ↪ with one string per line.
529
530     :param filepath: path towards the file that is being read.
531     """
532     with open(filepath) as f:
533         content = f.readlines()
534     return content
535
536 def get_code_files_not_yet_included_in_appendices(
537     code_filepaths, contained_codes, extension
538 ):
539     """Returns a list of filepaths that are not yet properly included
540     ↪ in some appendix of this project.
541
542     :param code_filepath: Absolute path to all the code files in
543     ↪ this project (source directory).
544     (either python files or compiled jupyter notebook pdfs).
545     :param contained_codes: list of Appendix objects that include
546     ↪ either python files or compiled jupyter notebook pdfs,
547     ↪ which
548     are already included in the appendix tex files. (Does not care
549     ↪ whether those appendices are also actually
550     included in the main or not.)
551     :param extension: The file extension of the file that is sought
552     ↪ in the appendix line. Either ".py" or ".pdf".
553     """
554     contained_filepaths = list(
555         map(lambda contained_file: contained_file.code_filepath,
556 ↪ contained_codes)
557     )
558     not_contained = []
559     for filepath in code_filepaths:
560         if not filepath in contained_filepaths:
561             not_contained.append(filepath)
562     return not_contained
563
564 def create_appendices_with_code(
565     appendix_dir, code_filepaths, extension, project_nr, root_dir
566 ):

```

```

559 """Creates the latex appendix files in with relevant codes
    ↳ included.
560
561 :param appendix_dir: Absolute path that contains the appendix .
    ↳ tex files.
562 :param code_filepaths: Absolute path to code files that are not
    ↳ yet included in an appendix
563 (either python files or compiled jupyter notebook pdfs).
564 :param extension: The file extension of the file that is sought
    ↳ in the appendix line. Either ".py" or ".pdf".
565 :param project_nr: The number indicating which project this code
    ↳ pertains to.
566 :param root_dir: The root directory of this repository.
567 """
568 appendix_filenames = []
569 appendix_reference_index = (
570     get_index_of_auto_generated_appendices(appendix_dir,
    ↳ extension) + 1
571 )
572
573 for code_filepath in code_filepaths:
574     latex_relative_filepath = (
575         f"latex/project{project_nr}/../../{code_filepath[len(
    ↳ root_dir):]}"
576     )
577     code_path_from_latex_main_path = f"../../{code_filepath[len(
    ↳ root_dir):]}"
578     content = []
579     filename = get_filename_from_dir(code_filepath)
580
581     content = create_section(appendix_reference_index, filename,
    ↳ content)
582     content = add_include_code_in_appendix(
583         content,
584         code_filepath,
585         code_path_from_latex_main_path,
586         extension,
587         latex_relative_filepath,
588         project_nr,
589         root_dir,
590     )
591
592     print(f"content={content}")
593
594     overwrite_content_to_file(
595         content,
596         f"{appendix_dir}Auto_generated_{extension[1:]}_App{
    ↳ appendix_reference_index}.tex",
597         False,
598     )
599     appendix_filenames.append(
600         f"Auto_generated_{extension[1:]}_App{
    ↳ appendix_reference_index}.tex"
601     )
602     appendix_reference_index = appendix_reference_index + 1
603 return appendix_filenames
604
605
606 def add_include_code_in_appendix(
607     content,
608     code_filepath,
609     code_path_from_latex_main_path,

```

