System Requirements Specification Index

For

Wildlife Conservation Tracking System

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Wildlife Conservation Tracking System

System Requirements Specification

1 PROJECT ABSTRACT

The Indian Wildlife Foundation (IWF) requires a specialized tracking system to monitor endangered species across various wildlife sanctuaries in India. The system will track animal populations, monitor habitat conditions, record conservation efforts, and generate analytical reports. It will enable conservation officers to efficiently manage wildlife data by categorizing species by conservation status, filtering by habitat type, updating population counts, and calculating conservation metrics. This system provides an efficient way for wildlife experts to organize and analyze critical conservation data.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System needs to store and categorize different types of wildlife data (species, population, habitat, conservation status) 2. System must support filtering species by conservation status, habitat type, population trend, or specific sanctuary 3. Console should handle different dictionary operations like Basic filtering (species by conservation status, habitat type), Dictionary comprehension (for population ranges, habitat conditions), Dictionary methods (update, get, items), Dictionary merging and transformation, Dictionary-based data analysis

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Species Records:
    - Must be stored as dictionaries with fields for id, name, scientific_name, conservation_status, population, habitat_type, sanctuaries, threats
    - Must be stored in a dictionary variable with species ID as key
    - Example: `"SP001": {"name": "Bengal Tiger", "scientific_name": "Panthera tigris tigris", "conservation_status": "Endangered", "population": 3500, "habitat_type": "Forest", "sanctuaries": ["Sundarbans", "Jim Corbett", "Bandhavgarh"], "threats": ["Poaching", "Habitat Loss", "Human Conflict"]}`
2. Conservation Status:
    - Must be one of: "Least Concern", "Near Threatened", "Vulnerable", "Endangered", "Critically Endangered"
    - Must be stored as string in species' "conservation_status" field
    - Example: "Endangered"
3. Habitat Types:
    - Must be one of: "Forest", "Grassland", "Wetland", "Mountain", "Desert"
    - Must be stored as string in species' "habitat_type" field
    - Example: "Forest"
4. Population:
    - Must be stored as integer in "population" field
    - Must be greater than or equal to 0
    - Example: 3500
5. Predefined Species:
    - Must use these exact predefined species in the initial species dictionary:
        - `"SP001": {"name": "Bengal Tiger", "scientific_name": "Panthera tigris tigris", "conservation_status": "Endangered", "population": 3500, "habitat_type": "Forest", "sanctuaries": ["Sundarbans", "Jim Corbett",

"Bandhavgarh"], "threats": ["Poaching", "Habitat Loss", "Human Conflict"]}`

- `"SP002": {"name": "Asian Elephant", "scientific_name": "Elephas maximus", "conservation_status": "Endangered", "population": 27000, "habitat_type": "Forest", "sanctuaries": ["Periyar", "Nagarhole", "Jim Corbett"], "threats": ["Habitat Loss", "Human Conflict", "Poaching"]}`
- `"SP003": {"name": "Indian Rhinoceros", "scientific_name": "Rhinoceros unicornis", "conservation_status": "Vulnerable", "population": 3600, "habitat_type": "Grassland", "sanctuaries": ["Kaziranga", "Manas", "Orang"], "threats": ["Poaching", "Habitat Loss", "Flooding"]}`
- `"SP004": {"name": "Snow Leopard", "scientific_name": "Panthera uncia", "conservation_status": "Vulnerable", "population": 450, "habitat_type": "Mountain", "sanctuaries": ["Hemis", "Pin Valley", "Great Himalayan"], "threats": ["Climate Change", "Poaching", "Prey Depletion"]}`
- `"SP005": {"name": "Indian Vulture", "scientific_name": "Gyps indicus", "conservation_status": "Critically Endangered", "population": 30000, "habitat_type": "Grassland", "sanctuaries": ["Ranthambore", "Pench", "Bandhavgarh"], "threats": ["Diclofenac Poisoning", "Habitat Loss", "Food Scarcity"]}`

