Analytics - Product Quality

Date: 2020/02

SUMMARY:

- This notebook represents the project quality analysis of the date exposed right above.

TEAM:

Semester: 2023/01

Professor: Hilmer Neri

Members:

- Aline Lermen
- Caio Martins
- Dafne Moretti
- João Pedro
- João Vitor
- Lucas Gabriel
- Lucas Lima
- Eric Camargo
- Ester Flores
- Leonardo Ferreira
- Luana Torres
- Matheus Ferreira
- Pablo Guilherme
- Pedro Izarias
- Pedro Sena
- Suzane Duarte

Data de análise: [R1] - 26/05/2023 (Sprint 04)

LIBRARIES

In [63]:

```
# Deal with data
import pandas as pd
import json
from glob import glob
import os

# Deal with visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Deal with type hints
from typing import List

# Deal with time
```

```
import datetime
import openpyxl
```

GRAPH SETTINGS

In [29]:

```
%config InlineBackend.figure_format ='retina'
sns.set(font_scale=1.5)
sns.set_style('darkgrid',
              {'xtick.bottom' : True,
               'ytick.left': True,
               'grid.linestyle':'--',
               'font.monospace': ['Computer Modern Typewriter'],
               'axes.edgecolor' : 'white'})
```

DATAFRAME SETTINGS

In [30]:

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
```

Replace your semester, project name, repository name, and the programming language extension

In [31]:

```
# Set your repo major name here
# Example: fga-eps-mds-2022-1-MeasureSoftGram-
repo_name = 'fga-eps-mds-2023-1-Alectrion-'

# Add your repos here
# Example: 'Front': 'py',
repos_language = {
    'EquipamentApi': 'ts',
    'Gateway': 'ts',
    'UserAPI': 'ts',
    'FrontEnd': 'ts',
}
```

SonarCloud

Path to the folder with all your jsons

In [32]:

```
# Maybe you should change this path to your own path !!!!! PRESTAR ATENCAO !

sonar_files = glob('../analytics-raw-data/*.json')
```

Create DataFrame

Unmarshall json

In [33]:

```
def unmarshall(json_path: str) -> dict:
    with open(json_path) as json_file:
        json_obj = json.load(json_file)
    return json_obj
```

Create a list with all valid columns

In [34]:

```
metric_list = ['files',
               'functions',
               'complexity',
               'comment_lines_density',
               'duplicated_lines_density',
               'coverage',
               'ncloc',
               'tests',
               'test_errors',
               'test_failures',
               'test_execution_time',
               'security_rating']
```

Extract files dataframe out of component dataframe

In [35]:

```
def get_files_df(df: pd.DataFrame) -> pd.DataFrame:

    files_df = df[df['qualifier'] == 'FIL']

    files_df = files_df.dropna(subset=['functions', 'complexity','comment_li

    return files_df
```

Extract directories dataframe out of component dataframe

In [36]:

```
def get_dir_df(df: pd.DataFrame) -> pd.DataFrame:
    dirs = df[df["qualifier"] == "DIR"]

    newdf = pd.to_numeric(dirs["tests"])

    max_value_index = newdf.idxmax()

    return dirs.loc[max_value_index]
```

Extract uts dataframe out of component dataframe

In [37]:

```
def get_uts_df(df: pd.DataFrame) -> pd.DataFrame:
    uts_df = df[df['qualifier'] == 'UTS']

    uts_df = uts_df.fillna(0)

    uts_df = uts_df.dropna(subset=['test_execution_time'])

    return uts_df
```

Generate component dataframe

In [38]:

```
def metric_per_file(json_dict: dict) -> List[dict]:
    file_json = []

    for component in json_dict['components']:
        ncloc_value = 0
        for measure in component['measures']:
            if measure['metric'] == 'ncloc':
                ncloc_value = float(measure['value'])
                break

        if (component['qualifier'] == 'FIL' and ncloc_value > 0) \
                or component['qualifier'] == 'DIR' \
                or component['qualifier'] == 'UTS':
            file_json.append(component)
    return file_json


def generate_component_dataframe_data(
        metrics_list: List[str],
        file_component_data: List[dict],
        language_extension: str) -> pd.DataFrame:

    df_columns = metrics_list

    files_df = pd.DataFrame(columns = df_columns)
    dirs_df = pd.DataFrame(columns = df_columns)
    uts_df = pd.DataFrame(columns = df_columns)

    for file in file_component_data:
        try:
                if file['qualifier'] == 'FIL' and file['language'] == langua
                    for measure in file['measures']:
                        files_df.at[file['path'], measure['metric']] = measu

                    files_df['qualifier'] = file['qualifier']

                elif file['qualifier'] == 'DIR':
                    for measure in file['measures']:
                        dirs_df.at[file['path'], measure['metric']] = measu

                    dirs_df['qualifier'] = file['qualifier']

                elif file['qualifier'] == 'UTS':
                    for measure in file['measures']:
                        uts_df.at[file['path'], measure['metric']] = measure

                    uts_df['qualifier'] = file['qualifier']

        except:
            pass

    files_df.reset_index(inplace = True)
    dirs_df.reset_index(inplace = True)
    uts_df.reset_index(inplace = True)

    files_df = files_df.rename({'index': 'path'}, axis=1).drop(['files'], ax
    dirs_df = dirs_df.rename({'index': 'path'}, axis=1).drop(['files'], axis
    uts_df = uts_df.rename({'index': 'path'}, axis=1).drop(['files'], axis=

    df = pd.concat([files_df, dirs_df, uts_df], axis=0)

    return df


def create_component_df(json_list):
    df = pd.DataFrame()

    for json_path in json_list:
        file_component = unmarshall(json_path)
        file_component_data = metric_per_file(file_component)

        base_name = os.path.basename(json_path)

        file_component_dataframe = generate_component_dataframe_data(
            metric_list,
            file_component_data,
            language_extension = repos_language[base_name.split("-")[6]])
        file_component_dataframe['filename'] = base_name

        df = pd.concat([df, file_component_dataframe], ignore_index=True)

    aux_df = df['filename'].str.split(r"-(\d+-\d+-\d+-\d+-\d+-\d+)-(.*?).jsc

    df['repository'] = aux_df[0]
    df['datetime'] = aux_df[1]
    df['version'] = aux_df[2]

    df = df.sort_values(by=['repository', 'datetime'])

    return df
```

In [40]:

```
file_component_df = create_component_df(sonar_files)
file_component_df.repository.unique()
```

Out[40]:

```
array(['fga-eps-mds-2023-1-Alectrion-EquipamentApi',
       'fga-eps-mds-2023-1-Alectrion-FrontEnd',
       'fga-eps-mds-2023-1-Alectrion-UserAPI'], dtype=object)
```

Create dataframe per repository

In [41]:

```
repos_dataframes = []

for repo in repos_language.keys():
    dataframe = file_component_df[file_component_df['repository'] == repo_na
    repos_dataframes.append({'name': repo, 'df': dataframe})
```

Measure calculations according to Q-Rapids quality model

In [42]:

```
def _ncloc(df: pd.DataFrame) -> int:
    ncloc = 0
    for each in df['ncloc']:
        # try to cast the current ncloc value to int, if the value is NaN/N
        try:
            n = int(each)
        except ValueError:
            n = 0
        ncloc += n

    return ncloc
```

Quality Aspect - Maintainability

Factor - Code Quality

Complexity

In [43]:

