SOLENT UNIVERSITY

Department of Science and Engineering

MSc (Hons) **Computer Engineering**

Software Design and Development – COM714

Academic Year 2024-2025

Thomas Andrews (102592142)

Git Repository: https://github.com/Urahara08/traveller-management-COM714

Git Project management: https://github.com/users/Urahara08/projects/6

Report – Assignment 1

Tutor: Daniel Olabanji

Table of Contents

Introduction:

This report aims to outline the production of a travel management software. The report will follow the order of the ICONIX process and is broken down into sections that help display the actions placed into the software to justify the production methods.

Three key diagrams have been produced to outline the production and the architecture of the software will be discussed throughout. As well as this various Unit tests have been conducted to maintain the software's functions and showcase its robustness.

Finally, a review and future works has been mentioned to outline what can be done to improve the software as well as pieces that have been removed due to code conflict.

Git repository: https://github.com/Urahara08/traveller-management-COM714

Software development:

Client requirements:

The body of the report will express the development of the software by following the ICONIX methodology. The Iconix methodology is a 'use case driven' software process that focuses on low-cost change complimented by rapid development. By combining 4 main steps the process is consistently changing whilst the software is developed so that the users' needs are met. The stages include establishing user requirements, identifying use cases, producing diagrams of domain models and sketching interfaces (Medium, 2022).

As a result of the methodology the report will be broken down into the following sections:

Table 1 ICONIX breakdown

ICONIX Process	
Phase of development:	Desired results:
1. Specification	To determine the functional requirements of the software product.
2. Modelling	To identify and produce the relationships between functions through various diagrams (UML).
3. Architecture	This section showcases the main attributes, classes and operational functions of the code and provides clear indication to how the code works together.
4. Code	This section includes the coding aspects for the code and is found in a zipped python file.
5. Testing	This section showcases the varying tests being conducted upon the code to show error handling as well as testing the overall functionality.
6. Review and feedback	This involves reviewing the process as well as the future works.

Specification:

This section aims to produce an understanding of what is required of the software and what the stakeholders are wanting from the project. I have broken the different sections down into a table and provided an explanation of the projected function.

Table 2 Stakeholders specifications

The stakeholder's requirements
The Travel Management System aims to significantly improve the efficiency of Solent Trips by replacing the existing paper-based system. The system would reduce paperwork, streamline communication between staff and travellers, and provide a centralised platform for managing all aspects of a trip. The software must be developed using Python and produced conforming to the recommended practices of the development language.

Table 3 Functional requirements and breakdown

1. Trip Management	• Create, view, update and delete trips. • Assign trip coordinator to manage trip. • Define trip details which will include, name, start date, duration and contact.
2. Traveller Management	• Create view, update and delete traveller profiles. • Gather essential traveller information (name, address, date of birth and emergency contact. • Record valid forms of government id.
3. Trip leg management	• Define the different legs of trip (location, destination, transport provider and transport method). • Specify types of leg (accommodation, interesting points and transfer points). • Add cost of leg (if applicable)
4. Trip coordinator	• Manage passengers for the trip. • Update trip legs. • Generate trip interests. • Handle payments and generate receipts.
5. Trip Manager	• Perform the tasks of trip coordinator. • Create, view, update and delete trip coordinators. • Generate total invoices for trips managed.
6. Administrator	• Perform all tasks of trip manager. • Create, view, update and delete trip managers. • View, update and delete trip invoices and payments. • Access total invoices and payment receipts for all trips.
7. Reporting and analytics	• Install analytics module that allows for admin and managers to generate reports on different aspects. • using libraries like Matplotlib or Plotly for visualising data.

Modelling:

This section exposes the relationships between the different functions. This is done by providing a detailed UML (Unified modelling language) diagram to expose and visualize the design of the system. The code is structured as a procedural program rather than an object orientated design. The difference being that the code uses dictionaries to store and manipulate data rather than using objects and classes.

The modelling section involves the creation of three different diagrams:

UML: which specifies the classers used as well as the relationships between the different sections of code so that the software interacts with each other (Miro 2024).

Use case Diagram: Which demonstrates the user's interaction with the software and at what points and what's parts of the software are accessed by which individual (Visual Paradigm (2019).

Robustness diagram: This highlights the interactions and how the segment of code interacts with each other to make sure that the functions can operate under different circumstances (Visual paradigm (2024).

All three of these diagrams can be found within the Git-repository.

Git repo: https://github.com/Urahara08/traveller-management-COM714

Architecture:

This section provides an understanding of the design of the software as well as discuss and expose how the different functions will interact and communicate with each other.

The below table showcases the main classes that provide functionality throughout the code. The code is developed as a procedural diagram rather than object orientated and as a result the other functions are provided through dictionaries rather than classes and objects.

Table 4 Relationship table

Function:	Attributes:
Trip	• **id: Unique identifier for the trip** • **name: Name of the trip** • **start_date: Date when the trip begins** • **duration: Length of the trip in days** • **coordinator: Reference to the trip coordinator (user ID)** • **contact: Contact information for the trip** • **travellers: List of traveller IDs associated with this trip** • **legs: List of trip leg IDs that make up this trip**
Traveller	• **id: Unique identifier for the traveller** • **name: Full name of the traveller** • **address: Traveler's address** • **dob: Date of birth** • **emergency_contact: Emergency contact information** • **gov_id_type: Type of government ID (passport, driver's license, etc.)** • **gov_id_number: Government ID number**
Trip Leg	• **id: Unique identifier for the trip leg** • **trip_id: Reference to the associated trip**

	• **start_location: Starting point of this leg** • **destination: Ending point of this leg** • **transport_provider: Company providing transport** • **transport_mode: Method of transportation (air, rail, bus, etc.)** • **leg_type: Type of leg (accommodation, point of interest, transfer)** • **cost: Cost of this trip leg**
User	• **id: Unique identifier for the user** • **username: User's login name** • **password: User's password** • **role: User's role in the system (coordinator, manager, or administrator)**

System Operations and functionality

Table 5 System operations table

System operations:	1. **Trip management (create, view, update, delete)** 2. **Traveler management (create, view, update, delete)** 3. **Trip leg management (create, view, update, delete)** 4. **User management based on roles (coordinator, manager, administrator)** 5. **Reporting and analytics on trips, travelers, and financial data**

Coding:

This section involves the actual coding and development of the software. This section involves the formation of the functions as well as putting the previous steps into action to make sure the code meets the desired specifications and requirements. Below are some classes that have been discussed and developed. An explanation is provided for each section to showcase understanding.

Please note that not all code is shown below. Just indications from some sections to showcase understanding. Please see repository for full code.

See Git repository for full code: https://github.com/Urahara08/traveller-management-COM714

Table 6 Trip management code breakdown

Trip Management	
Code segment:	Explanation:

def create_trip():
  """
  Create a new trip and add it to the `trips` list.
  """
  print("\n=== Create New Trip ===")
 	
  # Collect trip details from the user
  trip = {
    "id": str(uuid.uuid4())[:2], # Generate a short unique ID for the trip
    "name": get_input("Trip Name: "), # Name of the trip
 	
    "start_date": get_date_input("Start Date"), # Start date of the trip
	• This section involves the creation of the trip.
	• The trip generates an ID upon input this is done by using a ID generator and assigning the string a max id length of 2.
	• The ID assigns itself with the input of the name of the trip which is assigned the id “name”.
	• The code then requires the input of date which is assigned the id start_date. And by utilising the date/time import the date is input as follows “dd/mm/yy”.

