BACKEND EXERCISE + EXTRAS

Presented by **João Vasco**

Unbabel | 2024

OVERVIEW

Unbabel | 2024

OBJECTIVES

```
unbabel_cli --input_file events.json --window_size 10
```

- I'm building a CLI tool to calculate moving averages of translation delivery times from event streams, focusing on simplicity and efficiency.
- { "**timestamp**": "2018-12-26 18:12:19.903159",
  "translation_id": "5aa5b2f39f7254a75aa4",
  "source_language": "en",
  "target_language": "fr",
  "client_name": "airliberty",
  "event_name": "translation_delivered",
  "nr_words": 100,
  "**duration**": 20 }

- We want to count, for each minute, the moving average delivery time of all translations for the past 10 minutes

OBJECTIVES

- {"**timestamp**": "2018-12-26 18:11:08.509654","translation_id": "5aa5b2f39f7254a75aa5","source_language": "en","target_language": "fr","client_name": "airliberty","event_name": "translation_delivered","nr_words": 30, "**duration**": 20}

- {"**timestamp**": "2018-12-26 18:15:19.903159","translation_id": "5aa5b2f39f7254a75aa4","source_language": "en","target_language": "fr","client_name": "airliberty","event_name": "translation_delivered","nr_words": 30, "**duration**": 31}

- {"**timestamp**": "2018-12-26 18:23:19.903159","translation_id": "5aa5b2f39f7254a75bb3","source_language": "en","target_language": "fr","client_name": "taxi-eats","event_name": "translation_delivered","nr_words": 100, "**duration**": 54}

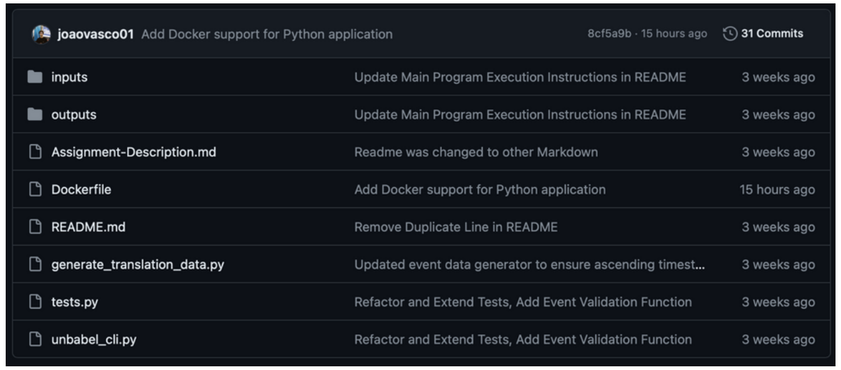
- **WINDOW SIZE IS 10**


```
{"date": "2018-12-26 18:11:00", "average_delivery_time": 0}
{"date": "2018-12-26 18:12:00", "average_delivery_time": 20}
{"date": "2018-12-26 18:13:00", "average_delivery_time": 20}
{"date": "2018-12-26 18:14:00", "average_delivery_time": 20}
{"date": "2018-12-26 18:15:00", "average_delivery_time": 20}
{"date": "2018-12-26 18:16:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:17:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:18:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:19:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:20:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:21:00", "average_delivery_time": 25.5}
{"date": "2018-12-26 18:22:00", "average_delivery_time": 31}
{"date": "2018-12-26 18:23:00", "average_delivery_time": 31}
{"date": "2018-12-26 18:24:00", "average_delivery_time": 42.5}
```

RESOLUTION

1

- Inputs
  - **Input_given.json**
  - **input_large.json**
    - **generate_translation_data.py**
- outputs
  - (same) given & large



2

- **Unbabel_cli.py**
  - Core of the program
- **tests.py**
  - provide the tests

RESOLUTION (RUN)

1

- generate_translation_data.py:

```
python3 generate_translation_data.py --input_file inputs/input_large.json
```

  - Generates 1000 inputs, adjustable on the code

2

- Unbabel_cli.py

```
python3 unbabel_cli.py --input_file inputs/input_large.json --window_size 10 --output_file outputs/output_large.json
```

- tests.py

```
python3 tests.py
```

RESOLUTION (PARSING)

1

- 2 Functions
  - Parse_input
  - Parse_arguments

```python
def main():
    """
    Main function to handle the workflow.
    """
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

    try:
        args = parse_arguments()
        window_size_minutes = args.window_size
        file_path = args.input_file
        output_file_path = args.output_file
```