6. New Species:

   - Must use these exact predefined items in the new species dictionary:
     - `"NS001": {"name": "Ganges River Dolphin", "scientific_name": "Platanista gangetica", "conservation_status": "Endangered", "population": 3500, "habitat_type": "Wetland", "sanctuaries": ["Vikramshila", "National Chambal", "Katerniaghat"], "threats": ["Water Pollution", "Fishing Nets", "Dams"]}`
     - `"NS002": {"name": "Great Indian Bustard", "scientific_name": "Ardeotis nigriceps", "conservation_status": "Critically Endangered", "population": 150, "habitat_type": "Grassland", "sanctuaries": ["Desert National Park", "Kutch Bustard", "Rollapadu"], "threats": ["Habitat Loss", "Power Lines", "Predation"]}`

3.2 Operations Constraints

1. Dictionary Creation:

   - Must use proper dictionary creation syntax
   - Example: `{"id": "SP001", "name": "Species Name", ...}`

2. Dictionary Access:
    - Must use proper key access methods
    - Example: `species_data[species_id]` or `species_data.get(species_id)`
3. Dictionary Filtering:
    - Must use dictionary comprehension
    - Example: `{sid: species for sid, species in species_data.items() if species["conservation_status"] == "Endangered"}`
4. Dictionary Merging:
    - Must use dictionary unpacking
    - Example: `{**existing_dict, **new_dict}`
5. Dictionary Transformation:
    - Must use dictionary unpacking with modification
    - Example: `{**species, "newly_added": True}`
6. Dictionary Methods:
    - Must use dictionary methods (keys, values, items)
    - Example: `species_data.items()`, `species_data.keys()`
7. Dictionary-based Analytics:
    - Must use dictionaries for storing and calculating statistics
    - Example: `status_counts = {}`
8. Error Handling:
    - Must check if keys exist before accessing
    - Example: `if species_id in species_data:`
9. Immutability:
    - Must create new dictionaries rather than modifying in place
    - Example: `updated_species_data = species_data.copy()`

10. Dictionary Comprehension:

- Must use dictionary comprehension for filtering and transforming
- Example: `{k: v for k, v in d.items() if condition}`

3.3 Output Constraints

1. Display Format:
   - Show species ID, name, scientific name, conservation status, population, habitat type, sanctuaries, threats
     - § Format population with thousands separator
     - § Format conservation status with color indicators (optional)
     - § Each species must be displayed on a new line

2. Output Format:
   - Must show in this order:
     - § Show "== WILDLIFE CONSERVATION TRACKING SYSTEM =="
     - § Show "Total Species: {count}"
     - § Show "Conservation Statuses: {statuses}"
     - § Show "Current Species Data:"
     - § Show species with format: "{id} | {name} ({scientific_name}) | {conservation_status} | Population: {population} | Habitat: {habitat_type} | Sanctuaries: {sanctuaries} | Threats: {threats}"
     - § Show "Filtered Results:" when displaying search results

4. Template code Structure:

1. Data Management Functions:

- `initialize_data()` - creates the initial species and new species dictionaries

2. Dictionary Operation Functions:

- `filter_by_conservation_status(species_data, status)` - filters species by conservation status
- `filter_by_population_range(species_data, min_population, max_population)` filters species by population range
- `filter_by_habitat_type(species_data, habitat_type)` - filters species by habitat type
- `filter_by_sanctuary(species_data, sanctuary)` - filters species by sanctuary
- `find_species_with_keyword(species_data, keyword)` - finds species with keyword
- `update_species_population(species_data, species_id, new_population)` - updates a species' population
- `update_conservation_status(species_data, species_id, new_status)` - updates conservation status
- `add_species_threat(species_data, species_id, new_threat)` - adds a threat to a species
- `merge_species_data(existing_species, new_species)` - merges two species dictionaries
- `calculate_status_counts(species_data)` - counts species in each conservation status
- `calculate_total_population(species_data)` - calculates total population across all species
- `find_most_threatened_species(species_data)` - finds most threatened species
- `create_population_brackets(species_data)` - groups species into population brackets

3. Display Functions:

- `get_formatted_species(sid, species)` - formats a species for display
- `display_data(data, data_type)` - displays species or other data types