"duration": get_int_input("Duration (days): "),
# Duration of the trip in days 	
 	
"coordinator": get_input("Trip Coordinator: "),
# ID of the trip coordinator 	
 	
"contact": get_input("Contact Information: "), # Contact details for the trip
"travelers": [],
# List of traveler IDs associated with the trip 	
 	
"legs": []
# List of trip leg IDs associated with the trip 	
}
 	
trips.append(trip)
# Add the trip to the `trips` list 	
print(f"Trip '{trip['name']}' created successfully with ID: {trip['id']}")
	• The duration is then input. This duration is assigned in days and requires the user to input a number.
	• The coordinator is then assigned which will require the user to input the id of a coordinator created in the tripcoordinator function. This ID will pull information from the list item and then assign the coordinator to the trip.
	• Contact information is then input in the form of a number.
	• The travellers are then input. This is done by drawing an ID from the traveller function. A Traveller must be created before completing the trip but it can be corrected and added further along. This ID once input will assign the created traveller to the trip.
	• The leg item Is added using the same ID strategy. This is created in the tripleg function.
	
	• Once completed and all fields are fulfilled the inputs. The function is then saved to a list item and stored in the code. The ID is then produced and displayed which will be needed for future inputs.

def view_trips(): """ *Display all trips in the system.*	• This section displays all of the trips within the system and displays the

"""
print("\n=== All Trips ===")
 	
if not trips: # Check if there are no trips
print("No trips found.")
return
 	
# Iterate through each trip and display its details 	
for trip in trips:
print(f"ID: {trip['id']}")
print(f"Name: {trip['name']}")
print(f"Start Date: {trip['start_date'].strftime('%d/%m/%Y')}") # Format the date for display
 	
print(f"Duration: {trip['duration']} days")
print(f"Coordinator: {trip['coordinator']}")
print(f"Number of Travelers: {len(trip['travelers'])}") # Count the number of travelers
 	
print(f"Number of Trip Legs: {len(trip['legs'])}") # Count the number of trip legs
 	
print("-" * 30) # Separator for readability
	saved trips from the list directory.
	• The error handling is produced as if no trips are saved then the message “No trips found” is produced.
	• The relevant sections are produced, and the user can read the trip in the order of input. The strftime command is used to determine the order in which the date is displayed.
	• The len command is used to count the traveller ids input to determine the number of travellers.
	• This is also used to count and produce the number of trip legs assigned.
	• The print separator is used so that the terminal produced the content spaced out to improve readability.

def update_trip():
"""
Update the details of an existing trip.
"""
trip_id = get_input("\nEnter Trip ID to update: ") # Get the trip ID from the user
 	
for trip in trips: # Search for the trip with the given ID
if trip['id'] == trip_id:
print(f"Updating Trip: {trip['name']}")
 	
# Update each field, allowing the user to leave it unchanged 	
trip['name'] = get_input(f"Trip Name [{trip['name']}]: ", True) or trip['name']
	• These sections are input to update and delete the trips. If needed.
	• The function operates the same as the original input and once the trip ID has been input the functions iterate the same allowing for the user to re-input details including traveller id and coordinator id.
	• The data handling and errors produced are as a result in the code not producing an input that is ‘True’ the error messages

```
        date_str = get_input(f"Start Date
[{trip['start_date'].strftime('%d/%m/%Y')}]
(DD/MM/YYYY): ", True)
        if date_str:
          try:
            day, month, year = map(int,
date_str.split('/'))
            trip['start_date'] =
datetime.date(year, month, day)
          except:
            print("Invalid date format. Start
date not updated.")

        duration_str = get_input(f"Duration
[{trip['duration']}] days: ", True)
        if duration_str:
          try:
            trip['duration'] =
int(duration_str)
          except:
            print("Invalid number. Duration
not updated.")

        trip['coordinator'] = get_input(f"Trip
Coordinator ID [{trip['coordinator']}]: ",
True) or trip['coordinator']
        trip['contact'] = get_input(f"Contact
Information [{trip['contact']}]: ", True) or
trip['contact']

        print(f"Trip '{trip['name']}' updated
successfully")
        return

    print(f"Trip with ID {trip_id} not found.")
# If no trip matches the given ID


def delete_trip():
    """
    Delete a trip from the system.
    """
    trip_id = get_input("\nEnter Trip ID to
delete: ")  # Get the trip ID from the user

    for i, trip in enumerate(trips):  # Search
for the trip with the given ID
```

then read as shown in the code to the left.

- If the code is successfully update, then a message is produced to say that this has been done and vice versa.

```
        if trip['id'] == trip_id:
            trip_name = trip['name']
            del trips[i]  # Remove the trip from the list
            print(f"Trip '{trip_name}' deleted successfully")
            return

    print(f"Trip with ID {trip_id} not found.")  # If no trip matches the given ID
```

Table 7 Traveller management code breakdown

Traveller management	

`# Traveller management functions`
`def create_traveller():`
`    """Create a new traveller profile"""`
`    print("\n=== Create New Traveler ===")`
 	
`    traveller = {`
`        "id": str(uuid.uuid4())[:8],  # Generate a short unique ID`
`        "name": get_input("Full Name: "),`
`        "address": get_input("Address: "),`
`        "dob": get_date_input("Date of Birth"),`
`        "emergency_contact": get_input("Emergency Contact: "),`
`        "gov_id_type": get_input("Government ID Type: "),`
`        "gov_id_number": get_input("Government ID Number: ")`
`    }`
`    travelers.append(traveller)`
`    print(f"Traveller '{traveller['name']}' created successfully with ID: {traveler['id']}")`
 	
`def view_travelers():`
`    """Display all travelers"""`
`    print("\n=== All Travelers ===")`
`    if not travelers:`
`        print("No travelers found.")`
`        return`
`    for traveler in travelers:`
`        print(f"ID: {traveler['id']}")`
`        print(f"Name: {traveler['name']}")`
`        print(f"Date of Birth: {traveler['dob'].strftime('%d/%m/%Y')}")`
`        print(f"ID Type: {traveler['gov_id_type']}")`
`        print(f"ID Number: {traveler['gov_id_number']}")`
`        print("-" * 30)`
	• This section includes the creation of a traveller.
	• The main function creates a traveller and produces a unique ID that is then associated with the traveller which can be input at different stages.
	• The main details and subclasses include getting a name, address, date of birth, emergency contact and then various IDs.
	• Once the traveller is created a message is produced as well as the traveller ID for reference.
	• As well as this the user can view the travellers to refer to the ID. This is handled by placing an error message to indicate if no travellers have been created.
	• The traveller's information is produced in order of input and the date and time is placed in the sae order as the trip management by using the strftime function.

```
def update_traveler():
  """Update an existing traveler"""
  traveler_id = get_input("\nEnter Traveler
ID to update: ")

  for traveler in travelers:
    if traveler['id'] == traveler_id:
      print(f"Updating Traveler:
{traveler['name']}")

      traveler['name'] = get_input(f"Full
Name [{traveler['name']}]: ", True) or
traveler['name']
      traveler['address'] =
get_input(f"Address [{traveler['address']}]:
", True) or traveler['address']

      date_str = get_input(f"Date of Birth
[{traveler['dob'].strftime('%d/%m/%Y')}]
(DD/MM/YYYY): ", True)
      if date_str:
        try:
          day, month, year = map(int,
date_str.split('/'))
          traveler['dob'] =
datetime.date(year, month, day)
        except:
          print("Invalid date format. Date
of birth not updated.")

      traveler['emergency_contact'] =
get_input(f"Emergency Contact
[{traveler['emergency_contact']}]: ", True)
or \

traveler['emergency_contact']
      traveler['gov_id_type'] =
get_input(f"Government ID Type
[{traveler['gov_id_type']}]: ", True) or
traveler[
        'gov_id_type']
      traveler['gov_id_number'] =
get_input(f"Government ID Number
[{traveler['gov_id_number']}]: ", True) or \
```