```

610     extension,
611     latex_relative_filepath,
612     project_nr,
613     root_dir,
614 ):
615     """Includes the latex code that includes code in the script.
616
617     :param content: The latex content that is being written to an
618         ↪ appendix.
619     :param code_path_from_latex_main_path: the path to the code as
620         ↪ seen from the folder that contains main.tex.
621     :param extension: The file extension of the file that is sought
622         ↪ in the appendix line. Either ".py" or ".pdf".
623     :param latex_relative_filepath_to_codefile: The latex compilation
624         ↪ requires a relative path towards code files
625         that are included. Therefore, a relative path towards the code is
626         ↪ given.
627     """
628     print(f"before={content}")
629     # TODO: append if exists
630     content.append(
631         f"\IfFileExists{{latex/project{project_nr}/../../{
632             ↪ code_filepath[len(root_dir):]}}}{{"
633     )
634     # append current line
635     content.append(get_latex_inclusion_command(extension,
636         ↪ latex_relative_filepath))
637     # TODO: append {}
638     content.append(f"}}{{{"")
639     # TODO: code_path_from latex line
640     content.append(
641         get_latex_inclusion_command(extension,
642             ↪ code_path_from_latex_main_path)
643     )
644     # TODO: add closing bracket }
645     content.append(f"}}}")
646     print(f"after={content}")
647     return content
648
649 def get_index_of_auto_generated_appendices(appendix_dir, extension):
650     """Returns the maximum index of auto generated appendices of
651     a specific extension type.
652
653     :param extension: The file extension of the file that is sought
654         ↪ in the appendix line. Either ".py" or ".pdf".
655     :param appendix_dir: Absolute path that contains the appendix .
656         ↪ tex files.
657     """
658     max_index = -1
659     appendices =
660         ↪ get_auto_generated_appendix_filenames_of_specific_extension
661         ↪ (
662             appendix_dir, extension
663         )
664     for appendix in appendices:
665         substring = f"Auto_generated_{extension[1:]}_App"
666         # remove left of index
667         remainder = appendix[appendix.rfind(substring) + len(
668             ↪ substring) :]
669         # remove right of index
670         index = int(remainder[:-4])

```

```

659         if index > max_index:
660             max_index = index
661     return max_index
662
663
664 def get_auto_generated_appendix_filenames_of_specific_extension(
665     appendix_dir, extension
666 ):
667     """Returns the list of auto generated appendices of
668     a specific extension type.
669
670     :param extension: The file extension of the file that is sought
671         ↳ in the appendix line. Either ".py" or ".pdf".
672     :param appendix_dir: Absolute path that contains the appendix .
673         ↳ tex files.
674     """
675     appendices_of_extension_type = []
676
677     # get all appendices
678     appendix_files = get_filenames_in_dir(".tex", appendix_dir)
679
680     # get appendices of particular extension type
681     for appendix_filepath in appendix_files:
682         right_of_slash = appendix_filepath[appendix_filepath.rfind("/")
683             ↳ ") + 1 :]
684         if (
685             right_of_slash[: 15 + len(extension) - 1]
686             == f"Auto_generated_{extension[1:]}"
687         ):
688             appendices_of_extension_type.append(appendix_filepath)
689     return appendices_of_extension_type
690
691 def create_section(appendix_reference_index, code_filename, content):
692     """Creates the header of a latex appendix file, such that it
693     ↳ contains a section that
694     indicates the section is an appendix, and indicates which python
695     ↳ or notebook file is
696     being included in that appendix.
697
698     :param appendix_reference_index: A counter that is used in the
699     ↳ label to ensure the appendix section labels are unique.
700     :param code_filename: file name of the code file that is included
701     :param content: A list of strings that make up the appendix, with
702     ↳ one line per element.
703     """
704     # write section
705     left = "\section{Appendix "
706     middle = code_filename.replace("-", "\-")
707     right = "}\label{app:"
708     end = "}" # TODO: update appendix reference index
709     content.append(f"{left}{middle}{right}{appendix_reference_index}{
710         ↳ end}")
711     return content
712
713 def overwrite_content_to_file(content, filepath, content_has_newlines
714     ↳ =True):
715     """Writes a list of lines of tex code from the content argument
716     ↳ to a .tex file
717     using overwriting method. The content has one line per element.

```