2

- Parse arguments
  - Parses the command-line , creates an object that cotains them (args.input_file | args.output_file | args.window_size)
  - Called at the beginning of the main

RESOLUTION (PARSING)

1

- Parse_input:
  - Loads the json provided on the input file.
  - With various fields, including one timestamp which -> converted into datatime, for easier process and calculations

2

- It is called on the main function right after parse_arguments

```
try:
    args = parse_arguments()
    window_size_minutes = args.window_size
    file_path = args.input_file
    output_file_path = args.output_file


    events = parse_input(file_path)
```

```
def parse_input(file_path):
    """
    Parses the input file containing translation events.

    Each line in the file is expected to be a JSON object with various fields,
    including a timestamp, which is converted to a datetime object for easier
    processing in later stages.

    Parameters:
    file_path (str): The path to the input file.

    Returns:
    list[dict]: A list of dictionaries, each representing a translation event.
    """
    translations = []

    try:
        with open(file_path, 'r') as input_file:
            for line in input_file:
                # Parsing the JSON line and converting timestamp
                translation = json.loads(line.strip())
                translation['timestamp'] = datetime.strptime(
                    translation['timestamp'], "%Y-%m-%d %H:%M:%S.%f")
                translations.append(translation)

    except FileNotFoundError:
        print(f"Error: File not found - {file_path}")
    except json.JSONDecodeError:
        print("Error: Invalid JSON format in file")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

    return translations
```

RESOLUTION (CLASS)

1

- Structure
  - Class (EventWindow)
    - Adding Event
    - Removing Expired Event
    - Calculate Moving Average

2

- EventWindow:
  - Initialized with the window size
  - **Is a deque**
  - Each event added
    - Duration
    - Expiration_timestamp = timestamp of input + windowsize

```python
class EventWindow:
    def __init__(self, window_size_minutes):
        """
        Initializes the EventWindow with a specified window size.

        Args:
            window_size_minutes (int): Size of the time window in minutes for calculating moving average.
        """
        self.events = deque()
        self.window_size = timedelta(minutes=window_size_minutes)

    def add_event(self, event_timestamp, duration):
        """
        Adds an event to the window.

        Args:
            event_timestamp (datetime): Timestamp of the event.
            duration (float): Duration of the event.
        """
        expiration_time = event_timestamp + self.window_size
        self.events.append((duration, expiration_time))

    def remove_expired_events(self, current_time):
        """
        Removes events from the window that are older than the window size.

        Args:
            current_time (datetime): The current time to compare for expiration.
        """
        while self.events and self.events[0][1] < current_time:
            self.events.popleft()

    def calculate_moving_average(self):
        """
        Calculates the moving average of the durations of the events in the window.

        Returns:
            float: The moving average of the event durations.
        """
        if not self.events:
            return 0
        total_duration = sum(duration for duration, _ in self.events)
        return total_duration / len(self.events)
```

RESOLUTION (MAIN)

1

- While Loop :
  - From the first input timestamp (rounded down)
  - To the last input timestamp (rounded up)
  - Adds an event when :
    - There are still events & timestamp of event is shorter then current time

2

- In every loop checks if there are events to remove
  - Current time > Expiration time of event
- Calculates the moving average
- Writes in the Output

```python
def main():
        event_window = EventWindow(window_size_minutes)

        with open(output_file_path, 'w') as output_file:
            # Determine the start and end times for processing
            start_time = round_down_time(events[0]['timestamp'])
            end_time = round_up_time(events[-1]['timestamp'])


            current_time = start_time
            event_index = 0

            while current_time <= end_time:
                # Add new events that fall within the current minute
                while event_index < len(events) and events[event_index]['timestamp'] < current_time :
                    event_window.add_event(events[event_index]['timestamp'], events[event_index]['duration'])
                    event_index += 1

                # Remove expired events and calculate moving average
                event_window.remove_expired_events(current_time)
                moving_average = event_window.calculate_moving_average()

                # Print or store the result

                output = {"date": current_time.strftime("%Y-%m-%d %H:%M:%000"), "average_delivery_time": moving_average}
                output_file.write(json.dumps(output) + '\n')

                # Increment current time by one minute
                current_time += timedelta(minutes=1)
```

RESOLUTION (TESTS)

1

- Tests file
  - Error such as “File not found”
  - Invalid Json
  - Unexpected Error
  - No events in the file
  - Window_size <=0

2

- This cases are treated in the code in validate_and_parse_events and whenever it seems appropriate to include exceptions.
- Containing a detailed log

