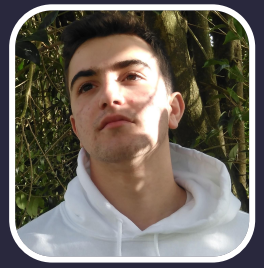
/FotoFaces

Licenciatura em Engenharia Informática
Projeto de Informática
Grupo 01

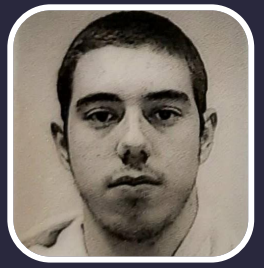
/Team


Vicente Costa


Pedro Lopes


Filipe Gonçalves


João Borges


Gonçalo Machado

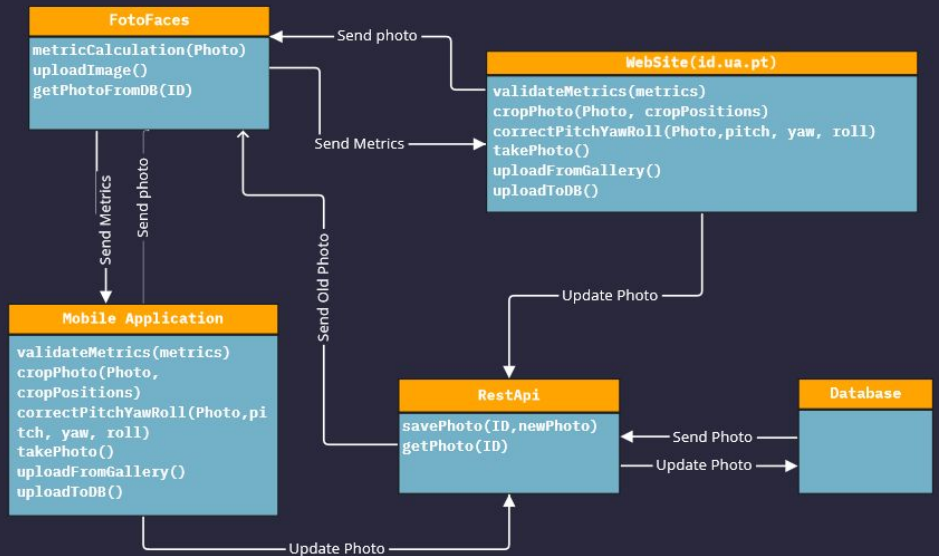
Advisors


António Neves

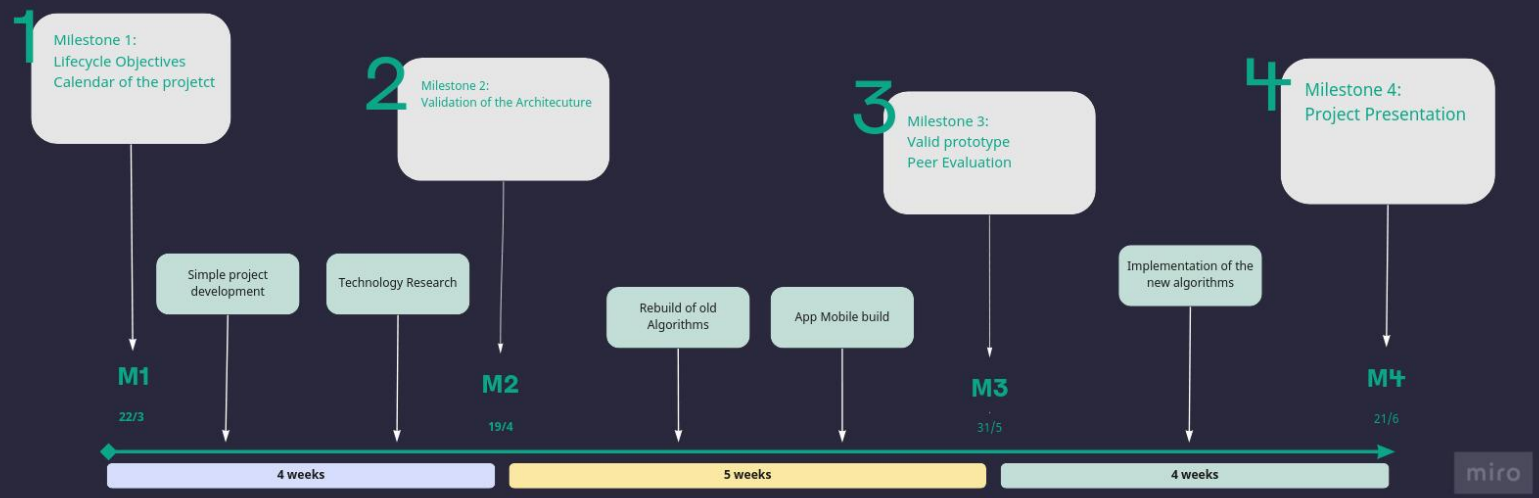

José Vieira


Daniel Canedo

/Project


FotoFaces
metricCalculation(Photo)
uploadImage()
getPhotoFromDB(ID)
Send photo
Send Metrics
WebSite(id.ua.pt)
validateMetrics(metrics)
cropPhoto(Photo, cropPositions)
correctPitchYawRoll(Photo,pitch, yaw, roll)
takePhoto()
uploadFromGallery()
uploadToDB()
Send Metrics
Send photo
Send Old Photo
Update Photo
Mobile Application
validateMetrics(metrics)
cropPhoto(Photo,
cropPositions)
correctPitchYawRoll(Photo,pi
tch, yaw, roll)
takePhoto()
uploadFromGallery()
uploadToDB()
RestApi
savePhoto(ID,newPhoto)
getPhoto(ID)
Database
Send Photo
Update Photo
Update Photo

/Project


1
Milestone 1:
Lifecycle Objectives
Calendar of the projetct
2
Milestone 2:
Validation of the Architecuture
3
Milestone 3:
Valid prototype
Peer Evaluation
4
Milestone 4:
Project Presentation
Simple project development
Technology Research
Rebuild of old Algorithms
App Mobile build
Implementation of the new algorithms
M1
22/3
M2
19/4
M3
31/5
M4
21/6
4 weeks
5 weeks
4 weeks
miro

/TABLE OF CONTENTS

/01 /Database

Database creation and simple calls

/02 /APIs

APIs and message dealer

/03 /Mobile_App

Work done in the mobile application

/04 /FotoFaces

Work done to the FotoFaces API

/01 ⟶ /Database

users	
id	int
email	varchar
full_name	varchar
password	varchar
photo	blob


```python
# mock database created with sqlite3
with sql.connect('mydb') as con:
    cur = con.cursor()
    cur.execute("""CREATE TABLE if not exists User (
        id int primary key,
        email text,
        full_name text,
        password varchar,
        photo blob);""")
    cur.close()
```

/01 ⟶ /Database

```python
def update_photo(identification, photo):

    # database call
    logging.debug("-- Begin -- Database call while updating old photo")