4. Program Control Functions:

- display_menu() -> None
- `main()` - main program function

5. DETAILED FUNCTION IMPLEMENTATION STRUCTURE

5.1 Data Initialization Functions

1. Write a Python function to initialize wildlife conservation system data. Define: `initialize_data()`

The function should:

- Create predefined species dictionary with **exact variable name** `species_data`
- Create new species dictionary with **exact variable name** `new_species`
- Return tuple containing `(species_data, new_species)`

Must include these exact predefined species:
species_data = { "SP001": { "name": "Bengal Tiger", "scientific_name": "Panthera tigris tigris", "conservation_status": "Endangered", "population": 3500, "habitat_type": "Forest", "sanctuaries": ["Sundarbans", "Jim Corbett", "Bandhavgarh"], "threats": ["Poaching", "Habitat Loss", "Human Conflict"] }, "SP002": { "name": "Asian Elephant", "scientific_name": "Elephas maximus", "conservation_status": "Endangered", "population": 27000, "habitat_type": "Forest", "sanctuaries": ["Periyar", "Nagarhole", "Jim Corbett"], "threats": ["Habitat Loss", "Human Conflict", "Poaching"] } # Continue with SP003, SP004, SP005...}

-

Must include these exact new species:
new_species = { "NS001": { "name": "Ganges River Dolphin", "scientific_name": "Platanista gangetica", "conservation_status": "Endangered", "population": 3500, "habitat_type": "Wetland", "sanctuaries": ["Vikramshila", "National Chambal", "Katerniaghat"], "threats": ["Water Pollution", "Fishing Nets", "Dams"] }, "NS002": {

"name": "Great Indian Bustard", "scientific_name": "Ardeotis nigriceps", "conservation_status": "Critically Endangered", "population": 150, "habitat_type": "Grassland", "sanctuaries": ["Desert National Park", "Kutch Bustard", "Rollapadu"], "threats": ["Habitat Loss", "Power Lines", "Predation"] }}

-
- Dictionary structure: Each species must contain exactly these keys: "name", "scientific_name", "conservation_status", "population", "habitat_type", "sanctuaries", "threats"
- Data types: strings for text fields, integer for population, lists for sanctuaries and threats

5.2 Dictionary Filtering Functions

2. Write a Python function to filter species by conservation status. Define: `filter_by_conservation_status(species_data, status)`

The function should:

- Accept parameters: `species_data` (dict), `status` (string)
- Validate inputs: Check for None values and raise ValueError if found
- **Use dictionary comprehension**: `{sid: species for sid, species in species_data.items() if species["conservation_status"] == status}`
- Valid conservation statuses: "Least Concern", "Near Threatened", "Vulnerable", "Endangered", "Critically Endangered"
- Return filtered dictionary containing only species with matching conservation status
- Example: filter_by_conservation_status(species_data, "Endangered") returns all endangered species
- Preserve original dictionary structure and all fields for matched species

3. Write a Python function to filter species by population range. Define: `filter_by_population_range(species_data, min_population, max_population)`

The function should:

- Accept parameters: `species_data` (dict), `min_population` (int), `max_population` (int)
- Validate inputs: Check None values, negative minimum, and logical range order
- Input validation: `if min_population > max_population: raise ValueError("Minimum population cannot be greater than maximum population")`

- **Use dictionary comprehension**: `{sid: species for sid, species in species_data.items() if min_population <= species["population"] <= max_population}`
- Range logic: Include species where population falls within or equals the boundary values
- Return filtered dictionary with species matching population criteria
- Example: filter_by_population_range(species_data, 1000, 5000) returns species with populations between 1000-5000
- Handle edge cases: exact boundary matches, empty results for impossible ranges

4. Write a Python function to filter species by habitat type. Define: `filter_by_habitat_type(species_data, habitat_type)`

The function should:

- Accept parameters: `species_data` (dict), `habitat_type` (string)
- Validate inputs: Check for None values and raise ValueError if found
- **Use dictionary comprehension**: `{sid: species for sid, species in species_data.items() if species["habitat_type"] == habitat_type}`
- Valid habitat types: "Forest", "Grassland", "Wetland", "Mountain", "Desert"
- Return filtered dictionary containing only species with matching habitat type
- Case sensitivity: Exact string match required
- Example: filter_by_habitat_type(species_data, "Forest") returns all forest-dwelling species