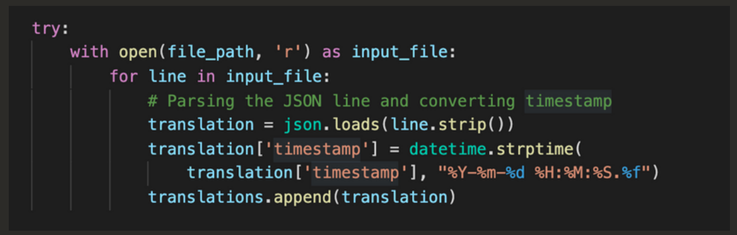
```python
def validate_and_parse_events(window_size_minutes, file_path):
    """
    Validates the window size and parses the events from the given file.

    Args:
        window_size_minutes (int): The size of the time window in minutes.
        file_path (str): The path to the input file containing event data.

    Returns:
        list[dict]: A list of event data dictionaries.

    Raises:
        ValueError: If the window size is not a positive integer.
    """
    if window_size_minutes <= 0:
        raise ValueError("Window size must be a positive integer.")

    events = parse_input(file_path)
    if not events:
        logging.error("No events found in the input file.")
        sys.exit(1)

    return events
```

FUTURE WORK

- Write the code in multiple files for more organization, not everything on unbabel_cli.py:
  - **event_window.py** - Contains the EventWindow class
  - **utils.py** - Manages parsing input files and command-line arguments, including validation functions.
  - **main.py** - The main script that ties everything together, using the functions and classes from the other modules.
- Loop of Main in other function

FUTURE WORK

- Deal with bigger Data:
  - Treat the dataset as smaller datasets. Alloc memory and free, to avoid problems on processing on the function parse_input
  - Use parallel programing, where I can treat the datasets with multiple threads



```
try:
    with open(file_path, 'r') as input_file:
        for line in input_file:
            # Parsing the JSON line and converting timestamp
            translation = json.loads(line.strip())
            translation['timestamp'] = datetime.strptime(
                translation['timestamp'], "%Y-%m-%d %H:%M:%S.%f")
            translations.append(translation)
```

- Update the method to only remove expired events when necessary, for example, right before calculating the moving average or adding a new event. Depends on the frequency of those events. “analyze the dataset”.

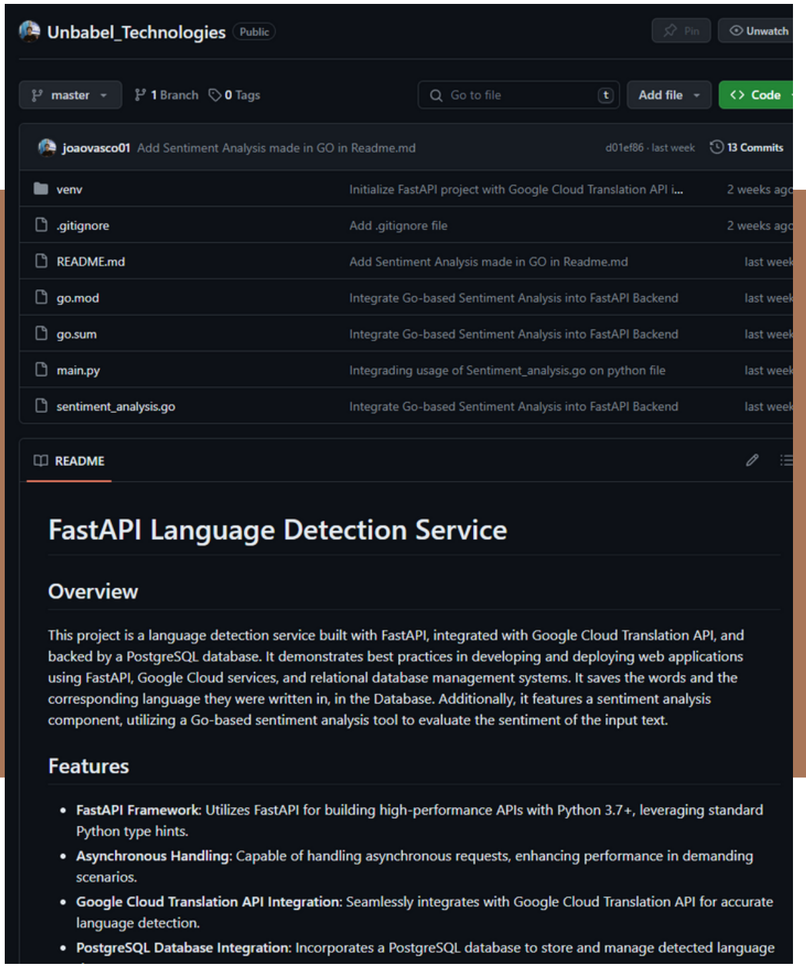
EXTRA

First Objective

- Detect the language that was written in a sentence
  - Using the GoogleTranslation API