    try :
        with sql.connect('mydb') as con:
            cur = con.cursor()
            # UPDATE <Table of Users> SET <photo> = <inputed photo> WHERE <identification> = <inputed identification>
            cur.execute(f"UPDATE User SET photo = \'{photo}\' WHERE id = \'{identification}\'")
            con.commit()
            cur.close()
            logging.debug(f"Successful query")
    # in case we get an unexpected error
    except KeyError as k:
        logging.debug(f"Error: {k}")
        return json.dumps({"command": "upload_photo", "id": identification, "photo": photo, "error": k})

    logging.debug("-- End -- Database call while updating old photo")

    # return identification and photo
    return json.dumps({"command": "upload_photo", "status": "successful"})
```

Get old photo and update new photo

```python
def get_photo(identification):

    # database call
    logging.debug("-- Begin -- Database call while getting the old photo")
    try:
        with sql.connect('mydb') as con:
            cur = con.cursor()
            # SELECT * FROM <Table of Users> WHERE <identification> = <inputed identification>
            cur.execute(f"SELECT photo FROM User WHERE id = \'{identification}\'")
            result = cur.fetchall()
            cur.close()
            # in case we dont get any results
            if result == []:
                # debug
                logging.debug(f"{result}")
                return json.dumps({"command": "get_photo", "id": identification, "error": "result is empty"})
            logging.debug(f"Successful query")
    # in case we get an unexpected error
    except KeyError as k:
        logging.debug(f"Error: {k}")
        return json.dumps({"command": "get_photo", "id": identification, "error": k})

    logging.debug("-- End -- Database call while getting the old photo")

    # extract photo from the list of results
    # result len should be 1 and only 1
    photo = result[0][0]

    # return identification and photo
    return json.dumps({"command": "get_photo", "id": identification, "photo": photo})
```