5. Write a Python function to filter species by sanctuary. Define: `filter_by_sanctuary(species_data, sanctuary)`

The function should:

- Accept parameters: `species_data` (dict), `sanctuary` (string)
- Validate inputs: Check for None values and raise ValueError if found
- **Use dictionary comprehension with list membership**: `{sid: species for sid, species in species_data.items() if sanctuary in species["sanctuaries"]}`
- Search logic: Check if sanctuary name exists in the sanctuaries list for each species
- Return filtered dictionary containing species found in the specified sanctuary
- Case sensitivity: Exact string match required within sanctuary list
- Example: filter_by_sanctuary(species_data, "Jim Corbett") returns all species found in Jim Corbett sanctuary

6. Write a Python function to find species with keyword search. Define: `find_species_with_keyword(species_data, keyword)`

The function should:

- Accept parameters: `species_data` (dict), `keyword` (string)
- Validate inputs: Check for None values and raise ValueError if found
- Convert keyword to lowercase: `keyword = keyword.lower()`

Use dictionary comprehension with multiple field search:
{sid: species for sid, species in species_data.items() if keyword in species["name"].lower() or keyword in species["scientific_name"].lower() or any(keyword in threat.lower() for threat in species["threats"])}

-
- Search fields: name, scientific_name, and all items in threats list
- Case insensitive: Convert all text to lowercase for comparison
- Return filtered dictionary containing species matching keyword in any searched field
- Use any() function: For checking keyword presence in threats list
- Example: find_species_with_keyword(species_data, "poach") returns species with "poaching" in their threats

5.3 Dictionary Update Functions

7. Write a Python function to update species population. Define:
`update_species_population(species_data, species_id, new_population)`

The function should:

- Accept parameters: `species_data` (dict), `species_id` (string), `new_population` (int)
- Validate inputs: Check None values, negative population, and species existence
- Species validation: `if species_id not in species_data: raise ValueError(f"Species ID {species_id} not found")`
- **Create copy of original dictionary**: `updated_species_data = species_data.copy()`
- **Use dictionary unpacking to update**: `updated_species_data[species_id] = {**updated_species_data[species_id], "population": new_population}`
- Preserve immutability: Do not modify original dictionary
- Return updated dictionary with new population value
- Example: update_species_population(species_data, "SP001", 4000) returns new dictionary with SP001 population set to 4000
- Data integrity: All other fields remain unchanged

8. Write a Python function to update conservation status. Define:
`update_conservation_status(species_data, species_id, new_status)`

The function should:

- Accept parameters: `species_data` (dict), `species_id` (string), `new_status` (string)
- Validate inputs: Check None values, valid status, and species existence
- Status validation: `valid_statuses = ["Least Concern", "Near Threatened", "Vulnerable", "Endangered", "Critically Endangered"]`
- Status check: `if new_status not in valid_statuses: raise ValueError("Invalid conservation status")`
- **Create copy and use dictionary unpacking**: `updated_species_data[species_id] = {**updated_species_data[species_id], "conservation_status": new_status}`
- Preserve immutability: Original dictionary remains unchanged
- Return updated dictionary with new conservation status
- Example: update_conservation_status(species_data, "SP003", "Endangered") updates SP003 status to Endangered

9. Write a Python function to add species threat. Define: `add_species_threat(species_data, species_id, new_threat)`

The function should:

- Accept parameters: `species_data` (dict), `species_id` (string), `new_threat` (string)
- Validate inputs: Check None values, empty string, and species existence
- Empty threat check: `if new_threat is None or new_threat == "": raise ValueError("New threat cannot be None or empty")`
- **Create copy of dictionary**: `updated_species_data = species_data.copy()`
- **Check for duplicate threats**: `if new_threat not in updated_species_data[species_id]["threats"]:`
- **Create copy of threats list**: `updated_threats = updated_species_data[species_id]["threats"].copy()`
- **Append new threat**: `updated_threats.append(new_threat)`
- **Use dictionary unpacking**: `updated_species_data[species_id] = {**updated_species_data[species_id], "threats": updated_threats}`
- Prevent duplicates: Only add threat if it doesn't already exist
- Preserve immutability: Create new lists and dictionaries
- Return updated dictionary with new threat added
- Example: add_species_threat(species_data, "SP001", "Disease") adds Disease to SP001's threats list