```
def m1(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_non_complex_files = len(files_df[(files_df['complexity'].astype
                                                        files_df['functions'].astype(

    return density_non_complex_files
```

Comments

In [44]:
```
def m2(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_comment_files = len(files_df[(files_df['comment_lines_density']
                                         (files_df['comment_lines_density']

    return density_comment_files
```

Duplications

In [45]:
```
def m3(df: pd.DataFrame):

    files_df = get_files_df(df)

    duplication = len(files_df[(files_df['duplicated_lines_density'].astype

    return duplication
```

Quality Aspect - Reliability

Factor - Testing Status

Passed tests

In [46]:
```
def m4(df: pd.DataFrame):

    dir_df = get_dir_df(df)

    passed_tests = (float(dir_df['tests']) - (float(dir_df['test_errors'])
               float(dir_df['tests'])

    return passed_tests
```

Fast test builds

In [47]:
```
def m5(df: pd.DataFrame):
    dir_df = get_uts_df(df)

    density_fast_test_builds = len(dir_df[(dir_df['test_execution_time'].as
                               len(dir_df['test_execution_time'].astype(flo
    return density_fast_test_builds
```

Test coverage

In [48]:
```
def m6(df: pd.DataFrame):

    files_df = get_files_df(df)

    density_test_coverage = len(files_df[(files_df['coverage'].astype(float
    return density_test_coverage
```

Calculate measures for each repository

In [49]:

```
def create_metrics_df(df: pd.DataFrame) -> pd.DataFrame:

    date_time_vec = df['datetime'].unique()

    m1_list = []
    m2_list = []
    m3_list = []
    m4_list = []
    m5_list = []
    m6_list = []

    ncloc_list = []
    repository_list = []
    version_list = []

    for version in date_time_vec:

        version_df = df[df['datetime'] == version]

        try:
            m1_list.append(m1(version_df))
        except Exception:
            m1_list.append(0)

        try:
            m2_list.append(m2(version_df))
        except Exception:
            m2_list.append(0)

        try:
            m3_list.append(m3(version_df))
        except Exception:
            m3_list.append(0)

        try:
            m4_list.append(m4(version_df))
        except Exception:
            m4_list.append(0)

        try:
            m5_list.append(m5(version_df))
        except Exception:
            m5_list.append(0)

        try:
            m6_list.append(m6(version_df))
        except Exception:
            m6_list.append(0)

        ncloc_list.append(_ncloc(version_df))
        repository_list.append(version_df['repository'].iloc[0])
        version_list.append(version)

    final_dict = {
        'm1': m1_list,
        'm2': m2_list,
        'm3': m3_list,
        'm4': m4_list,
        'm5': m5_list,
        'm6': m6_list,
        'repository': repository_list,
        'version': version_list,
        'ncloc': ncloc_list
    }

    metrics_df = pd.DataFrame(final_dict)

    return metrics_df
```

In [50]:

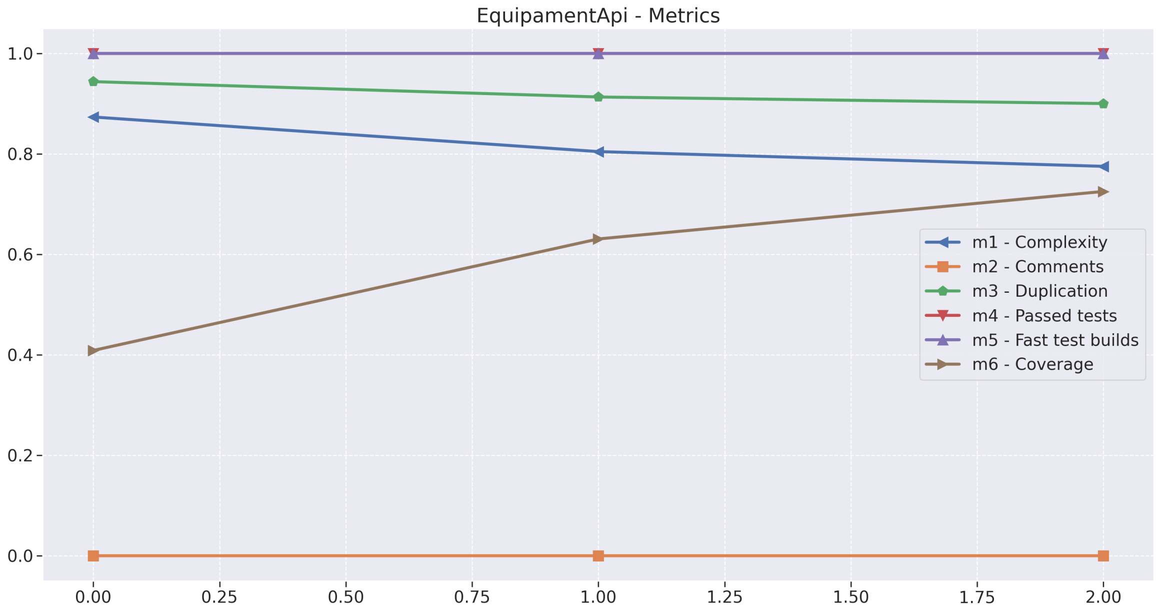
```
# Here we will create a dictionary with the metrics for each repository
metrics = {}

for repo_df in repos_dataframes:
    metrics[repo_df['name']] = create_metrics_df(repo_df['df'])
```

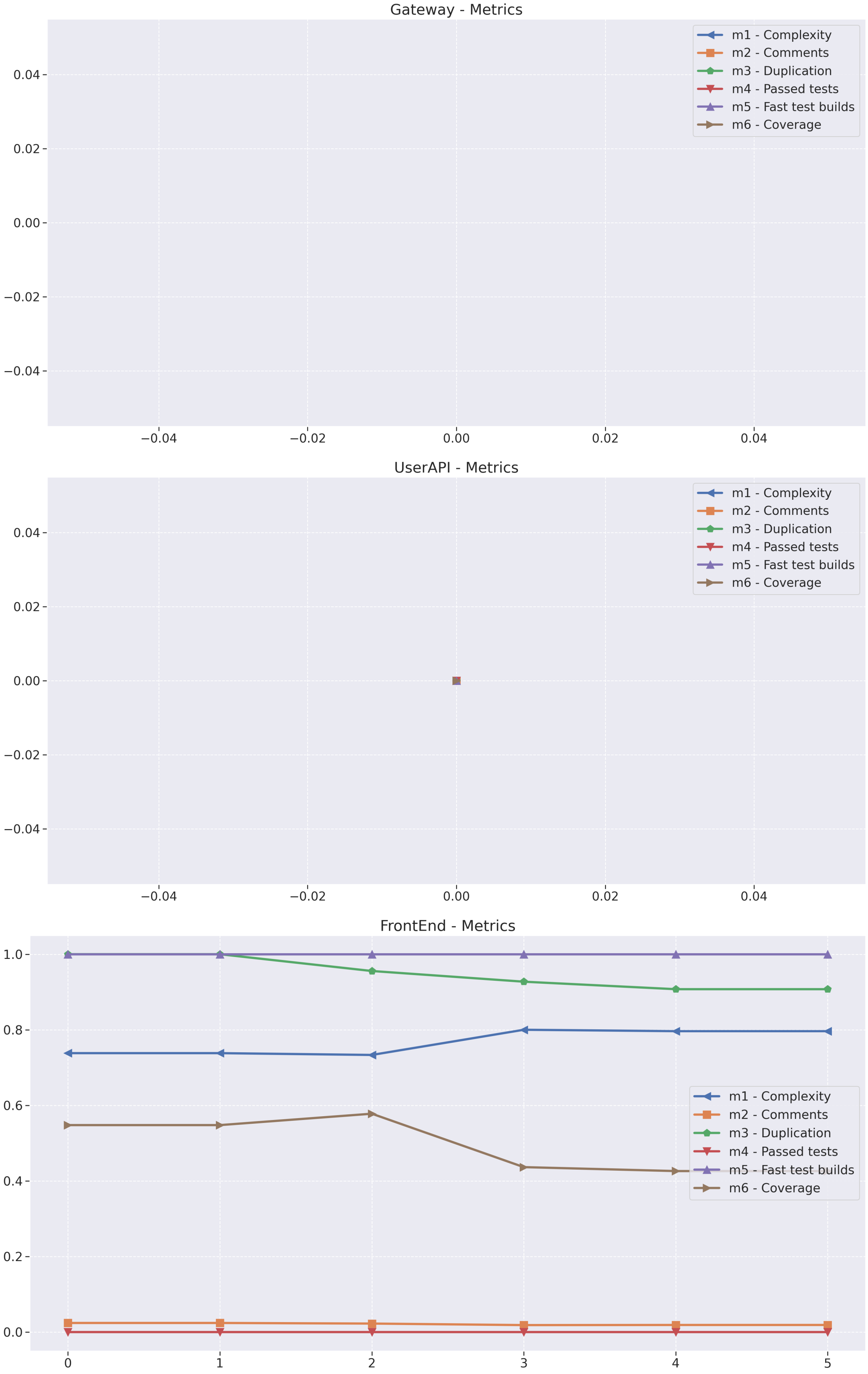
Data visualization

In this area you will need to plot the metrics of each repository.

In [51]:

```
for name, data in metrics.items():
    fig = plt.figure(figsize=(20, 10))

    plt.plot(data['m1'], linewidth=3, marker='<', markersize=10, label="m1
    plt.plot(data['m2'], linewidth=3, marker='s', markersize=10, label="m2
    plt.plot(data['m3'], linewidth=3, marker='p', markersize=10, label="m3
    plt.plot(data['m4'], linewidth=3, marker='v', markersize=10, label="m4
    plt.plot(data['m5'], linewidth=3, marker='^', markersize=10, label="m5
    plt.plot(data['m6'], linewidth=3, marker='>', markersize=10, label="m6

    plt.title(f"{name} - Metrics", fontsize=20)
    plt.legend(loc='best')
    plt.show()
```





Gateway - Metrics

UserAPI - Metrics

FrontEnd - Metrics

Quality factor and aspect aggregation

In [52]:

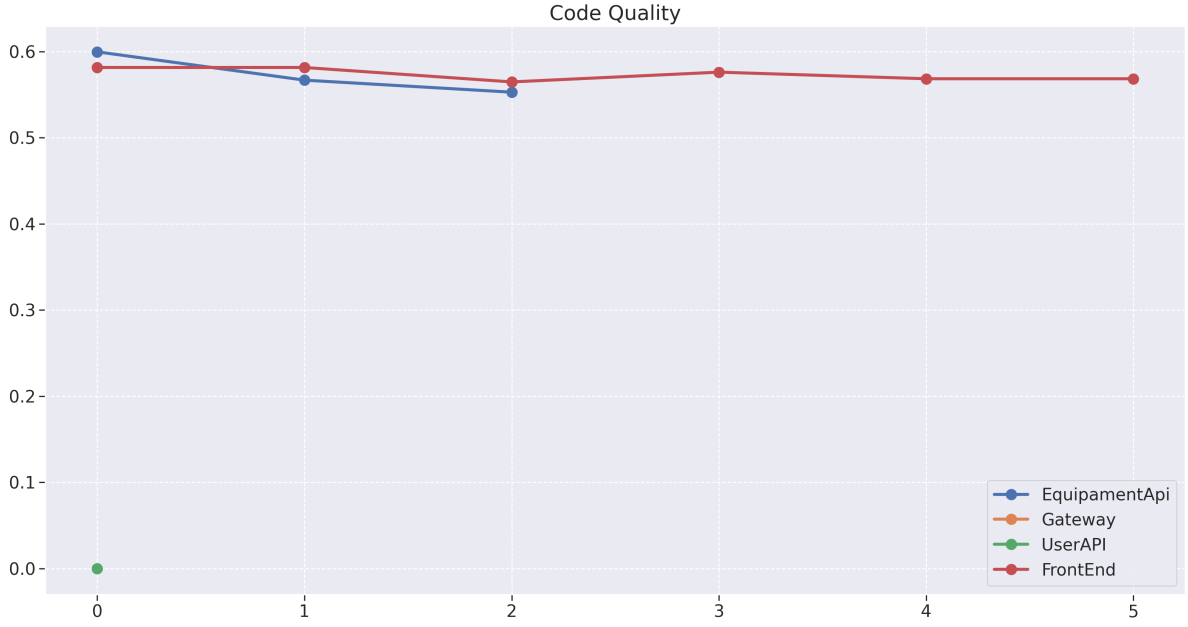
```
psc1 = 1
psc2 = 1
pc1 = 0.5
```

```
pc2 = 0.5
pm1 = 0.33
pm2 = 0.33
pm3 = 0.33
pm4 = 0.25
pm5 = 0.25
pm6 = 0.5


# Here you will need to create the code_quality and testing_status metrics

for name, data in metrics.items():
    data['code_quality'] = ((data['m1']*pm1) + (data['m2']*pm2) + (data['m3
    data['testing_status'] = ((data['m4']*pm4) + (data['m5']*pm5) + (data['r
```

Code Quality visualization

In [53]:

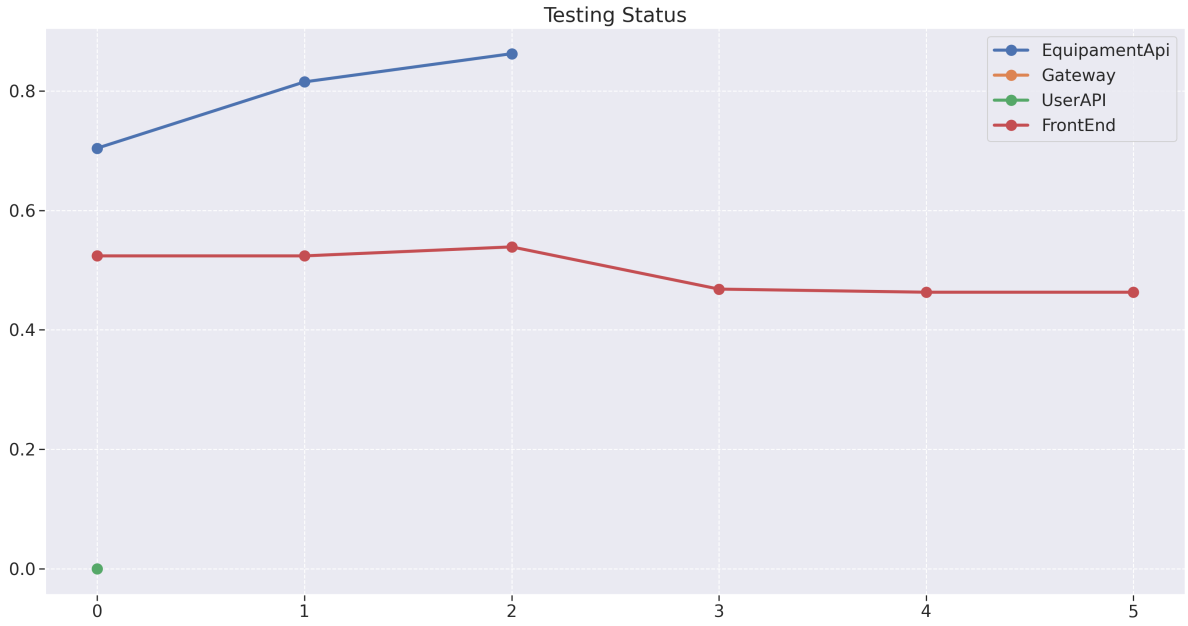
```
fig = plt.figure(figsize=(20, 10))

for name, data in metrics.items():
    plt.plot(data['code_quality'], linewidth=3, marker='o', markersize=10, l

plt.title("Code Quality", fontsize=20)
plt.legend(loc='best')
plt.show()
```



Testing Status visualization

In [54]:

```
fig = plt.figure(figsize=(20, 10))

for name, data in metrics.items():
    plt.plot(data['testing_status'], linewidth=3, marker='o', markersize=10

plt.title("Testing Status", fontsize=20)
plt.legend(loc='best')
plt.show()
```



Aggregations

```
In [55]: for name, data in metrics.items():
             data['Maintainability'] = data['code_quality'] * pc1
             data['Reliability'] = data['testing_status'] * pc2
             data['total'] = data['Maintainability'] + data['Reliability']
```

Repositories analysis

```
In [56]: def get_characteristc_stats(repo_series):
             return {
                 'mean': repo_series.mean(),
                 'mode': repo_series.mode(),
                 'median': repo_series.median(),
                 'std': repo_series.std(),
                 'var': repo_series.var(),
                 'min': repo_series.min(),
                 'max': repo_series.max()
             }
```

```
In [57]: def analysis(metrics, name):
             maintainability_stats = pd.DataFrame(get_characteristc_stats(metrics["Ma
                                                  columns=['mean', 'mode', 'median', 'st

             reliability_stats = pd.DataFrame(get_characteristc_stats(metrics["Relia
                                                  columns=['mean', 'mode', 'median', 'std',


             print("Maintainability Stats")
             print(maintainability_stats.to_string(index=False))

             print("Reliability Stats")
             print(reliability_stats.to_string(index=False))

             fig = plt.figure(figsize=(20, 10))

             plt.plot(metrics['Maintainability'], linewidth=3, marker='o', markersize
             plt.plot(metrics['Reliability'], linewidth=3, marker='*', markersize=10

             plt.ylim(0.1,1.1)
```

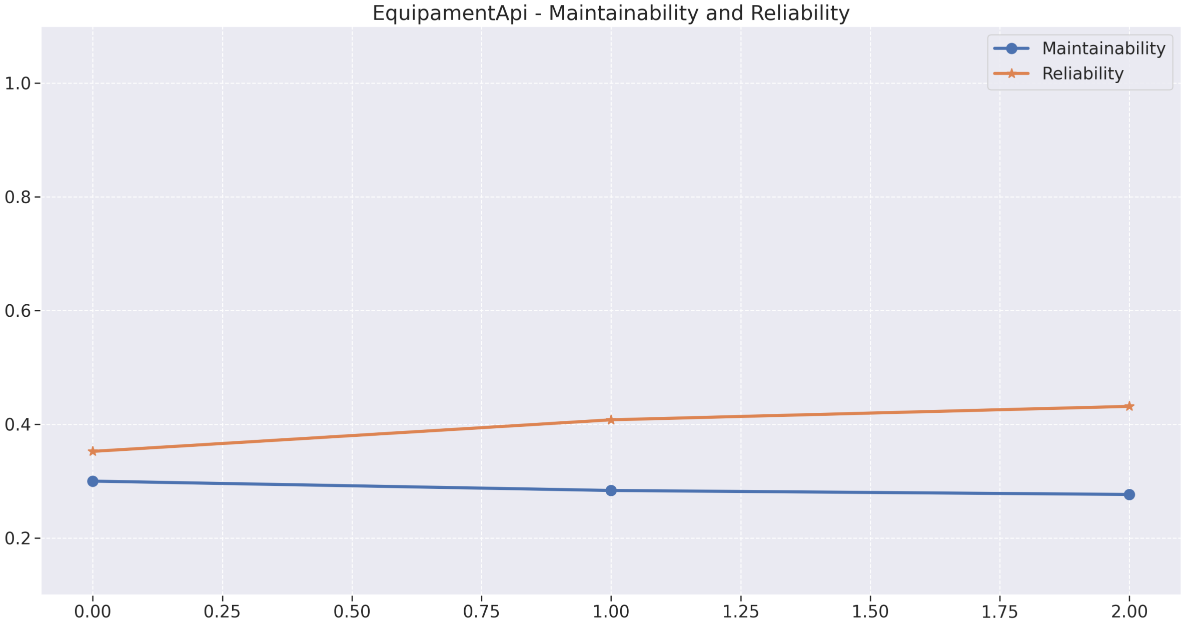
```
    plt.title(f'{name} - Maintainability and Reliability', fontsize=20)
    plt.legend(loc='best')
    plt.show()

    fig = plt.figure(figsize=(20, 10))

    plt.plot(metrics['total'], linewidth=3, marker='X', markersize=5)