Second Objective

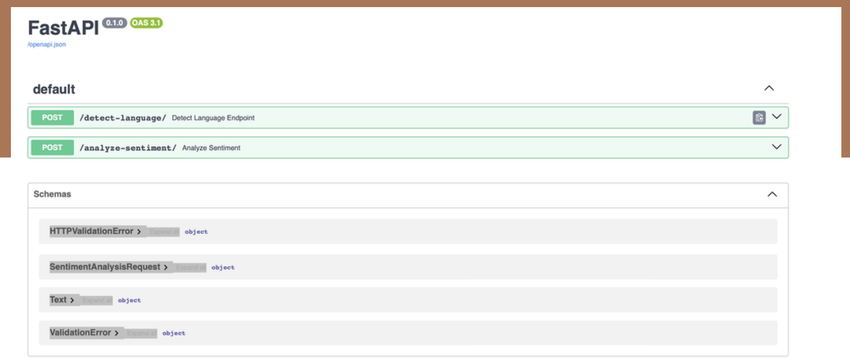
- Sentiment Analysis of that sentence
  - 1 in case it is a positive one “Love you”
  - 0 in other cases “Hate you”

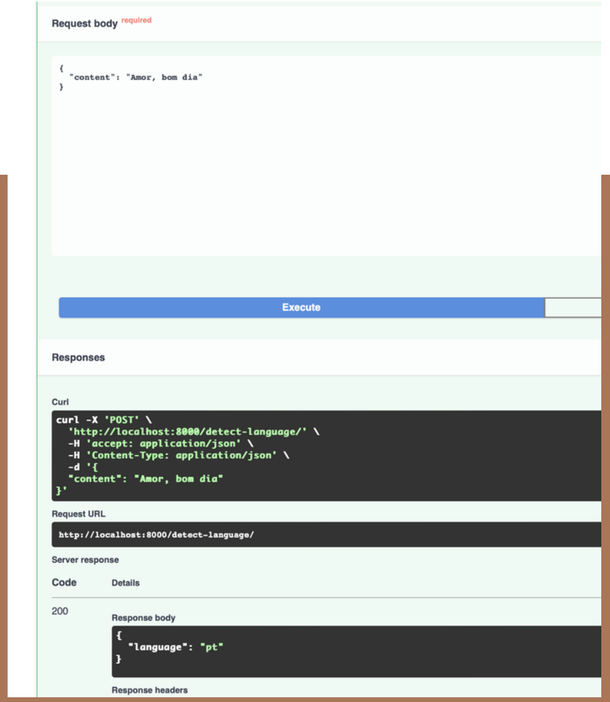


EXTRA 1ST

Fast API \ GOOGLE API \ POSTGRES

- Worked with Fast API
- Managed to work with different APIs
- Managed to make the Language Detection Work
- Stored the results of the language detected and the sentence in POSTGRESQL for eventual future training






```
mydatabase=> SELECT * FROM language_detection;
 id |  content   | detected_language
----+------------+-------------------
  1 | string     | en
  2 | amigo      | pt
  3 | unbabel    | en
  4 | buongiorno | it
  5 | portugal   | en
(5 rows)

mydatabase=>
```

```python
def detect_language(text: str) -> str:
    """
    Detects the language of the given text using Google Cloud Translation API.

    Args:
        text (str): Text whose language needs to be detected.

    Returns:
        Detected language code (e.g., 'en' for English).

    Raises:
        Exception: Any exceptions that occur during the API call.
    """
    try:
        credentials = service_account.Credentials.from_service_account_file(
            GOOGLE_CREDENTIALS, scopes=["https://www.googleapis.com/auth/cloud-platform"])
        client = translate.Client(credentials=credentials)
        result = client.detect_language(text)
        return result['language']
    except Exception as e:
        # Consider replacing print with logging for production use
        print(f"An error occurred in detect_language: {e}")
        raise

@app.post("/detect-language/")
def detect_language_endpoint(text: Text, db: Session = Depends(get_db)):
    """
    FastAPI endpoint to detect the language of provided text and store in database.

    Args:
        text (Text): Instance of Text model containing content to be analyzed.
        db (Session): Database session dependency injected by FastAPI.

    Returns:
        JSON response with the detected language.
    """
    if not text.content:
        raise HTTPException(status_code=400, detail="No content provided")

    detected_language = detect_language(text.content)

    # Create and add new record to the database
    language_record = LanguageDetection(content=text.content, detected_language=detected_language)
    db.add(language_record)
    db.commit()

    return {"language": detected_language}
```