/02 → /APIs

API with database calls

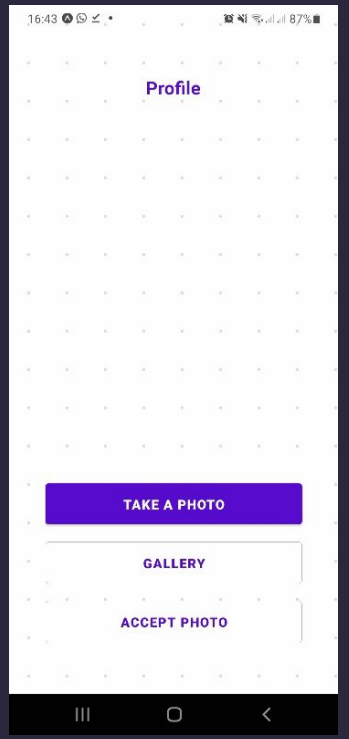
- update_Photo(ID, photo)
- get_Photo(ID)


```python
consumer = Consumer(config)

# Subscribe topic
consumer.subscribe([TOPIC_CONSUME], on_assign=reset_offset)

# Poll for new messages from Kafka and print them.
try:
    while True:
        msg = consumer.poll(1.0)
        if msg is None:
            pass
        elif msg.error():
            print(f"ERROR: {msg.error()}")
        else:
            msg_json = json.loads(msg.value().decode('utf-8'))
            print(f"Receiving message -> msg: {msg_json}")

            # print("Consumed event from topic {topic}: message = {message:12}".format(
            #     topic=msg.topic(), message=msg.value().decode('utf-8')))

            if msg_json["command"] == "update_photo":
                new_msg = update_photo(int(msg_json["id"]), msg_json["photo"])
            elif msg_json["command"] == "get_photo":
                new_msg = get_photo(int(msg_json["id"]))
            else:
                new_msg = {"error": "command not recognized"}

            # send new msg
            print(f"Send new message back")
            config_parser = ConfigParser()
            config_parser.read_file(args.config_file)
            config = dict(config_parser['default'])

            # Create Producer instance
            producer = Producer(config)
            producer.produce(TOPIC_PRODUCE, json.dumps(new_msg))
            producer.flush()
            producer.close()
```

/03 ⟶ /Mobile_App

/03.1 ⟶ /Take_Photo

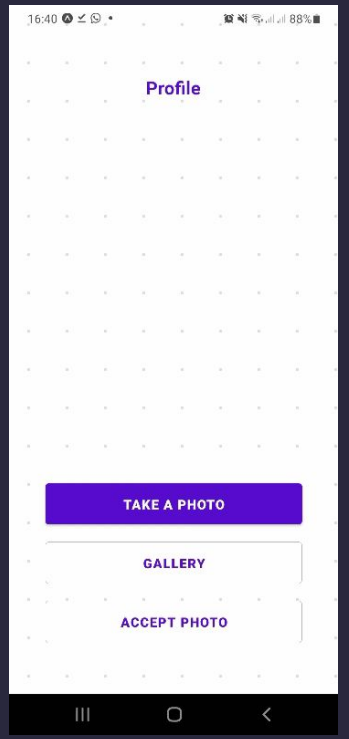
User chooses to take a picture and submit it for validation



/03 ⟶ /Mobile_App

/03.2 ⟶ /Get_Photo

User chooses to get a photo from his gallery
and submit it for validation



/03 ⟶ /Mobile_App

/03.3 ⟶ /API_calls

```
onPress={() =>  {
                const response = async () => {
                  await fetch('http://127.0.0.1:5000/', {
                    method: 'POST',
                    headers: {'Content-Type':'application/json', 'Access-Control-Allow-Origin': '*'},
                    body: JSON.stringify({
                      candidate: image ,
                      id:'10'
                    })
                  }).then((data) => {console.log(data)});
                }
              }
        }
```

/04 ⟶ /FotoFaces

/04.1 ⟶ /FotoFaces -> Plugins

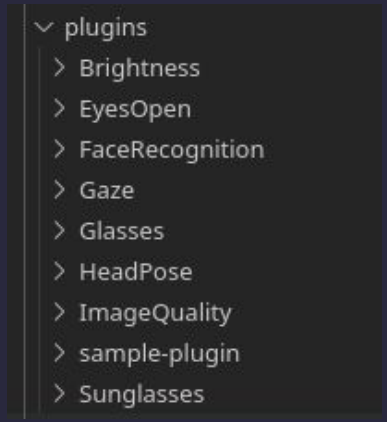
```
reference = cv2.imdecode(
    np.frombuffer(base64.b64decode(img2), np.uint8),
    cv2.IMREAD_COLOR,
)
resp = __init_app(

    candidate=candidate,
    reference=reference,
    raw_shape=raw_shape,
    image=image,
    shape=shape,
    final_img=final_img,
)
print(f"{resp}")
for k,v in resp.items():
    data[k] = v
__print_plugins_end()
return data
```