    plt.ylim(0.1,1.1)
    plt.title(f'{name} - Total', fontsize=20)
    plt.show()
```

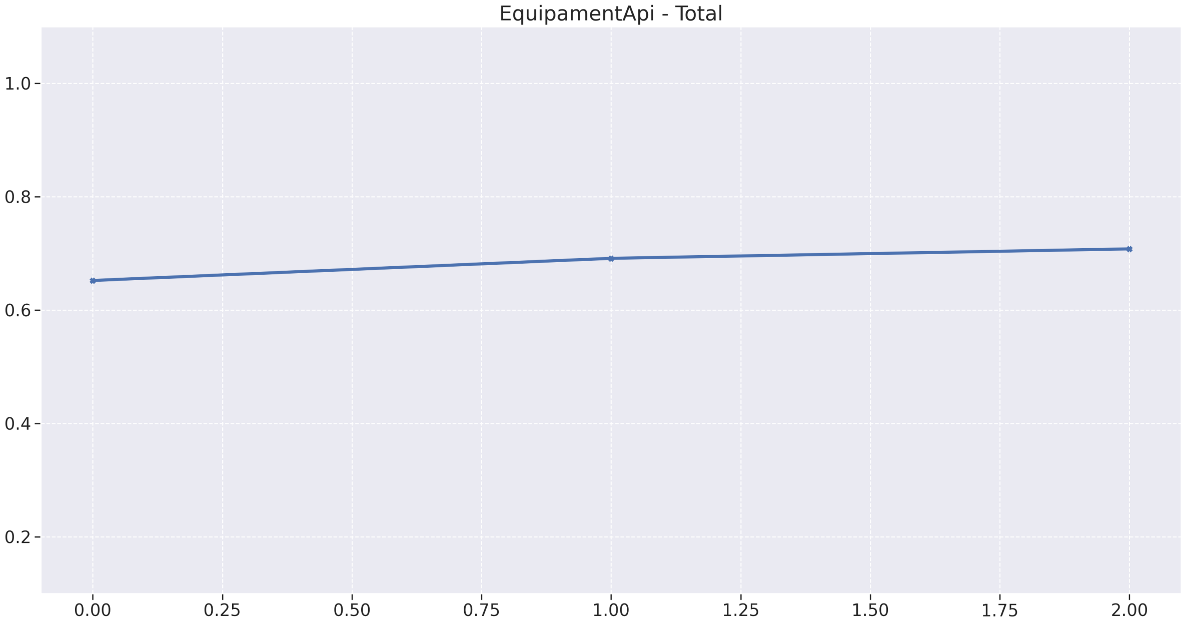
Analysis loop in each repo

In [58]:

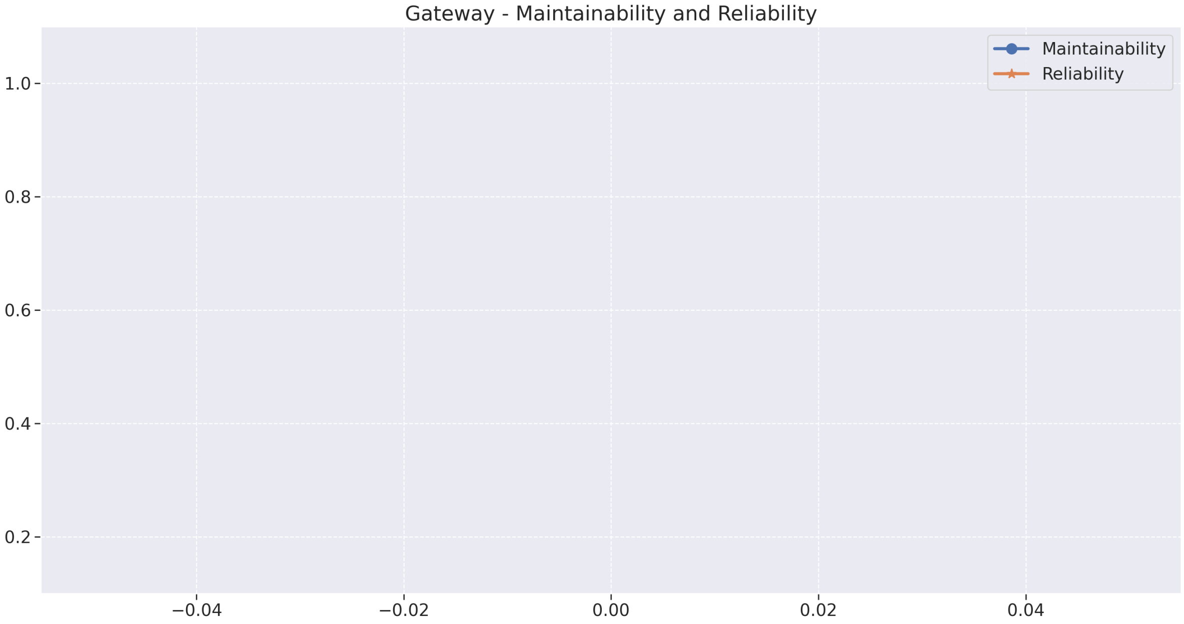
```
for name, data in metrics.items():
    print(name)
    analysis(data, name)
```

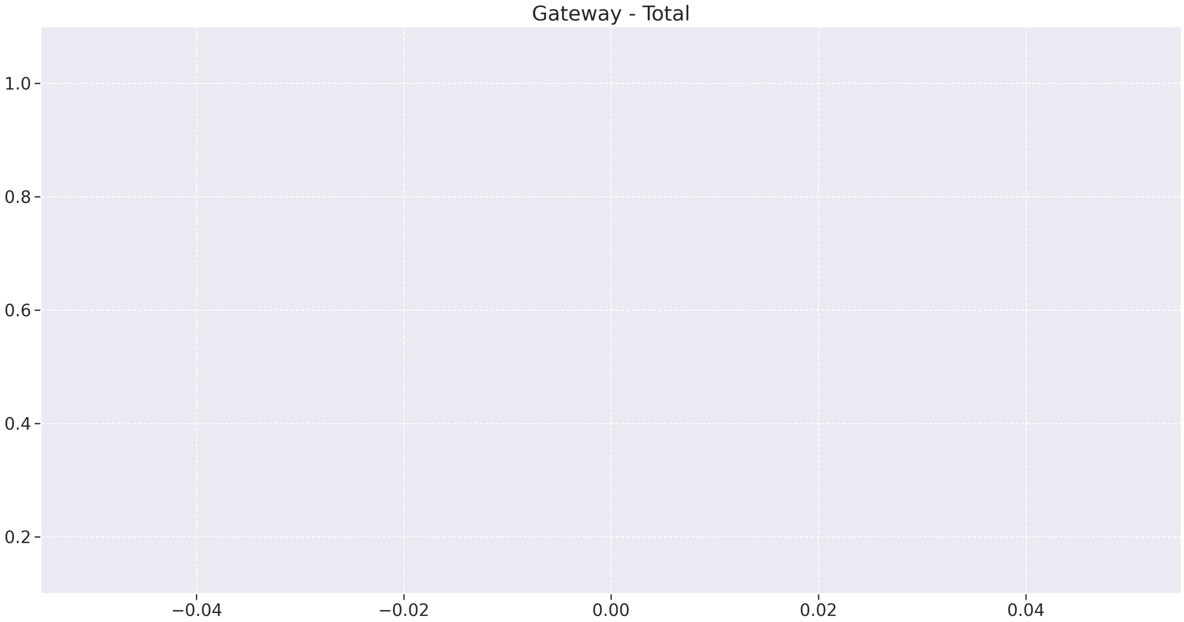
```
EquipamentApi
Maintainability Stats
    mean     mode  median      std      var      min      max
0.286511 0.276375 0.28337 0.012019 0.000144 0.276375 0.299789
0.286511 0.283370 0.28337 0.012019 0.000144 0.276375 0.299789
0.286511 0.299789 0.28337 0.012019 0.000144 0.276375 0.299789
Reliability Stats
   mean     mode   median      std     var      min     max
0.39699 0.352113 0.407609 0.040623 0.00165 0.352113 0.43125
0.39699 0.407609 0.407609 0.040623 0.00165 0.352113 0.43125
0.39699 0.431250 0.407609 0.040623 0.00165 0.352113 0.43125
```


EquipamentApi - Maintainability and Reliability

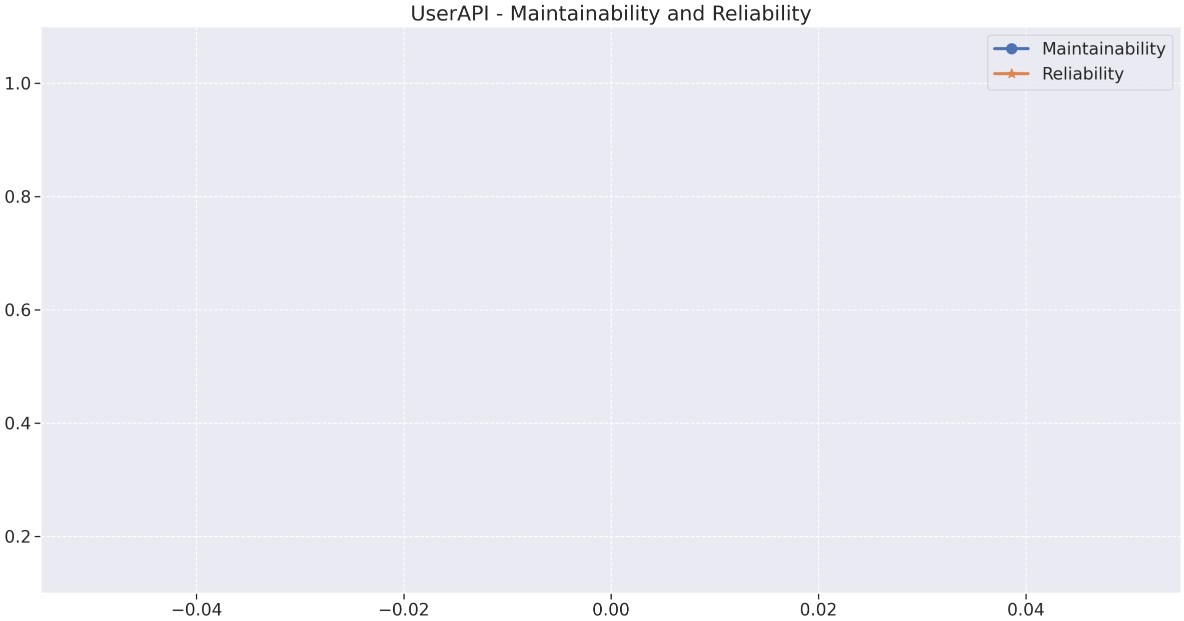