5.4 Dictionary Merging and Transformation Functions

10. Write a Python function to merge species data dictionaries. Define: `merge_species_data(existing_species, new_species)`

The function should:

- Accept parameters: `existing_species` (dict), `new_species` (dict)
- Validate inputs: Check for None values and raise ValueError if found
- **Create copy of existing data**: `merged_species_data = existing_species.copy()`

Use dictionary unpacking with transformation:
for sid, species in new_species.items(): merged_species_data[sid] = {**species, "newly_added": True}

- Add transformation flag: Each new species gets "newly_added": True field
- Preserve original dictionaries: Do not modify input dictionaries
- Handle conflicts: New species with same ID will overwrite existing ones
- Return merged dictionary containing all species with new arrivals marked
- Example: merge_species_data(species_data, new_species) combines dictionaries and marks new species

5.5 Dictionary Analysis Functions

11. Write a Python function to calculate status counts. Define: `calculate_status_counts(species_data)`

The function should:

- Accept parameter: `species_data` (dict)
- Validate input: Check for None values and raise ValueError if found
- **Initialize empty counts dictionary**: `status_counts = {}`

Use dictionary iteration:
for species in species_data.values(): status = species["conservation_status"] if status in status_counts: status_counts[status] += 1 else: status_counts[status] = 1

- Alternative approach: Use dictionary.get() method for cleaner code
- Count each occurrence: Increment counter for each conservation status
- Return dictionary with status names as keys and counts as values
- Example: calculate_status_counts(species_data) returns {"Endangered": 2, "Vulnerable": 2, "Critically Endangered": 1}

12. Write a Python function to calculate total population. Define: `calculate_total_population(species_data)`

The function should:

- Accept parameter: `species_data` (dict)
- Validate input: Check for None values and raise ValueError if found
- **Use sum() with generator expression**: `return sum(species["population"] for species in species_data.values())`
- Generator expression: Efficient way to iterate through all population values
- Return integer representing total population across all species
- Example: calculate_total_population(species_data) returns sum of all species populations
- Handle empty dictionary: Returns 0 for empty species data

13. Write a Python function to find most threatened species. Define: `find_most_threatened_species(species_data)`

The function should:

- Accept parameter: `species_data` (dict)
- Validate input: Check for None, empty dictionary, and raise ValueError if found

Define status order dictionary:
status_order = { "Least Concern": 0, "Near Threatened": 1, "Vulnerable": 2, "Endangered": 3, "Critically Endangered": 4}

-

Create threat assessment function:
def threat_key(item): sid, species = item return (status_order[species["conservation_status"]], -species["population"])

-
- **Use max() with custom key**: `return max(species_data.items(), key=threat_key)`
- Sorting logic: First by conservation status (higher index = more threatened), then by population (negative for ascending)
- Return tuple: (species_id, species_data) of most threatened species
- Example: find_most_threatened_species(species_data) returns the species with highest threat level

14. Write a Python function to create population brackets. Define: `create_population_brackets(species_data)`

The function should:

- Accept parameter: `species_data` (dict)
- Validate input: Check for None values and raise ValueError if found

Initialize brackets dictionary:
population_brackets = { "critical": [], # 0-500 "endangered": [], # 501-5000 "vulnerable": [], # 5001-20000 "stable": [] # 20001+}

Categorize each species:
for sid, species in species_data.items(): population = species["population"] if population <= 500: population_brackets["critical"].append(sid) elif population <= 5000: population_brackets["endangered"].append(sid) elif population <= 20000: population_brackets["vulnerable"].append(sid) else: population_brackets["stable"].append(sid)

- Population ranges: critical (0-500), endangered (501-5000), vulnerable (5001-20000), stable (20001+)
- Return dictionary with bracket names as keys and lists of species IDs as values
- Example: create_population_brackets(species_data) groups species by population size