```

711 :param content: The content that is being written to file.
712 :param filepath: Path towards the file that is being read.
713 :param content_has_newlines: (Default value = True)
714 """
715 with open(filepath, "w") as f:
716     for line in content:
717         if content_has_newlines:
718             f.write(line)
719         else:
720             f.write(line + "\n")
721
722
723 def get_appendix_tex_code(main_latex_filename):
724     """gets the latex appendix code from the main tex file.
725
726     :param main_latex_filename: Name of the main latex document of
727         ↳ this project number
728     """
729     main_tex_code = read_file(main_latex_filename)
730     start = "\\begin{appendices}"
731     end = "\\end{appendices}"
732     start_index = get_index_of_substring_in_list(main_tex_code, start
733         ↳ ) + 1
734     end_index = get_index_of_substring_in_list(main_tex_code, end)
735     return main_tex_code, start_index, end_index, main_tex_code[
736         ↳ start_index:end_index]
737
738
739 def get_index_of_substring_in_list(lines, target_substring):
740     """Returns the index of the line in which the first character of
741         ↳ a latex substring if it is found
742         ↳ uncommented in the incoming list.
743
744     :param lines: List of lines of latex code.
745     :param target_substring: Some latex command/code that is sought
746         ↳ in the incoming text.
747     """
748     for i in range(0, len(lines)):
749         if target_substring in lines[i]:
750             if not line_is_commented(lines[i], target_substring):
751                 return i
752
753
754 def update_appendix_tex_code(appendix_filename, is_from_root_dir,
755     ↳ project_nr):
756     """Returns the latex command that includes an appendix .tex file
757         ↳ in an appendix environment
758         ↳ as can be used in the main tex file.
759
760     :param appendix_filename: Name of the appendix that is included
761         ↳ by the generated command.
762     :param project_nr: The number indicating which project this code
763         ↳ pertains to.
764     """
765     if is_from_root_dir:
766         left = f"\\input{{latex/project{project_nr}}/"
767     else:
768         left = "\\input{"
769     middle = "Appendices/"
770     right = "} \\newpage\n"
771     return f"{left}{middle}{appendix_filename}{right}"

```

```

764 def substitute_appendix_code(
765     end_index, main_tex_code, start_index,
766     ↪ updated_appendices_tex_code
767 ):
768     """Replaces the old latex code that included the appendices in
769     ↪ the main.tex file with the new latex
770     commands that include the appendices in the latex report.
771
772     :param end_index: Index at which the appendix section ends right
773     ↪ before the latex \end{appendix} line,
774     :param main_tex_code: The code that is saved in the main .tex
775     ↪ file.
776     :param start_index: Index at which the appendix section starts
777     ↪ right after the latex \begin{appendix} line,
778     :param updated_appendices_tex_code: The newly created code that
779     ↪ includes all the relevant appendices.
780     (relevant being (in order): manually created appendices, python
781     ↪ codes, pdfs of compiled jupyter notebooks).
782     """
783     updated_main_tex_code = (
784         main_tex_code[0:start_index]
785         + updated_appendices_tex_code
786         + main_tex_code[end_index:]
787     )
788     print(f"start_index={start_index}")
789     return updated_main_tex_code
790
791 def get_filename_from_dir(path):
792     """Returns a filename from an absolute path to a file.
793
794     :param path: path to a file of which the name is queried.
795     """
796     return path[path.rfind("/") + 1 :]
797
798 def get_script_dir():
799     """returns the directory of this script regardless of from which
800     ↪ level the code is executed"""
801     return os.path.dirname(__file__)
802
803 class Appendix_with_code:
804     """stores in which appendix file and accompanying line number in
805     ↪ the appendix in which a code file is
806     already included. Does not take into account whether this
807     ↪ appendix is in the main tex file or not
808     """
809
810     def __init__(
811         self,
812         code_filepath,
813         appendix_filepath,
814         appendix_content,
815         file_line_nr,
816         extension,
817     ):
818         self.code_filepath = code_filepath
819         self.appendix_filepath = appendix_filepath
820         self.appendix_content = appendix_content
821         self.file_line_nr = file_line_nr

```

```

816         self.extension = extension
817
818
819 class Appendix:
820     """stores in appendix files and type of appendix."""
821
822     def __init__(
823         self,
824         appendix_filepath,
825         appendix_content,
826         appendix_type,
827         code_filename=None,
828         appendix_inclusion_line=None,
829     ):
830         self.appendix_filepath = appendix_filepath
831         self.appendix_filename = get_filename_from_dir(self.
            ↳ appendix_filepath)
832         self.appendix_content = appendix_content
833         self.appendix_type = appendix_type # TODO: perform
            ↳ validation of input values
834         self.code_filename = code_filename
835         self.appendix_inclusion_line = appendix_inclusion_line

```

H Appendix Gantt.py