- This section produces the ability to update the traveller's information so that details can bed input at a later stage (including various IDs).
- The update is done the same as the trips and works step by step so that the information can be input. By returning true the next input can then be put into the terminal.
- Various error handling messages have been input so that the information is put in correctly.

```
def delete_traveler():
    """Delete a traveler"""
    traveler_id = get_input("\nEnter Traveler ID to delete: ")

    for i, traveler in enumerate(travelers):
        if traveler['id'] == traveler_id:
            traveler_name = traveler['name']
            del travelers[i]
            print(f"Traveler '{traveler_name}' deleted successfully")
            return

    print(f"Traveler with ID {traveler_id} not found.")
```
	• This section allows for deletion of the traveller.
	• You must enter the ID of the traveller first and then once the traveller has been removed it will be removed from the list and directory. The message is then produced to confirm that the deletion has run true.

Table 8 Trip leg management code breakdown

Trip leg management	

# Trip leg management functions 	
 	
def create_trip_leg():
    """Create a new trip leg"""
    print("\n=== Create New Trip Leg ===")
 	
    trip_id = get_input("Trip ID: ")
 	
    # Check if trip exists
    trip_exists = False
    for trip in trips:
        if trip['id'] == trip_id:
            trip_exists = True
            break
 	
    if not trip_exists:
        print(f"Trip with ID {trip_id} not found.")
        return
 	
    leg = {
        "id": str(uuid.uuid4())[:8], # Generate a short unique ID
        "trip_id": trip_id,
        "start_location": get_input("Starting Location: "),
        "destination": get_input("Destination: "),
        "transport_provider": get_input("Transport Provider: "),
        "transport_mode": get_input("Mode of Transport: "),
        "leg_type": get_input("Leg Type (accommodation/poi/transfer): "),
        "cost": get_int_input("Cost: ")
    }
	• This section allows the user to create different trip legs that can then be assigned to the trips with the generated ID.
	• The trip ID is entered and upon that trip legs can be assigned to the trip themselves.
	• The code will look for the trip using the ID and if no trip can be found an error will be produced. This is done by returning false.
	• If returning true then the code will iterate through different questions including transport, destination, transport provider and cost.
	• The append is similar to both the traveller and the trip in that once an ID is entered the user will iterate through the same questions and change them accordingly. This is also managed by returning true or false and producing a message if false.
	• From this you can also view the trip legs associated with the trips.
	• You must enter the ID and with that the legs will appear and if not, then an error message will be produced.
	• From this code you can also delete a trip leg. Similar to both trip and traveller the ID must be input and if it matches and returns true then you can delete the trip leg from the string and ultimately the list storing the data.
	• You can also view the trip leg stored In the list item by inputting the ID. It will then produce the fields provided systematically.

Table 9 Users code breakdown

Users	

# User management functions 	
 	
def create_user():
    """Create a new user (coordinator/manager/admin)"""
    print("\n=== Create New User ===")
 	
    role = ""
    while role not in ["coordinator", "manager", "administrator"]:
        role = get_input("Role (coordinator/manager/administrator): ").lower()
        if role not in ["coordinator", "manager", "administrator"]:
            print("Invalid role. Please enter coordinator, manager, or administrator.")
	• The User class allows users to input new users to the system so that they can access the system themselves.
	• By applying different user credentials basic security can be added to the code. As a result, the credentials are saved as strings whilst the code is running.
	• Different levels of access can also be applied to the users.
	• Again, much like the previous sections users can be viewed as well as adjusted as see fit. All that is needed is the ID of the user.

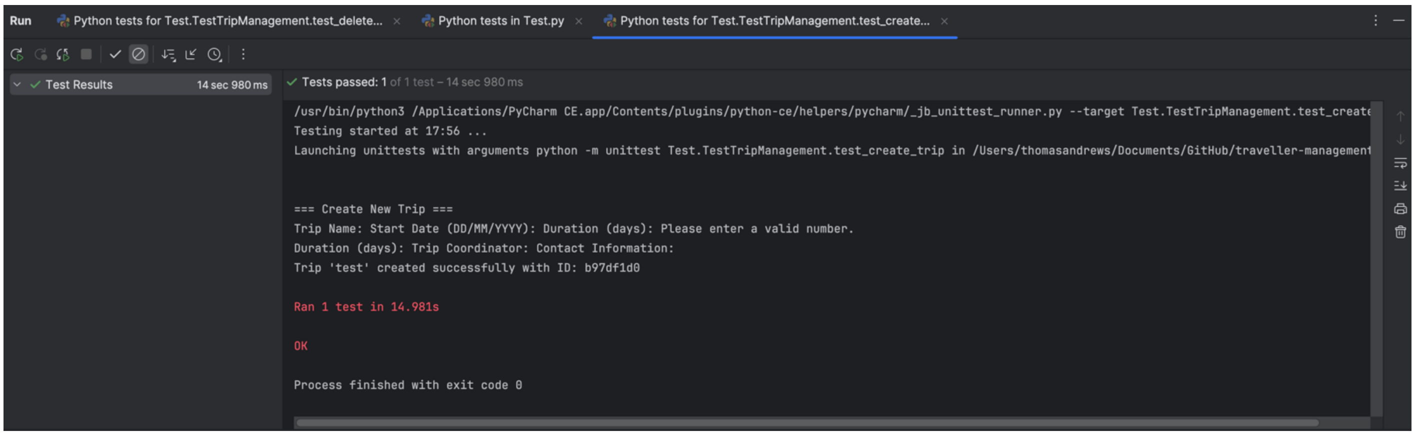
Testing:

Unit testing using Unittest import.

This section involves testing the individual functions to make sure that they are working and producing the correct results and that they function together. Below are a series of screenshots showing the successful tests. This has been done by creating a separate file for testing and running the test individually. The code for each test can be located on the test.py file. Each one is separated and labelled using comments. But the results are as shown below.