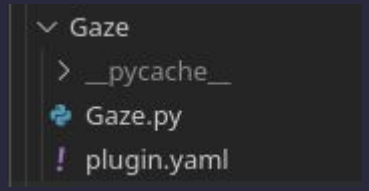
```
def __init_app(**args):
    return plEngine.start(**args)

plEngine = PluginEngine(options=
                        {"log_level": "DEBUG", "directory": "./plugins/"},
        )
```

Plugin Initialization

/04 → /FotoFaces

/04.2 → /FotoFaces -> Refactor



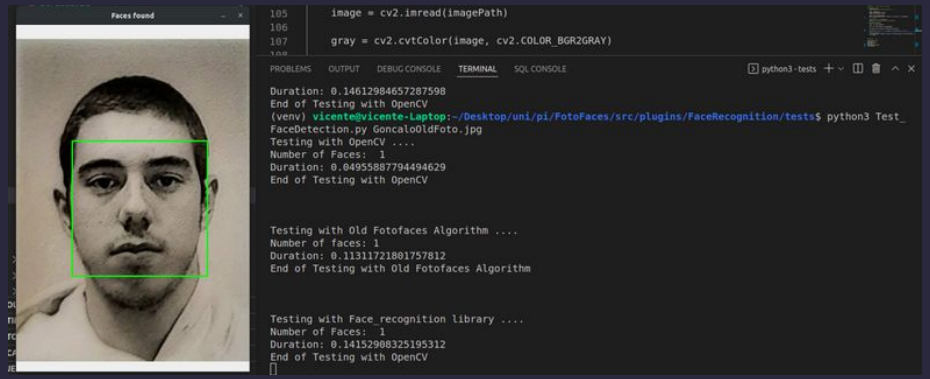


Algorithms refactorization to plugin architecture

```python
class GazePlugin(PluginCore):

    def __init__(self, logger: Logger) -> None:
        super().__init__(logger)
        self.meta = Meta(
            name='Gaze Plugin',
            description='Plugin for Gaze detection',
            version='0.0.1'
        )

    def dist_ratio(self, p1, p2, center):···


    def invoke(self, args):···
```

/04 ⟶ /FotoFaces

/04.3 ⟶ /FotoFaces -> Testing



Face Detection

/04 ⟶ /FotoFaces

/04.3 ⟶ /FotoFaces -> Testing

```
(venv) vicente@vicente-Laptop:~/Desktop/uni/pi/FotoFaces/src/plugins/FaceRecognition/tests$ python3 Test_
FaceComparison.py mini1.png mini2.png
Testing with Old Fotofaces Algorithm ....
Result:True
Duration: 0.7444138526916504
End of Testing with Old Fotofaces Algorithm



Testing with Face Recognition lib ....
Result:True
Duration: 1.3539936542510986
End of Testing with Face Recognition lib



Testing with fotofaces2.0 ....
Result:True
Duration: 0.7140209674835205
End of Testing with fotofaces2.0
(venv) vicente@vicente-Laptop:~/Desktop/uni/pi/FotoFaces/src/plugins/FaceRecognition/tests$
```