```
1 # The bottom up model that computes the TAM and TSM
2 import random
3 import numpy as np
4
5 from .Plot_to_tex import Plot_to_tex as plt_tex
6 from .Create_python_gantt import create_python_gantt
7
8
9 class Gantt:
10     def __init__(self, filepath):
11         self.start_line = "@startgantt"
12         self.project_start_date = "2021/07-22"
13         self.closed_days = ["saturday", "sunday"]
14         self.gantt_font_size="skinparam classFontSize 100"
15         self.box_font_size="30"
16
17         #self.font_size="skinparam defaultFontSize 100"
18
19         self.parents = create_python_gantt()
20         self.end_line = "@endgantt"
21         self.lines = self.get_list()
22         self.write_gantt(filepath, self.lines)
23
24         self.costs = None
25
26     def get_list(self):
27         lines = []
28         lines.append(self.start_line)
29         lines.append(f"project starts the {self.project_start_date}")
30         lines = self.add_closed_dates(lines)
31         lines.append(self.gantt_font_size)
32         # lines.append(f"[{parent.description}] as [{parent.get_tag()}
33         ↪ ↪ ] lasts {parent.duration} days")
34         # for parent in self.parents:
35         lines = self.loop_through_parents_printing(lines)
36         lines.append(self.end_line)
37         return lines
38
39     def loop_through_parents_printing(self, lines):
40         # print descriptions
41         for i in range(0, len(self.parents)):
42             lines = self.print_parent_descriptions(lines, self.
43             ↪ ↪ parents[i])
44
45         # print order
46         for i in range(0, len(self.parents)):
47             if i > 0:
48                 if not self.parents[i].starts_at_child_nr_start is
49                 ↪ ↪ None:
50                     #print(f'starting parent at:{self.parents[i].
51                     ↪ ↪ starts_at_child_nr_start}')
52                     lines.append(
53                         f"[{self.parents[i].get_tag()}] starts at [{
54                         ↪ ↪ self.parents[i].
55                         ↪ ↪ starts_at_child_nr_start}]'s start"
56                     )
57             else:
58                 print(f'parent_descr={self.parents[i].description
59                 ↪ ↪ }, and starts at {self.parents[i].
60                 ↪ ↪ starts_at_child_nr_start} and tag={self.
```

```

53         ↪ parents[i].get_tag()}, writing end')
54         lines.append(
55             f"[{self.parents[i].get_tag()}] starts at [{
56                 ↪ self.parents[i-1].get_tag()}]'s end"
57         )
58         lines = self.print_parent_order(lines, self.parents[i])
59
60     # print colour
61     for i in range(0, len(self.parents)):
62         lines = self.print_parent_colour(lines, self.parents[i])
63
64     # compute costs
65     total_costs = 0
66     for i in range(0, len(self.parents)):
67         total_costs, lines = self.print_parent_costs(total_costs,
68             ↪ lines, self.parents[i])
69     return lines
70
71 def print_parent_descriptions(self, lines, parent):
72     lines.append("")
73     lines = self.print_descriptions(parent, lines)
74     return lines
75
76 def print_parent_order(self, lines, parent):
77     lines.append("")
78     lines = self.print_order(parent, lines)
79     return lines
80
81 def print_parent_colour(self, lines, parent):
82     lines.append("")
83     lines = self.print_colour(parent, lines)
84     lines.append("")
85     return lines
86
87 def print_parent_costs(self, total_costs, lines, parent):
88     lines.append("")
89     total_costs, lines = self.print_costs(total_costs, parent,
90         ↪ lines)
91     print(f'total_costs={total_costs}')
92     lines.append("")
93     return total_costs, lines
94
95 def print_descriptions(self, activity, lines):
96
97     if not activity.font_size is None:
98         font_size = activity.font_size
99     else:
100         font_size=self.box_font_size
101     lines.append(
102         f"<size:{font_size}>{activity.description}] as [{
103             ↪ activity.get_tag()}] lasts {activity.duration} days
104         ↪ "
105     )
106     for child in activity.children:
107         lines = self.print_descriptions(child, lines)
108     return lines
109
110 def print_order(self, activity, lines):
111     # Write order for all children in an activity
112     for i in range(0, len(activity.children)):
113         if i == 0:
114             lines.append(

```