Trip management

Create trip:

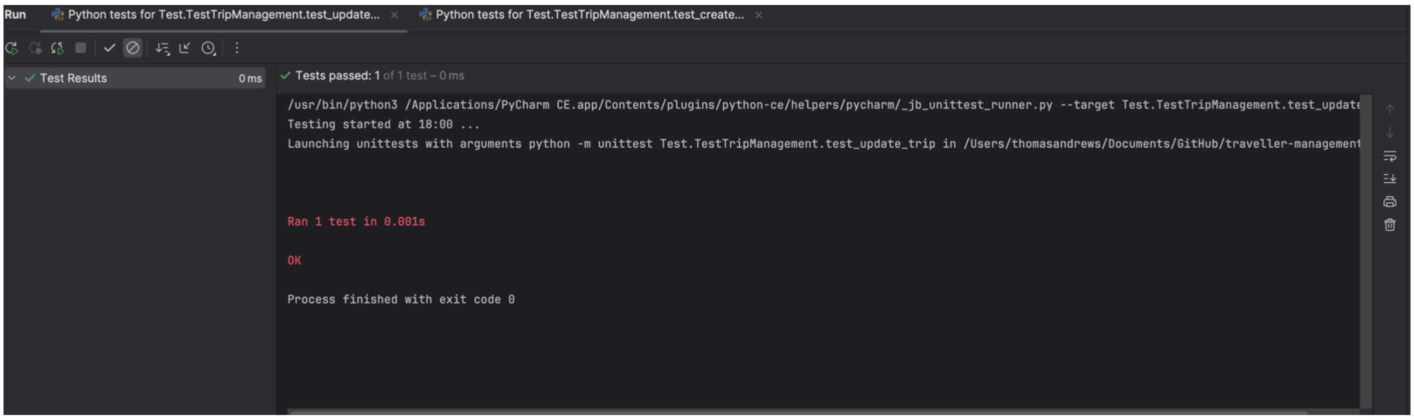

Run
Python tests for Test.TestTripManagement.test_delete...
Python tests in Test.py
Python tests for Test.TestTripManagement.test_create...
Test Results
14 sec 980 ms

Tests passed: 1 of 1 test – 14 sec 980 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTripManagement.test_create
Testing started at 17:56 ...
Launching unittests with arguments python -m unittest Test.TestTripManagement.test_create_trip in /Users/thomasandrews/Documents/GitHub/traveller-management


=== Create New Trip ===
Trip Name: Start Date (DD/MM/YYYY): Duration (days): Please enter a valid number.
Duration (days): Trip Coordinator: Contact Information:
Trip 'test' created successfully with ID: b97df1d0

Ran 1 test in 14.981s

OK

Process finished with exit code 0
```

Figure 1 Create trip unit test

Update trip:

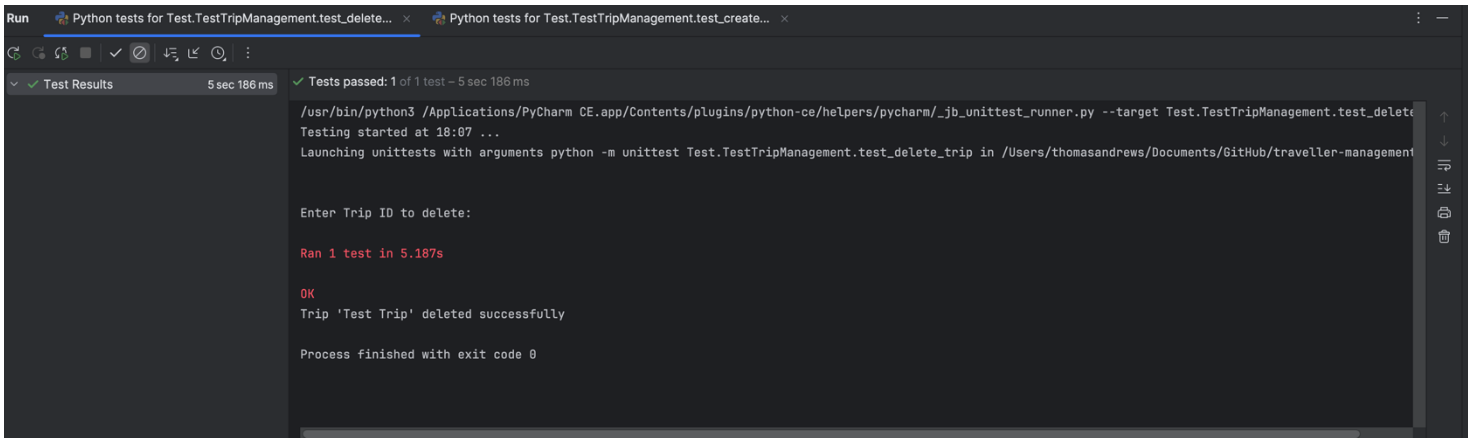

Run
Python tests for Test.TestTripManagement.test_update...
Python tests for Test.TestTripManagement.test_create...
Test Results
0 ms

Tests passed: 1 of 1 test – 0 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTripManagement.test_update
Testing started at 18:00 ...
Launching unittests with arguments python -m unittest Test.TestTripManagement.test_update_trip in /Users/thomasandrews/Documents/GitHub/traveller-management



Ran 1 test in 0.001s

OK

Process finished with exit code 0
```

Figure 2 Update trip unit test

Delete trip:



Run | Python tests for Test.TestTripManagement.test_delete... | Python tests for Test.TestTripManagement.test_create...

Test Results 5 sec 186 ms

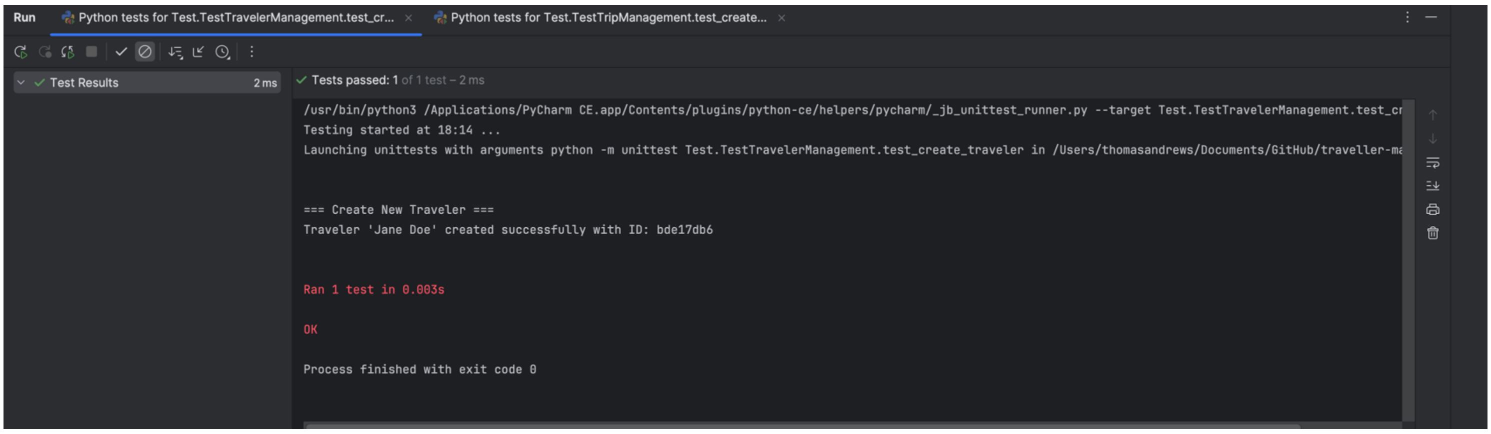
Tests passed: 1 of 1 test – 5 sec 186 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTripManagement.test_delete
Testing started at 18:07 ...
Launching unittests with arguments python -m unittest Test.TestTripManagement.test_delete_trip in /Users/thomasandrews/Documents/GitHub/traveller-management


Enter Trip ID to delete:

Ran 1 test in 5.187s

OK
Trip 'Test Trip' deleted successfully

Process finished with exit code 0
```

Figure 3 Delete trip unit test

Traveller Management

Create traveller:



Run | Python tests for Test.TestTravelerManagement.test_cr... | Python tests for Test.TestTripManagement.test_create...

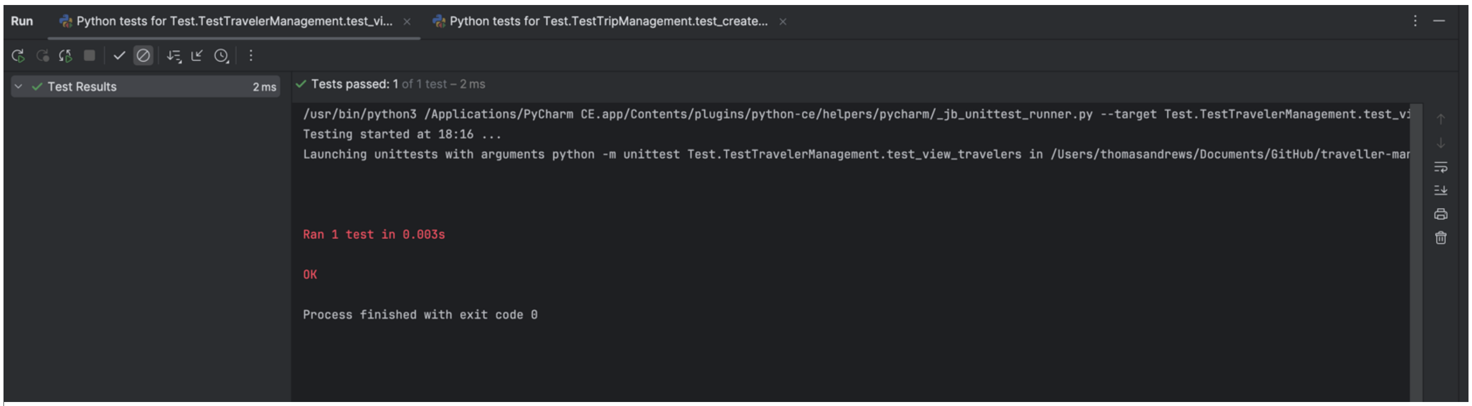
Test Results 2 ms

Tests passed: 1 of 1 test – 2 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTravelerManagement.test_cr
Testing started at 18:14 ...
Launching unittests with arguments python -m unittest Test.TestTravelerManagement.test_create_traveler in /Users/thomasandrews/Documents/GitHub/traveller-ma


=== Create New Traveler ===
Traveler 'Jane Doe' created successfully with ID: bde17db6


Ran 1 test in 0.003s

OK

Process finished with exit code 0
```

Figure 4 create traveller unit test

View traveller:



Run | Python tests for Test.TestTravelerManagement.test_vi... | Python tests for Test.TestTripManagement.test_create...

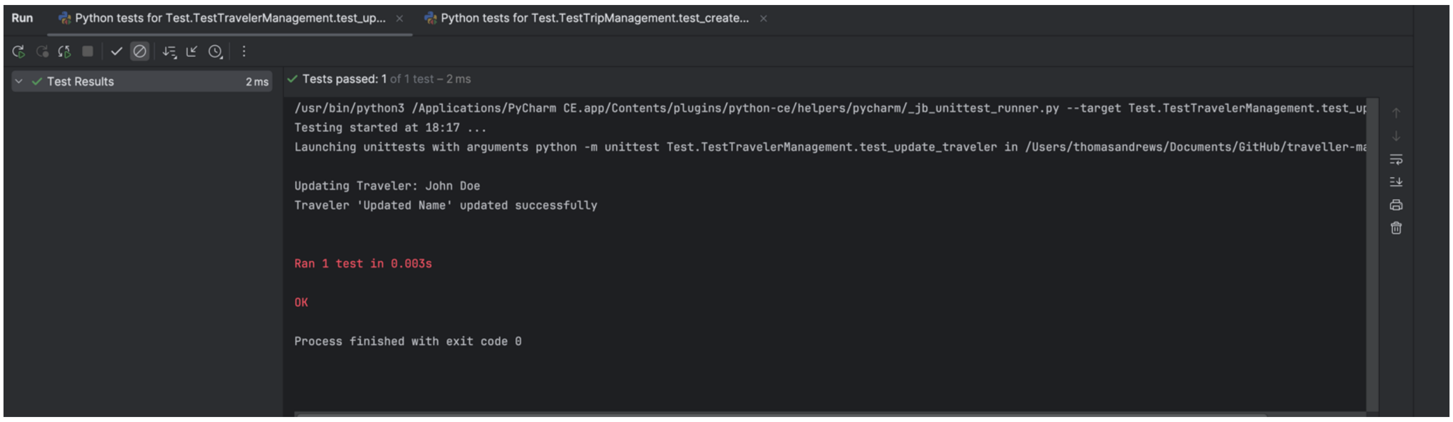
Test Results 2 ms

Tests passed: 1 of 1 test – 2 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTravelerManagement.test_vi
Testing started at 18:16 ...
Launching unittests with arguments python -m unittest Test.TestTravelerManagement.test_view_travelers in /Users/thomasandrews/Documents/GitHub/traveller-mar



Ran 1 test in 0.003s

OK

Process finished with exit code 0
```

Figure 5 View traveller unit test

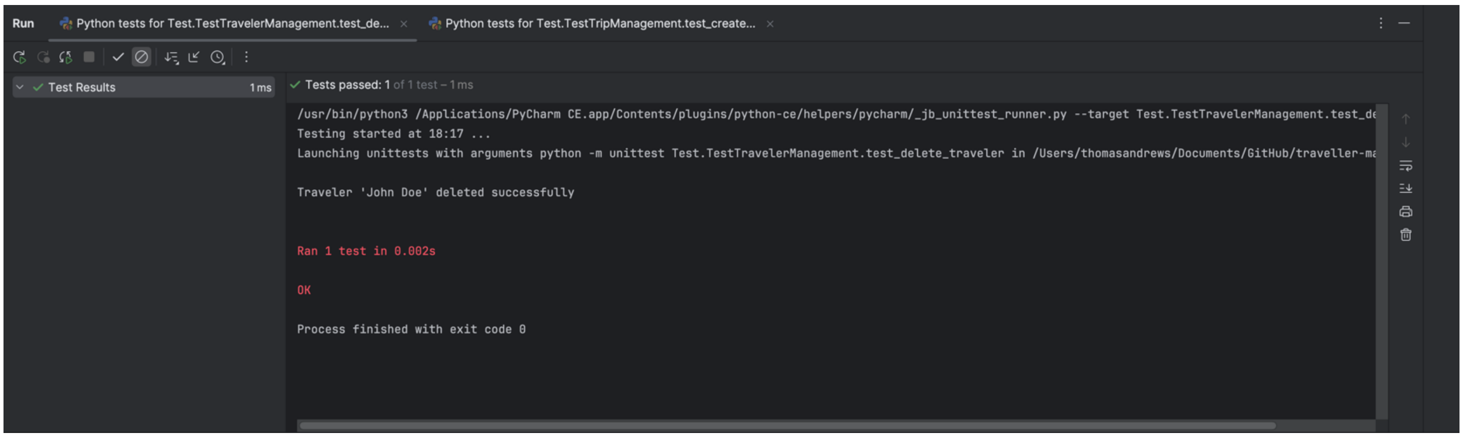
Update Traveller:


Run
Python tests for Test.TestTravelerManagement.test_up...
Python tests for Test.TestTripManagement.test_create...
Test Results 2 ms
Tests passed: 1 of 1 test – 2 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTravelerManagement.test_up
Testing started at 18:17 ...
Launching unittests with arguments python -m unittest Test.TestTravelerManagement.test_update_traveler in /Users/thomasandrews/Documents/GitHub/traveller-ma

Updating Traveler: John Doe
Traveler 'Updated Name' updated successfully


Ran 1 test in 0.003s

OK

Process finished with exit code 0
```