```
Gateway
Maintainability Stats
Empty DataFrame
Columns: [mean, mode, median, std, var, min, max]
Index: []
Reliability Stats
Empty DataFrame
Columns: [mean, mode, median, std, var, min, max]
Index: []
```

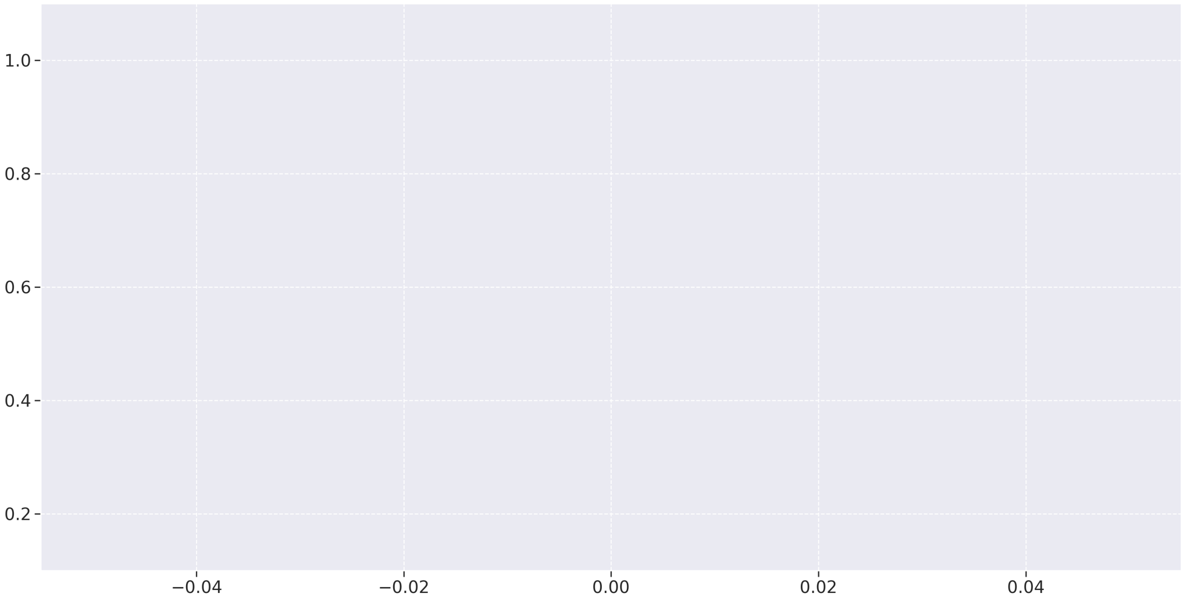




```
UserAPI
Maintainability Stats
 mean  mode  median std var  min  max
  0.0   0.0     0.0 NaN NaN  0.0  0.0
Reliability Stats
 mean  mode  median std var  min  max
  0.0   0.0     0.0 NaN NaN  0.0  0.0
```

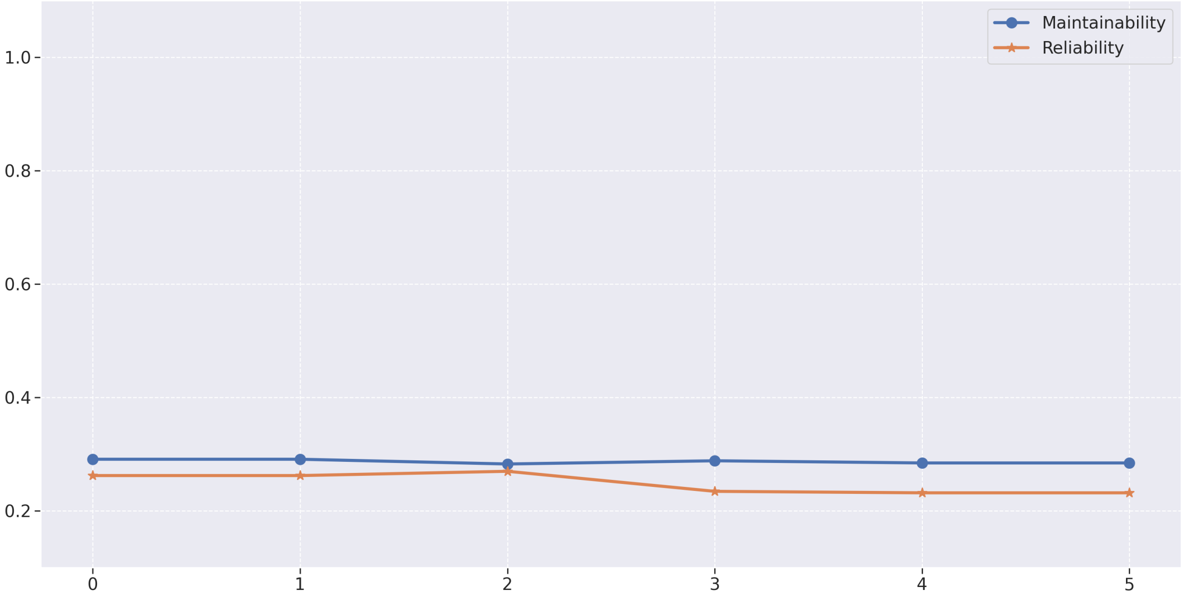


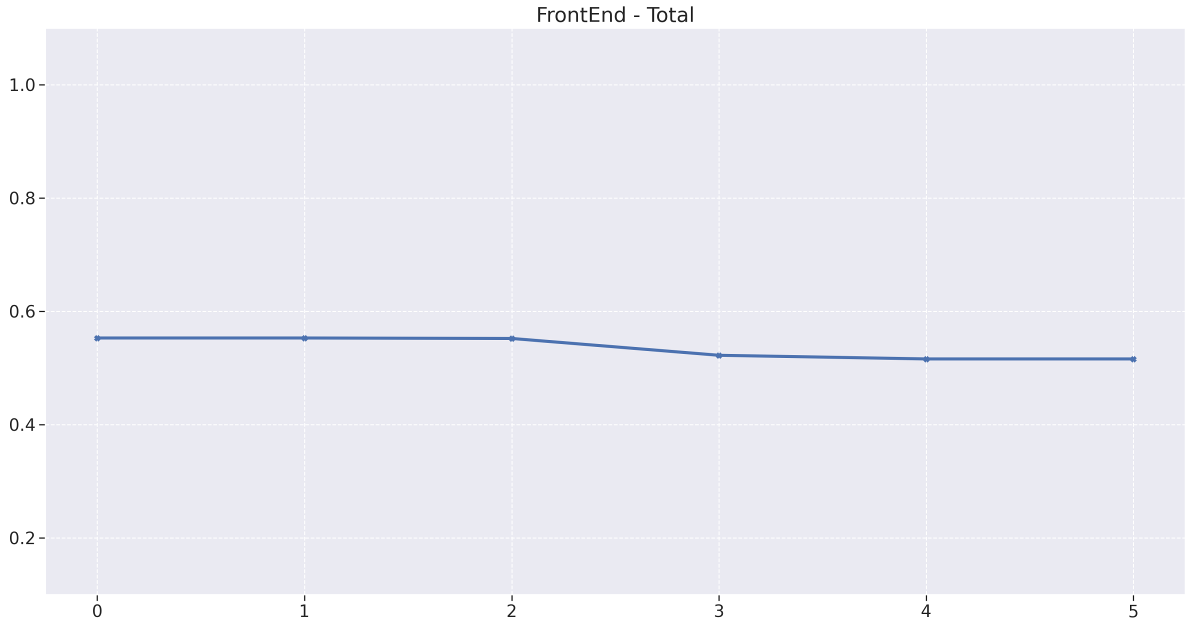
UserAPI - Total