```

109         f"[{activity.children[i].get_tag()}] starts at [{
110             ↳ activity.get_tag()}]'s start"
111     )
112     else:
113         # check if it starts at the start or end of some
114         ↳ activity
115         if not activity.children[i].starts_at_child_nr_start
116         ↳ is None:
117             #print(f'activity.description={activity.
118             ↳ description}, appending at {activity.
119             ↳ children[i].starts_at_child_nr_start}s
120             ↳ start')
121             lines.append(
122                 f"[{activity.children[i].get_tag()}] starts
123                 ↳ at [{activity.children[i].
124                 ↳ starts_at_child_nr_start}]'s start"
125             )
126         else:
127             #print(f'activity.description={activity.
128             ↳ description}, appending at {activity.
129             ↳ children[i-1].get_tag()}s end')
130             lines.append(
131                 f"[{activity.children[i].get_tag()}] starts
132                 ↳ at [{activity.children[i-1].get_tag()
133                 ↳ }] 's end"
134             )
135
136     # start recursive loop to write order of each of the
137     ↳ children
138     for child in activity.children:
139         lines = self.print_order(child, lines)
140     return lines
141
142 def print_colour(self, activity, lines):
143     lines.append(f"[{activity.get_tag()}] is colored in {
144         ↳ activity.colour}")
145     for child in activity.children:
146         lines = self.print_colour(child, lines)
147     return lines
148
149 def print_costs(self, total_costs, activity, lines):
150     lines.append(f"'[{activity.description}] takes: {activity.
151         ↳ duration}[days] equating to:{activity.duration*activity.
152         ↳ .hours_per_day}[hours] and costs: {activity.hourly_wage
153         ↳ } per hour, yielding activity costs: {activity.duration
154         ↳ *activity.hours_per_day*activity.hourly_wage} Euros.")
155     total_costs=total_costs+activity.duration*activity.
156     ↳ hours_per_day*activity.hourly_wage
157     for child in activity.children:
158         total_costs, lines = self.print_costs(total_costs, child,
159         ↳ lines)
160     return total_costs, lines
161
162 def add_closed_dates(self, lines):
163     for closed_day in self.closed_days:
164         lines.append(f"{closed_day} are closed")
165     return lines
166
167 def write_gantt(self, filepath, list):
168     with open(filepath, "w") as f:
169         for item in list:
170             f.write("%s\n" % item)

```

```
151         f.close()
152
153     def addTwo(self, x):
154         """adds two to the incoming integer and returns the result of
155             ↪ the computation."""
156         return x + 2
```

I Appendix Plot_to_tex.py

```
1  ### Call this from another file, for project 11, question 3b:
2  ### from Plot_to_tex import Plot_to_tex as plt_tex
3  ### multiple_y_series = np.zeros((nrOfDataSeries,nrOfDataPoints),
   ↪ dtype=int); # actually fill with data
4  ### lineLabels = [] # add a label for each dataseries
5  ### plt_tex.plotMultipleLines(plt_tex,single_x_series,
   ↪ multiple_y_series,"x-axis label [units]","y-axis label [units]
   ↪ ",lineLabels,"3b",4,11)
6  ### 4b=filename
7  ### 4 = position of legend, e.g. top right.
8  ###
9  ### For a single line, use:
10 ### plt_tex.plotSingleLine(plt_tex,range(0, len(dataseries)),
   ↪ dataseries,"x-axis label [units]","y-axis label [units]",
   ↪ lineLabel,"3b",4,11)
11
12 ### You can also plot a table directly into latex, see
   ↪ example_create_a_table(..)
13 ###
14 ### Then put it in latex with for example:
15 ### \begin{table}[H]
16 ###     \centering
17 ###     \caption{Results some computation.}\label{tab:some_computation
   ↪ }
18 ###     \begin{tabular}{|c|c|} % remember to update this to show all
   ↪ columns of table
19 ###         \hline
20 ###         \input{latex/project3/tables/q2.txt}
21 ###     \end{tabular}
22 ### \end{table}
23 import random
24 from matplotlib import lines
25 import matplotlib.pyplot as plt
26 import numpy as np
27 import os
28
29
30 class Plot_to_tex:
31     def __init__(self):
32         self.script_dir = self.get_script_dir()
33         print("Created main")
34
35     # plot graph (legendPosition = integer 1 to 4)
36     def plotSingleLine(
37         self,
38         x_path,
39         y_series,
40         x_axis_label,
41         y_axis_label,
42         label,
43         filename,
44         legendPosition,
45         project_nr,
46     ):
47         fig = plt.figure()
48         ax = fig.add_subplot(111)
49         ax.plot(x_path, y_series, c="b", ls="--", label=label,
   ↪ fillstyle="none")
50         plt.legend(loc=legendPosition)
51         plt.xlabel(x_axis_label)
```