Figure 6 Update traveller unit test

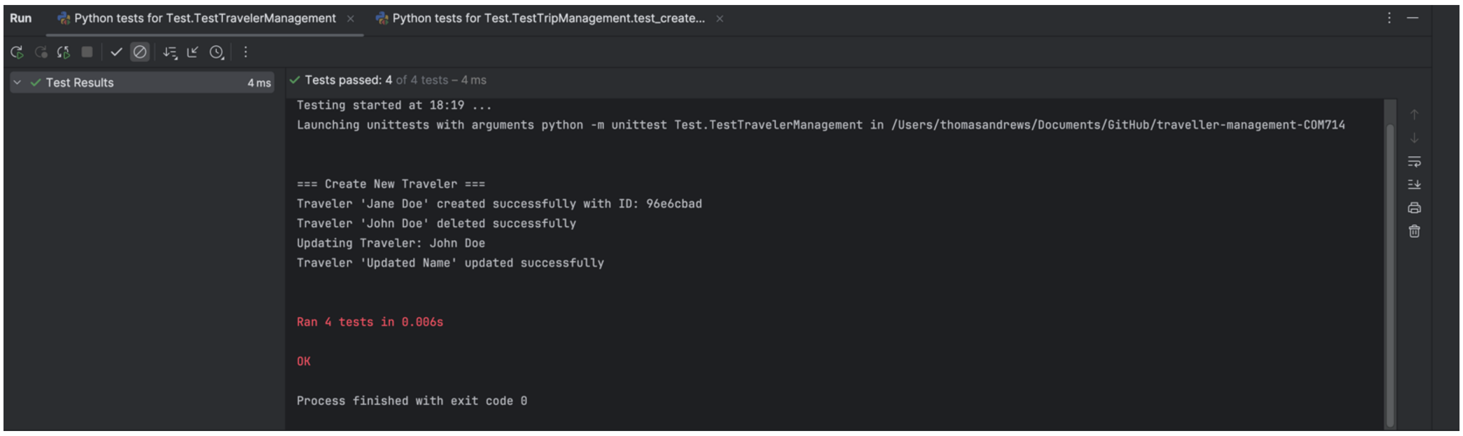
Delete Traveller:


Run
Python tests for Test.TestTravelerManagement.test_de...
Python tests for Test.TestTripManagement.test_create...
Test Results 1 ms
Tests passed: 1 of 1 test – 1 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestTravelerManagement.test_de
Testing started at 18:17 ...
Launching unittests with arguments python -m unittest Test.TestTravelerManagement.test_delete_traveler in /Users/thomasandrews/Documents/GitHub/traveller-ma

Traveler 'John Doe' deleted successfully


Ran 1 test in 0.002s

OK

Process finished with exit code 0
```

Figure 7 Delete traveller unit test

All Traveller functions working together:

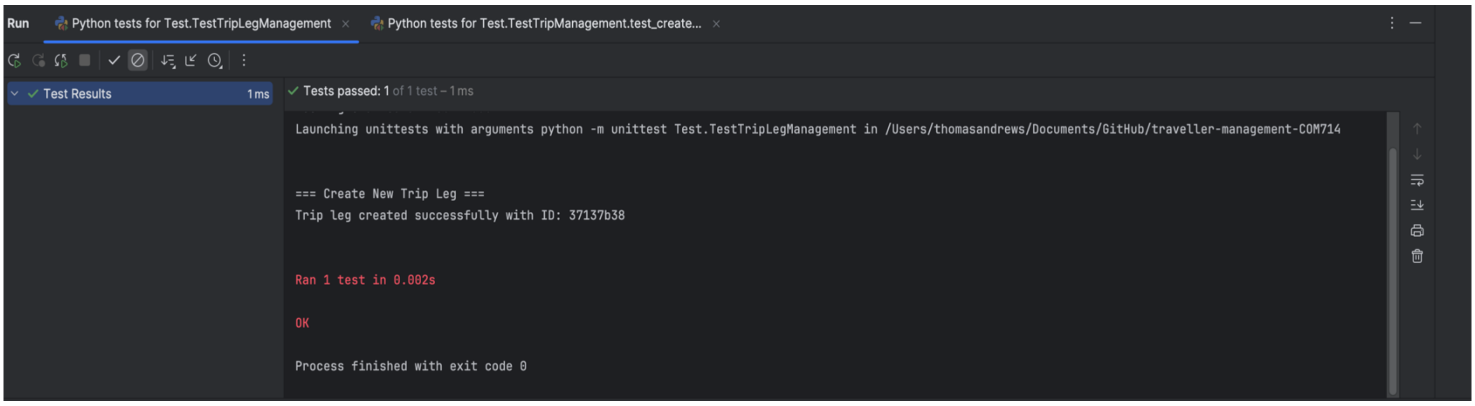

Run
Python tests for Test.TestTravelerManagement
Python tests for Test.TestTripManagement.test_create...
Test Results 4 ms
Tests passed: 4 of 4 tests – 4 ms

```
Testing started at 18:19 ...
Launching unittests with arguments python -m unittest Test.TestTravelerManagement in /Users/thomasandrews/Documents/GitHub/traveller-management-COM714


=== Create New Traveler ===
Traveler 'Jane Doe' created successfully with ID: 96e6cbad
Traveler 'John Doe' deleted successfully
Updating Traveler: John Doe
Traveler 'Updated Name' updated successfully


Ran 4 tests in 0.006s

OK

Process finished with exit code 0
```

Figure 8 Traveller unit test

Trip leg management

Create trip leg:

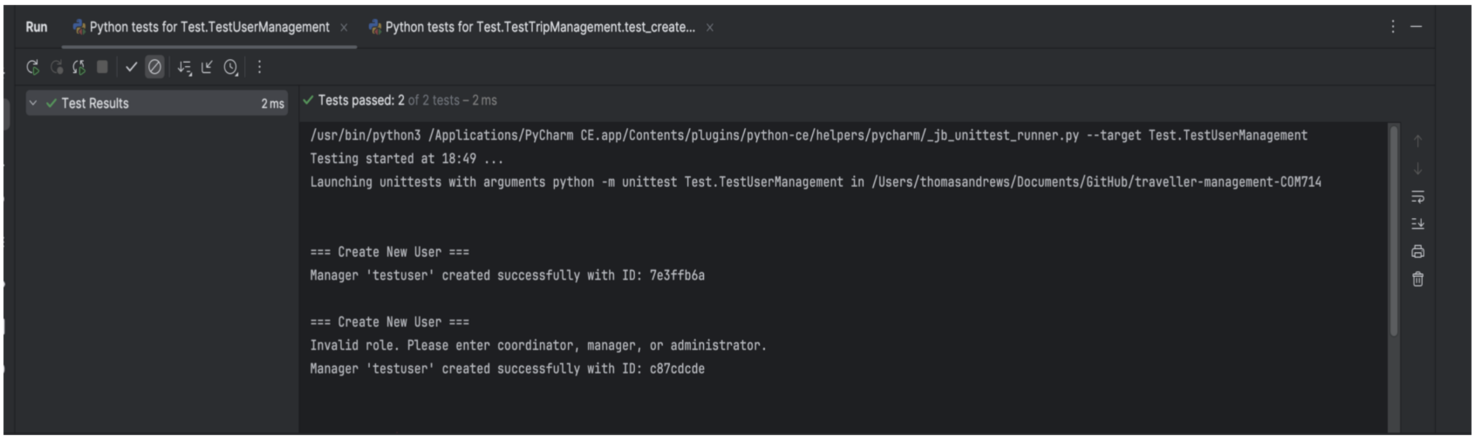

Run
Python tests for Test.TestTripLegManagement
Python tests for Test.TestTripManagement.test_create...
Test Results
1 ms