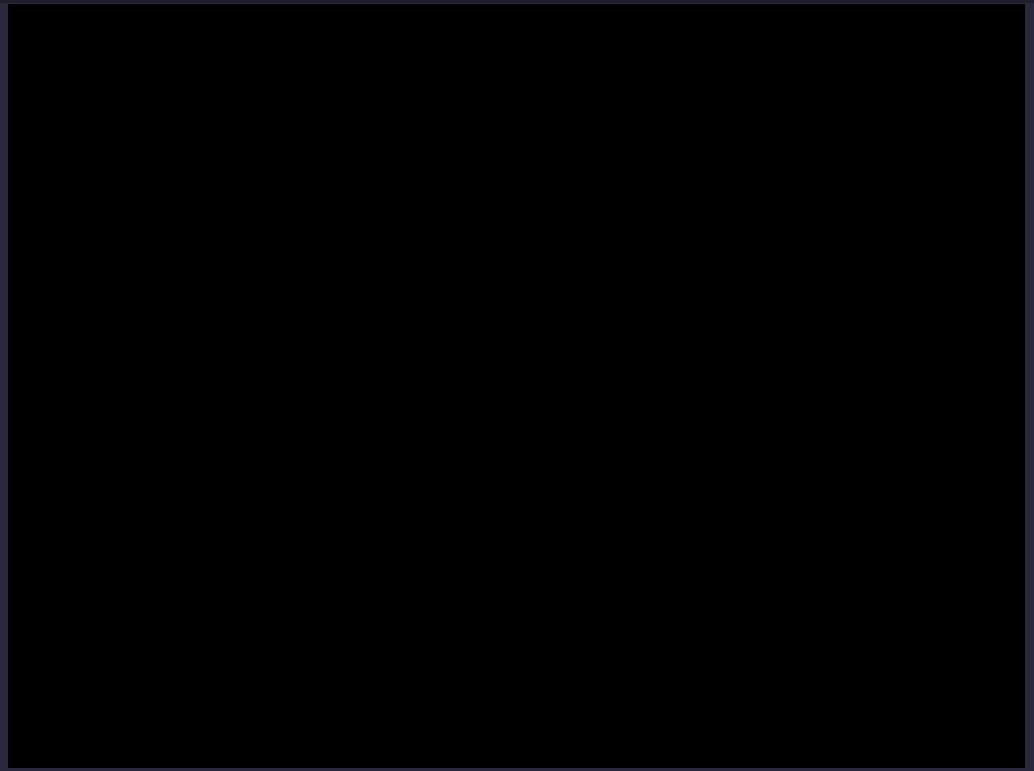
Same Person

```
(venv) vicente@vicente-Laptop:~/Desktop/uni/pi/FotoFaces/src/plugins/FaceRecognition/tests$ python3 Test_
FaceComparison.py mini1.png GoncaloOldFoto.jpg
Testing with Old Fotofaces Algorithm ....
Result:False
Duration: 0.5491247177124023
End of Testing with Old Fotofaces Algorithm



Testing with Face Recognition lib ....
Result:False
Duration: 0.9125103950500488
End of Testing with Face Recognition lib



Testing with fotofaces2.0 ....
Result:False
Duration: 0.5466806888580322
End of Testing with fotofaces2.0
(venv) vicente@vicente-Laptop:~/Desktop/uni/pi/FotoFaces/src/plugins/FaceRecognition/tests$
```

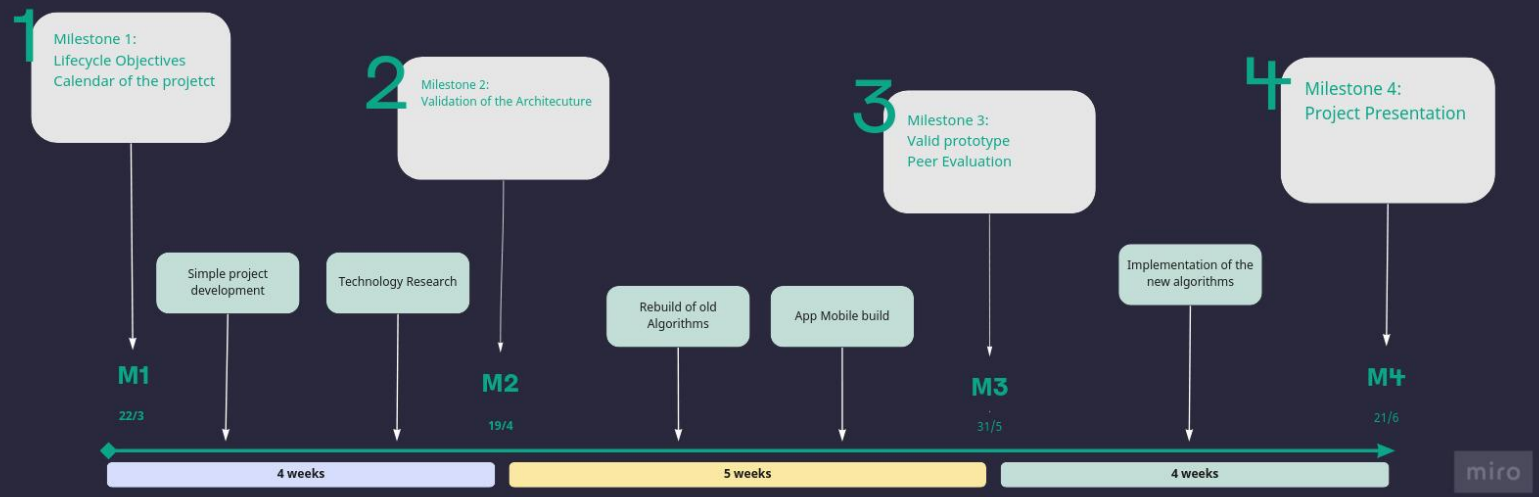
Different People

Face Comparison

DEMO



/05 → /Whats_Next



/05 ⟶ /Whats_Next

- Kafka scalability
- Implementation of new algorithms
- Testing
- Containerization