```

52     plt.ylabel(y_axis_label)
53     plt.savefig(
54         os.path.dirname(__file__)
55         + "/../../../latex/project"
56         + str(project_nr)
57         + "/Images/"
58         + filename
59         + ".png"
60     )
61
62     #         plt.show();
63
64     # plot graphs
65     def plotMultipleLines(
66         self, x, y_series, x_label, y_label, label, filename,
67         ↪ legendPosition, project_nr
68     ):
69         fig = plt.figure()
70         ax = fig.add_subplot(111)
71
72         # generate colours
73         cmap = self.get_cmap(len(y_series[:, 0]))
74
75         # generate line types
76         lineTypes = self.generateLineTypes(y_series)
77
78         for i in range(0, len(y_series)):
79             # overwrite linetypes to single type
80             lineTypes[i] = "_"
81             ax.plot(
82                 x,
83                 y_series[i, :],
84                 ls=lineTypes[i],
85                 label=label[i],
86                 fillstyle="none",
87                 c=cmap(i),
88             )
89             # color
90
91         # configure plot layout
92         plt.legend(loc=legendPosition)
93         plt.xlabel(x_label)
94         plt.ylabel(y_label)
95         plt.savefig(
96             os.path.dirname(__file__)
97             + "/../../../latex/project"
98             + str(project_nr)
99             + "/Images/"
100             + filename
101             + ".png"
102         )
103
104         print(f"plotted lines")
105
106     # Generate random line colours
107     # Source: https://stackoverflow.com/questions/14720331/how-to-
108     ↪ generate-random-colors-in-matplotlib
109     def get_cmap(n, name="hsv"):
110         """Returns a function that maps each index in 0, 1, ..., n-1
111             ↪ to a distinct
112             RGB color; the keyword argument name must be a standard mpl
113             ↪ colormap name."""

```

```

110         return plt.cm.get_cmap(name, n)
111
112     def generateLineTypes(y_series):
113         # generate varying linetypes
114         typeOfLines = list(lines.lineStyles.keys())
115
116         while len(y_series) > len(typeOfLines):
117             typeOfLines.append("-.")
118
119         # remove void lines
120         for i in range(0, len(y_series)):
121             if typeOfLines[i] == "None":
122                 typeOfLines[i] = "-"
123             if typeOfLines[i] == ":":
124                 typeOfLines[i] = ":"
125             if typeOfLines[i] == " ":
126                 typeOfLines[i] = "--"
127         return typeOfLines
128
129     # Create a table with: table_matrix = np.zeros((4,4),dtype=object
130     ↪ ) and pass it to this object
131     def put_table_in_tex(self, table_matrix, filename, project_nr):
132         cols = np.shape(table_matrix)[1]
133         format = "%s"
134         for col in range(1, cols):
135             format = format + " & %s"
136         format = format + ""
137         plt.savetxt(
138             os.path.dirname(__file__)
139             + "/../../../latex/project"
140             + str(project_nr)
141             + "/tables/"
142             + filename
143             + ".txt",
144             table_matrix,
145             delimiter=" & ",
146             fmt=format,
147             newline=" \\\\ \hline \n",
148         )
149
150     # replace this with your own table creation and then pass it to
151     ↪ put_table_in_tex(..)
152     def example_create_a_table(self):
153         project_nr = "1"
154         table_name = "example_table_name"
155         rows = 2
156         columns = 4
157         table_matrix = np.zeros((rows, columns), dtype=object)
158         table_matrix[:, :] = "" # replace the standard zeros with
159         ↪ empty cell
160         print(table_matrix)
161         for column in range(0, columns):
162             for row in range(0, rows):
163                 table_matrix[row, column] = row + column
164         table_matrix[1, 0] = "example"
165         table_matrix[0, 1] = "grid sizes"
166
167         self.put_table_in_tex(table_matrix, table_name, project_nr)
168
169     def get_script_dir(self):
170         """returns the directory of this script regardless of from
171         ↪ which level the code is executed"""

```

```
168         return os.path.dirname(__file__)
169
170
171 if __name__ == "__main__":
172     main = Plot_to_tex()
173     main.example_create_a_table()
```