```
Tests passed: 1 of 1 test – 1 ms
Launching unittests with arguments python -m unittest Test.TestTripLegManagement in /Users/thomasandrews/Documents/GitHub/traveller-management-COM714


=== Create New Trip Leg ===
Trip leg created successfully with ID: 37137b38


Ran 1 test in 0.002s

OK

Process finished with exit code 0
```

Figure 9Create trip leg unit test

User management

Create user and user function tests as well as error handling:

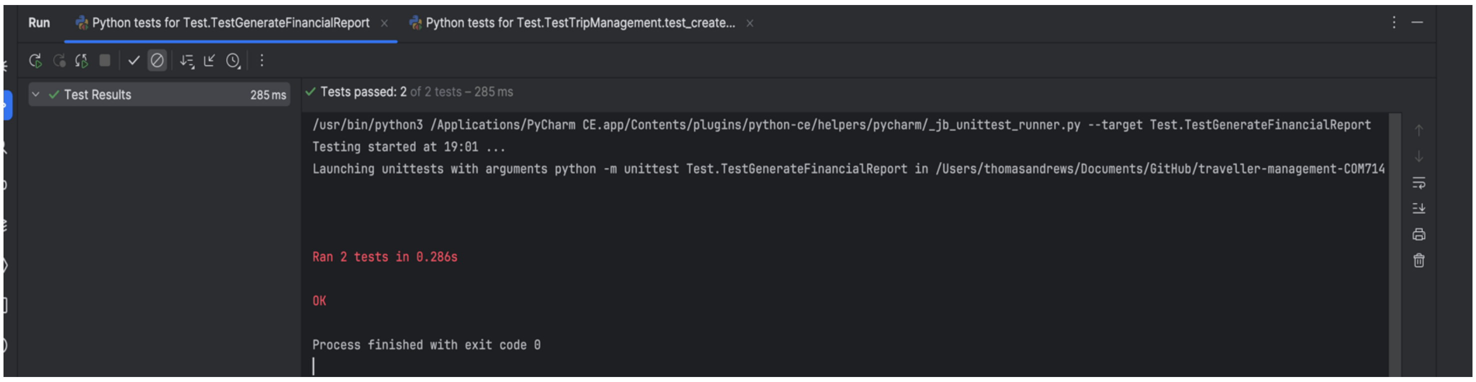

Run
Python tests for Test.TestUserManagement
Python tests for Test.TestTripManagement.test_create...
Test Results
2 ms

```
Tests passed: 2 of 2 tests – 2 ms
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestUserManagement
Testing started at 18:49 ...
Launching unittests with arguments python -m unittest Test.TestUserManagement in /Users/thomasandrews/Documents/GitHub/traveller-management-COM714


=== Create New User ===
Manager 'testuser' created successfully with ID: 7e3ffb6a

=== Create New User ===
Invalid role. Please enter coordinator, manager, or administrator.
Manager 'testuser' created successfully with ID: c87cdcde
```

Figure 10 User management unit test

Reporting and analytics

Testing report generation as well as error handling if no data.

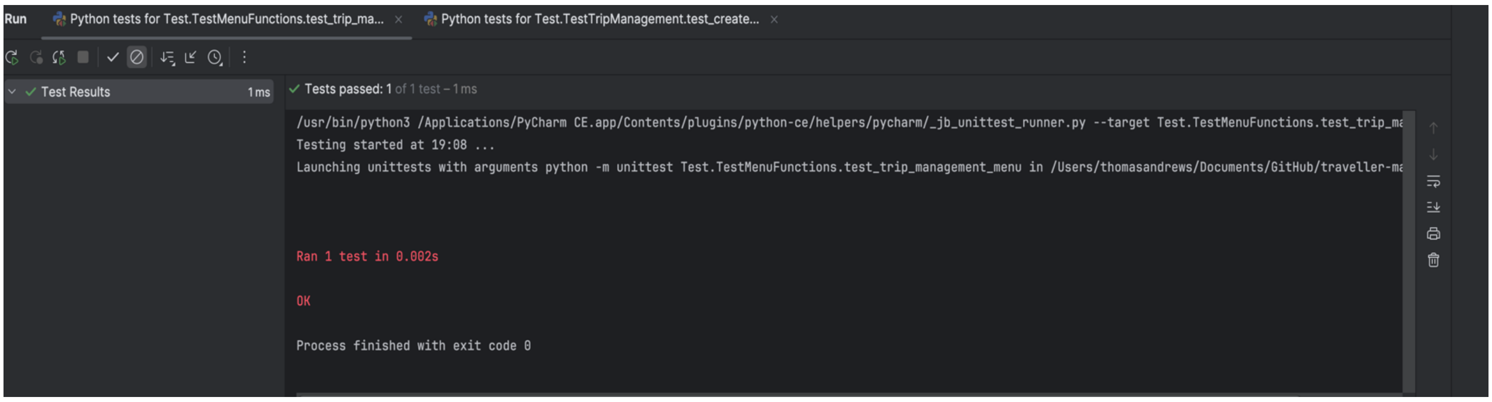

Run
Python tests for Test.TestGenerateFinancialReport
Python tests for Test.TestTripManagement.test_create...
Test Results
285 ms

```
Tests passed: 2 of 2 tests – 285 ms
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestGenerateFinancialReport
Testing started at 19:01 ...
Launching unittests with arguments python -m unittest Test.TestGenerateFinancialReport in /Users/thomasandrews/Documents/GitHub/traveller-management-COM714