```
FrontEnd
Maintainability Stats
    mean     mode   median      std      var      min      max
0.286683 0.284167 0.286083 0.003628 0.000013 0.282333 0.290714
0.286683 0.290714 0.286083 0.003628 0.000013 0.282333 0.290714
Reliability Stats
    mean     mode   median      std      var      min      max
0.248385 0.231481 0.247998 0.017804 0.000317 0.231481 0.269444
0.248385 0.261905 0.247998 0.017804 0.000317 0.231481 0.269444
```

FrontEnd - Maintainability and Reliability

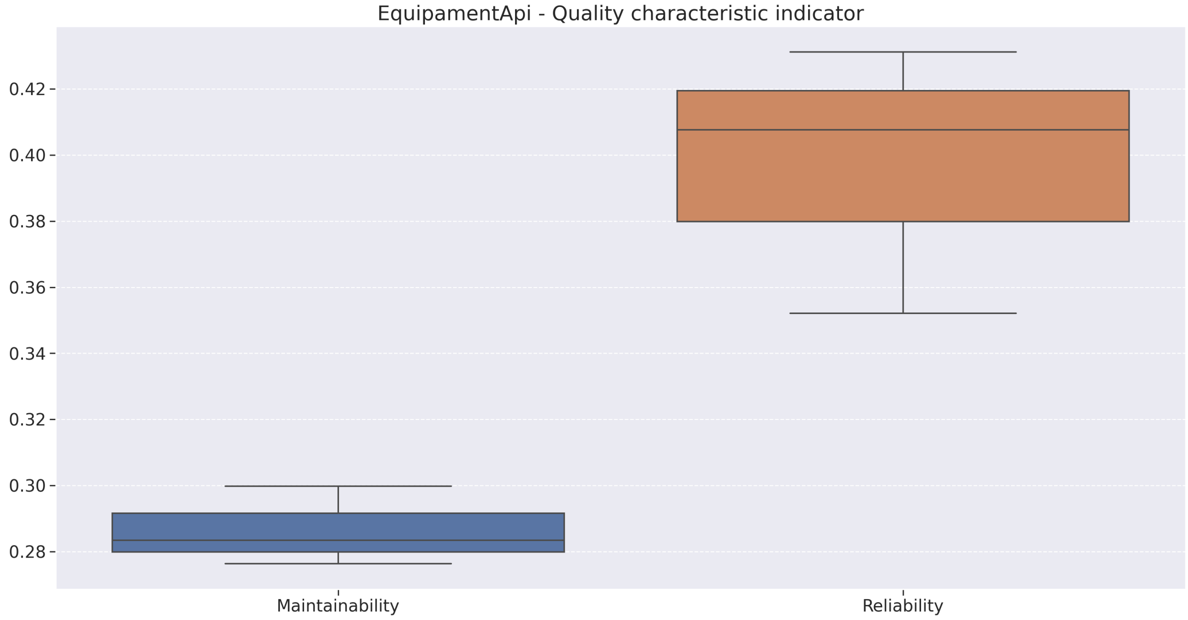


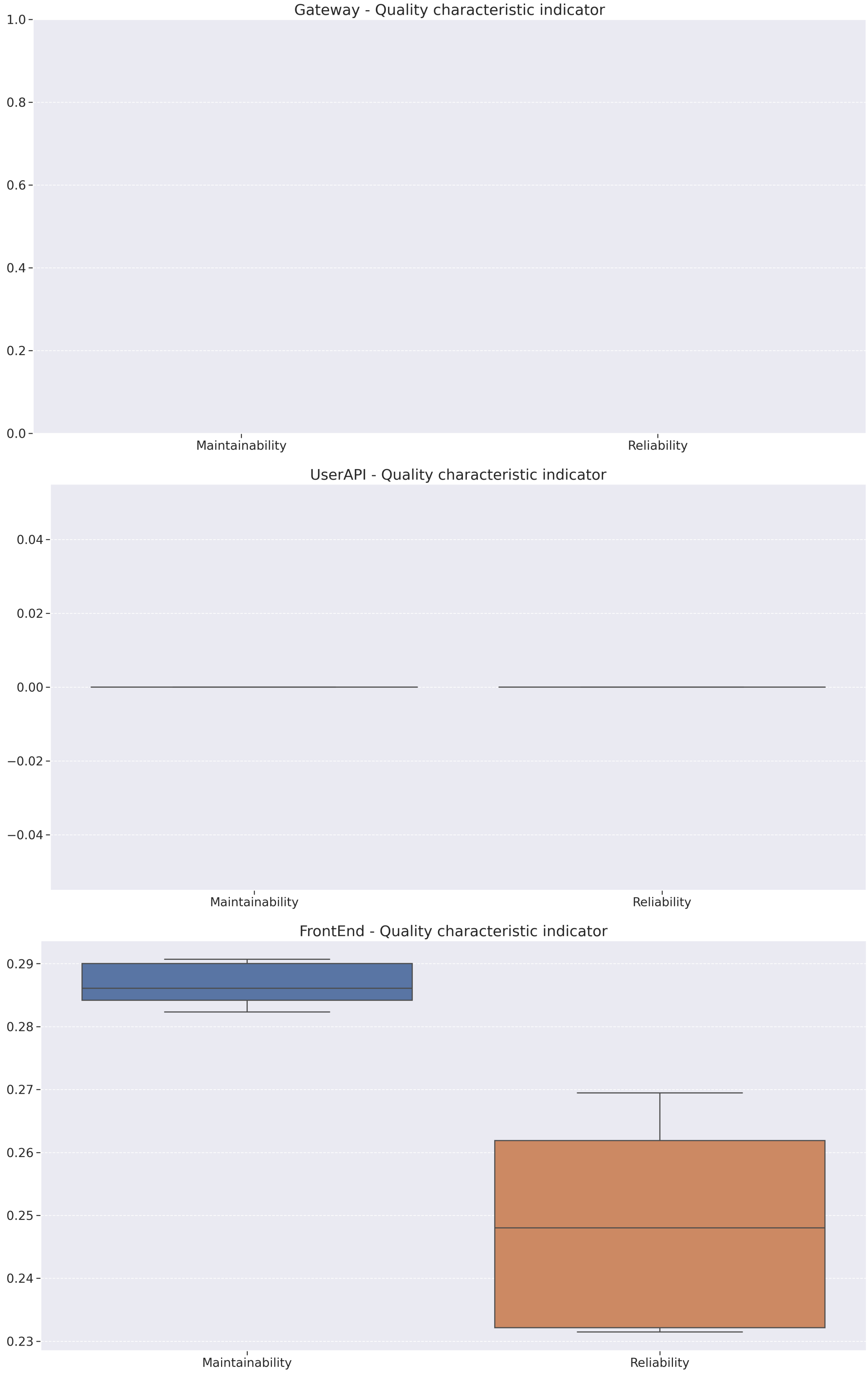


Quality characteristic indicator

In [59]:

```python
for name, data in metrics.items():
    fig = plt.figure(figsize=(20, 10))
    sns.boxplot(data=data[['Maintainability','Reliability']])

    plt.title(f"{name} - Quality characteristic indicator", fontsize=20)
    plt.show()
```



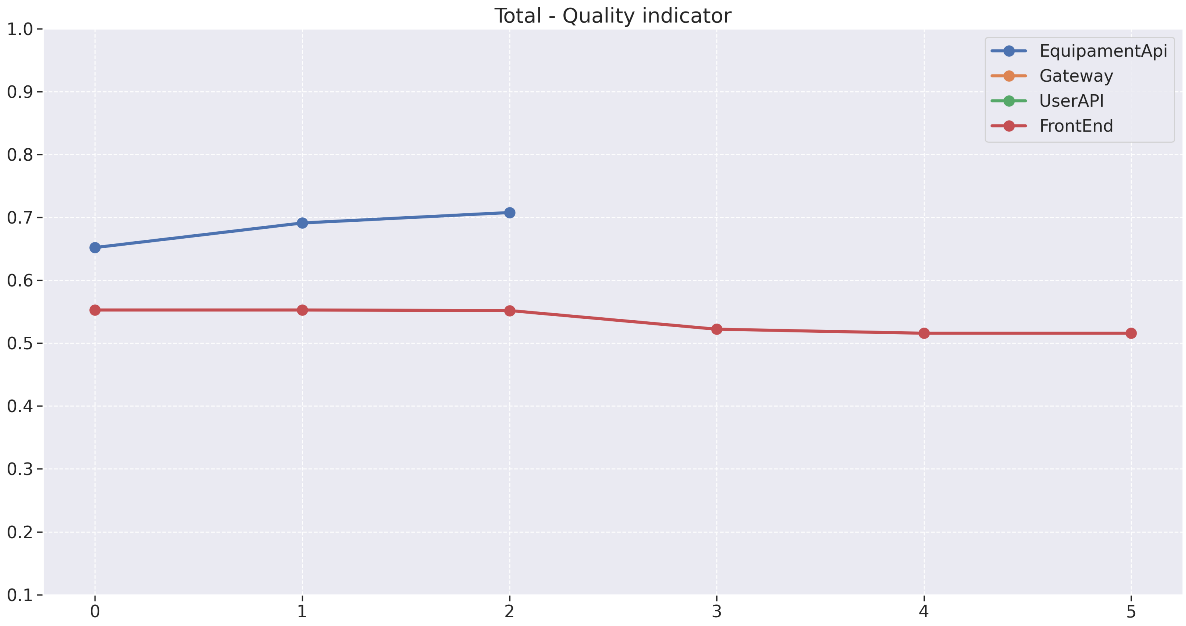


Quality indicator visualization

In [60]:

```
fig = plt.figure(figsize=(20, 10))
```

```
for name, data in metrics.items():
    plt.plot(data['total'], linewidth=3, marker='o', markersize=10, label=na

plt.ylim(.1,1)
plt.title("Total - Quality indicator", fontsize=20)
plt.legend(loc='best')
plt.show()
```



Export data

In [64]:

```
metrics_list = metrics.values()

metrics_df = pd.concat(metrics_list, ignore_index=True)

display(metrics_df)

current_datetime = datetime.datetime.now().strftime("%m-%d-%Y--%H-%M-%S")

metrics_df.to_excel('./data/fga-eps-mds-2023-1-Alectrion--{}.xlsx'.format(cu

metrics_df.to_csv('./data/fga-eps-mds-2023-1-Alectrion--{}.csv'.format(curre
```

	m1	m2	m3	m4	m5	m6	repository	version	ncloc	code_qua
0	0.873239	0.000000	0.943662	1.0	1.0	0.408451	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-13-2023-17-01-24	12409.0	0.599
1	0.804348	0.000000	0.913043	1.0	1.0	0.630435	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-13-2023-18-10-02	12409.0	0.566
2	0.775000	0.000000	0.900000	1.0	1.0	0.725000	fga-eps-mds-2023-1-Alectrion-EquipamentApi	05-17-2023-22-22-51	12409.0	0.552
3	0.000000	0.000000	0.000000	0.0	0.0	0.000000	fga-eps-mds-2023-1-Alectrion-UserAPI	05-12-2023-19-44-56	6296.0	0.000
4	0.738095	0.023810	1.000000	0.0	1.0	0.547619	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-15-2023-19-33-08	9157.0	0.581
5	0.738095	0.023810	1.000000	0.0	1.0	0.547619	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-17-2023-21-52-59	9157.0	0.581
6	0.733333	0.022222	0.955556	0.0	1.0	0.577778	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-17-2023-22-53-33	11714.0	0.564
7	0.800000	0.018182	0.927273	0.0	1.0	0.436364	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-19-2023-00-41-27	16848.0	0.576
8	0.796296	0.018519	0.907407	0.0	1.0	0.425926	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-19-2023-13-20-08	17516.0	0.568
9	0.796296	0.018519	0.907407	0.0	1.0	0.425926	fga-eps-mds-2023-1-Alectrion-FrontEnd	05-19-2023-23-11-30	17516.0	0.568

[R1] Análise geral da qualidade do Alectrion - 26/05/2023 (Sprint 04)

EquipamentAPI

- A duplicação de código no repositório EquipamentAPI está alta.
- A cobertura de código subiu desde o início do projeto até a Sprint 4, cerca de 25%, indo de 40% para 65%.
- A equipe reduziu a complexidade do código desde a Sprint 1 até a 4.
- A manutenibilidade permaneceu praticamente constante

No geral, os índices do repositório melhoraram, porém, ainda há necessidade de atuar em refatoração visando reduzir a duplicação.

UserAPI e Gateway

Os repositórios de Usuário e Gatway não foram analisados pois o foco do desenvolvimento foi no repositório de equipamentos.

FrontEnd

- A equipe percebeu que a duplicação de código no repositório FrontEnd está alta
- A cobertura de testes diminuiu
- A complexidade e manutenibilidade aumentaram

No geral, é necessário focar em testes e refatoração para redução de duplicação.

Consideração FINAL

A qualidade do código no repositório FrontEnd caiu em cerca de 5% e o EquipamentAPI subiu cerca de 5%, assim, a qualidade dos repositórios avaliados permaneceu praticamente constante, com poucas variações.

Assim, a partir desta análise, a equipe compreende a necessidade de atuar em testes e refatoração.