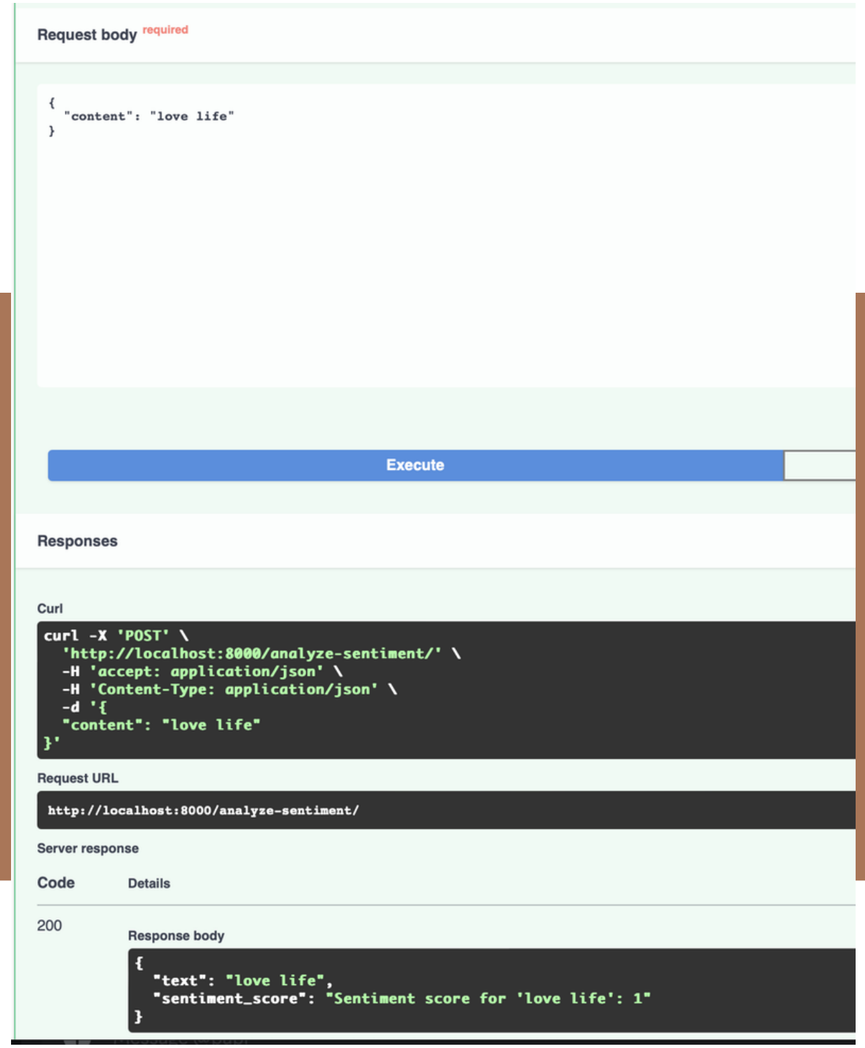
EXTRA 2ND & 3RD

GO

- Worked with GO since told it is one language used in Unbabel
- Project of my main interest Sentiment Analysis
- Explored the language

DOCKER

- Started to work with Docker. Created a Dockerfile for the Backend Challenge.



FUTURE WORK

- Create tests on the project to make it accessible to others.
- Make the Code work to not only detect but translate to English or other Desired Language

LEARNED / TRAINED

1

GO - On the part of the project I did my sentiment analysis it was benefic for me to revise some code from that language.

2

Docker - Even tho on University we are not required to learn it, it was an interesting journey on understanding how helpful it can be in a team project, and I built my own dockerfile for thr backend engineering challenge

3

Fast API & Google API - Create RESTful APIs with FastAPI, handling requests and responses, integrating external services like Google Cloud Translation, and executing subprocesses for sentiment analysis,

4

Postgres - Used this database to store the sentences and its' language. I could have used MongoDB but since the data is so simple the use for a relation database seemed more adequate

THANK YOU SO MUCH!

Presented by **João Vasco**

Unbabel | 2024