Ran 2 tests in 0.286s

OK

Process finished with exit code 0
```

Figure 11 reporting and analytics unit test

Main Menu

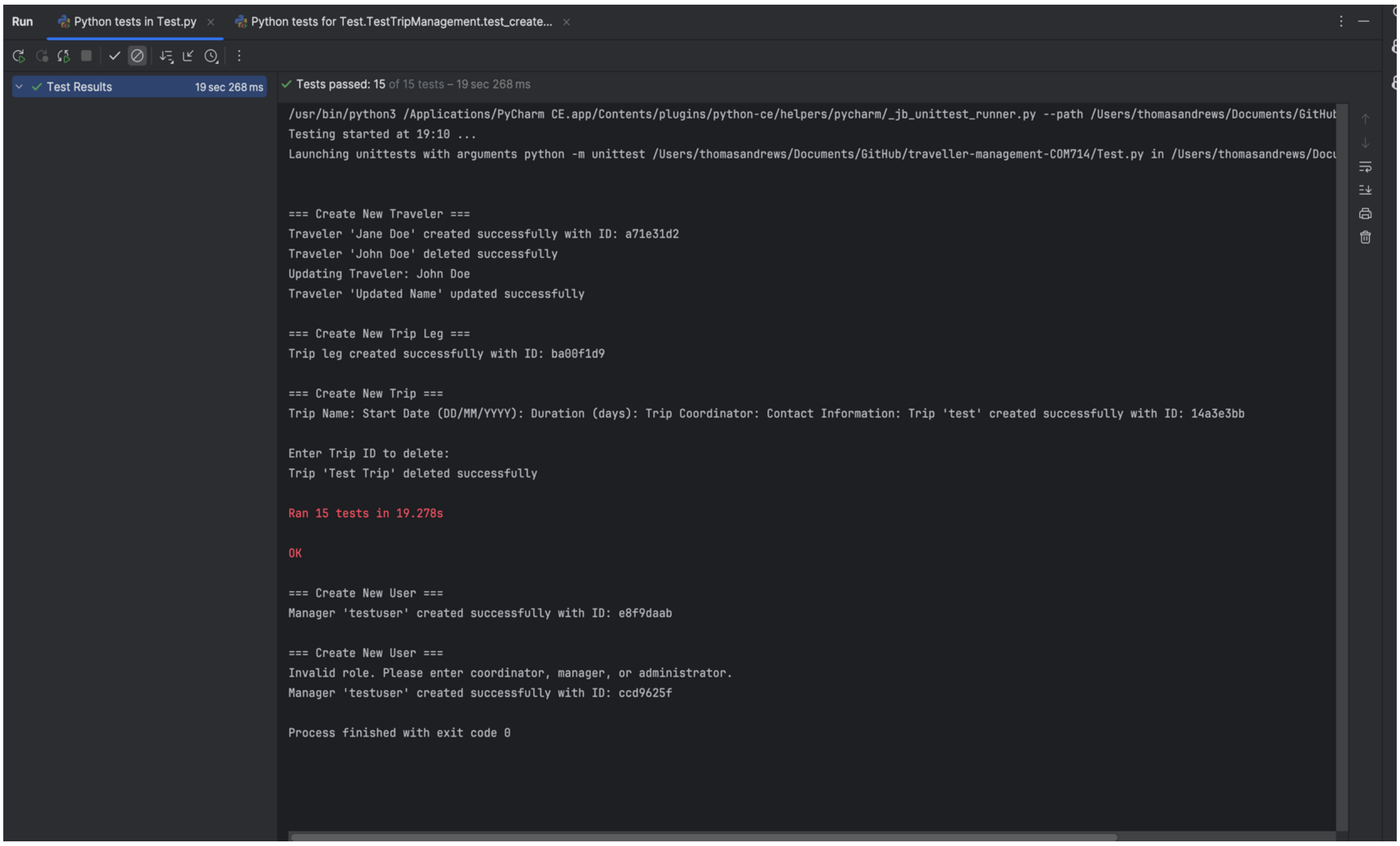
Menu functionality


Run
Python tests for Test.TestMenuFunctions.test_trip_ma...
Python tests for Test.TestTripManagement.test_create...
Test Results 1 ms
Tests passed: 1 of 1 test – 1 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --target Test.TestMenuFunctions.test_trip_ma
Testing started at 19:08 ...
Launching unittests with arguments python -m unittest Test.TestMenuFunctions.test_trip_management_menu in /Users/thomasandrews/Documents/GitHub/traveller-ma



Ran 1 test in 0.002s

OK

Process finished with exit code 0
```

Figure 12 Menu functionality unit test

Full comprehensive test of full test.py file


Run
Python tests in Test.py
Python tests for Test.TestTripManagement.test_create...
Test Results 19 sec 268 ms
Tests passed: 15 of 15 tests – 19 sec 268 ms

```
/usr/bin/python3 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --path /Users/thomasandrews/Documents/GitHub
Testing started at 19:10 ...
Launching unittests with arguments python -m unittest /Users/thomasandrews/Documents/GitHub/traveller-management-COM714/Test.py in /Users/thomasandrews/Docu


=== Create New Traveler ===
Traveler 'Jane Doe' created successfully with ID: a71e31d2
Traveler 'John Doe' deleted successfully
Updating Traveler: John Doe
Traveler 'Updated Name' updated successfully

=== Create New Trip Leg ===
Trip leg created successfully with ID: ba00f1d9

=== Create New Trip ===
Trip Name: Start Date (DD/MM/YYYY): Duration (days): Trip Coordinator: Contact Information: Trip 'test' created successfully with ID: 14a3e3bb

Enter Trip ID to delete:
Trip 'Test Trip' deleted successfully

Ran 15 tests in 19.278s

OK

=== Create New User ===
Manager 'testuser' created successfully with ID: e8f9daab

=== Create New User ===
Invalid role. Please enter coordinator, manager, or administrator.
Manager 'testuser' created successfully with ID: ccd9625f

Process finished with exit code 0
```

Figure 13 Full code unit test

Review/conclusion:

This section involves the review aspect of the Iconix framework.

Requirements for reference:

Desired function:	Functional requirement:
1. Trip Management	• Create, view, update and delete trips. • Assign trip coordinator to manage trip. • Define trip details which will include, name, start date, duration and contact.
2. Traveller Management	• Create view, update and delete traveller profiles. • Gather essential traveller information (name, address, date of birth and emergency contact. • Record valid forms of government id.
3. Trip leg management	• Define the different legs of trip (location, destination, transport provider and transport method). • Specify types of leg (accommodation, interesting points and transfer points). • Add cost of leg (if applicable)
4. Trip coordinator	• Manage passengers for the trip. • Update trip legs. • Generate trip interests. • Handle payments and generate receipts.
5. Trip Manager	• Perform the tasks of trip coordinator. • Create, view, update and delete trip coordinators. • Generate total invoices for trips managed.
6. Administrator	• Perform all tasks of trip manager. • Create, view, update and delete trip managers. • View, update and delete trip invoices and payments. • Access total invoices and payment receipts for all trips.
7. Reporting and analytics	• Install analytics module that allows for admin and managers to generate reports on different aspects. • using libraries like Matplotlib or Plotly for visualising data.

The software meets the desired functional requirements. As demonstrated through the testing the code successfully fulfils the functional requirements outlined throughout the report. The only thing missing from the code is the GUI aspect which was not fulfilled due to lack of time as well as the lack of understanding. The GUI activities kept interfering with the code and it was difficult to identify where to place the interface alongside the different classes. As a result, this was removed.

The testing file was successfully produced and although aspects of the testing were not successful. Overall these were minor changes including the conflict of date.time and start_date. But this was easily overcome. The production of unit testing as well as the production of an effective UML diagram took longer than anticipated and due to an issue with GitHub the connections had to be re-established, and the commits had to be pushed once again.

Overall, the project effectively provides the necessary functions and works as an effective travel management software.

Future works:

In the future I will place more time into the development of a user interface and work towards implementing a GUI. As well as this I will work to implement a database for storing data and provide a more systematic use of classes so that the list elements do not have to be used for storing classes.

References:

Haraphat01 (2022). Iconix Software Process - Haraphat01 - Medium. [online] Medium. Available at: https://medium.com/@haraphat/iconix-software-process-d58de03d9602 [Accessed 4 Apr. 2025].

Miro (2024). What is a UML diagram, and what is it used for? | Miro. [online] https://miro.com/. Available at: https://miro.com/diagramming/what-is-a-uml-diagram/. Accessed [20th March 2025]

Nishadha (2022). Use case diagram tutorial (guide with examples) | creately. [online] creately.com. Available at: https://creately.com/guides/use-case-diagram-tutorial/. Accessed [20th March 2025]

Visual Paradigm (2019). What is Use Case Diagram? [online] Visual-paradigm.com. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/. Accessed [20th March 2025]

Visual paradigm (2024). A Practical Tutorial on Robustness Analysis. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/. Accessed [20th March 2025]

www.visual-paradigm.com. (n.d.). A Practical Tutorial on Robustness Analysis. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/. Accessed [20th March 2025]