5.6 Display Functions

15. Write a Python function to format species for display. Define: `get_formatted_species(sid, species)`

The function should:

- Accept parameters: `sid` (string), `species` (dict)
- Validate input: Check for None values and raise ValueError if found
- **Format population with thousands separator**: `formatted_population = f"{species['population']:,}"`

Format lists as comma-separated strings:
sanctuaries = ", ".join(species["sanctuaries"])threats = ", ".join(species["threats"])

-
- **Check for newly added flag**: `newly_added = " [NEW]" if species.get("newly_added", False) else ""`

Return formatted string:
return (f"{sid} | {species['name']}{newly_added} ({species['scientific_name']}) | " f"{species['conservation_status']} | Population: {formatted_population} | " f"Habitat: {species['habitat_type']} | Sanctuaries: {sanctuaries} | " f"Threats: {threats}")

-
- String formatting: Use f-strings for clean, readable output
- Visual indicators: Add [NEW] tag for newly added species
- Consistent format: All species displayed with same structure
- Example output: "SP001 | Bengal Tiger (Panthera tigris tigris) | Endangered | Population: 3,500 | Habitat: Forest | Sanctuaries: Sundarbans, Jim Corbett, Bandhavgarh | Threats: Poaching, Habitat Loss, Human Conflict"

16. Write a Python function to display different data types. Define: `display_data(data, data_type)`

The function should:

- Accept parameters: `data` (varies), `data_type` (string)
- Handle None data: Print "No data to display." and return early

Implement conditional display logic:

```
 if data_type == "species" or data_type == "filtered":
   header = "\nCurrent Species Data:" if data_type == "species" else "\nFiltered Results:"
print(header)
for sid, species in data.items():
                print(get_formatted_species(sid, species))elif data_type == "status_counts":
        print("\nConservation Status Counts:")
for status, count in data.items():
        print(f"{status}: {count} species")# Continue with other data types...
```

- Data type handling: Different display formats for different data types
- Reuse formatting: Call get_formatted_species() for consistent species display
- Clear headers: Show appropriate headers for each data type
- Handle empty data: Show "No species to display." for empty results

5.7 Main Program Implementation

17. Write a Python main program for wildlife conservation system. Define: `main()`

The program should:

- **Initialize system data**: Call initialize_data() and assign to variables `species_data`, `new_species`
- **Display welcome information**: Show system title, total species count, and available conservation statuses
- **Implement menu-driven interface**: Create a loop with numbered options for different operations
- **Handle user input**: Get user choices and call appropriate functions based on selection

- **Provide error handling**: Use try-catch blocks to handle invalid inputs and function errors
- **Exit gracefully**: Include option to terminate the program and cleanup

Menu System Requirements:

- **Option 1**: View all species data using display function
- **Option 2**: Filter species by various criteria (status, population, habitat, sanctuary, keyword)
- **Option 3**: Update species data (population, conservation status, threats)
- **Option 4**: Add new species by merging new_species dictionary
- **Option 5**: View conservation statistics (status counts, population totals, brackets)
- **Option 0**: Exit the program

5.8 Implementation Requirements

Dictionary Operations Usage Requirements:

- **Dictionary comprehension**: Used in all filter functions for efficient filtering
- **Dictionary unpacking (**): Used in update functions to create modified copies
- **Dictionary methods (.items(), .values(), .keys())**: Used throughout for iteration and access
- **Dictionary merging**: Used in merge_species_data() for combining dictionaries
- **Dictionary copying (.copy())**: Used to preserve immutability in update functions
- **Dictionary transformation**: Used to add flags and modify data during merging

Data Structure Requirements:

- **Species dictionary structure**: Must contain exactly these keys: "name", "scientific_name", "conservation_status", "population", "habitat_type", "sanctuaries", "threats"
- **Dictionary variable names**: Must use exact names: `species_data`, `new_species`
- **Parameter names**: Must use exact parameter names in function definitions
- **Return values**: Functions must return appropriate data types (dict, tuple, int, or string)

6. EXECUTION STEPS TO FOLLOW:

1. Run the program

2. View the main menu

3. Select operations:

- Option 1: View Species Data
- Option 2: Filter Species
- Option 3: Update Species Data
- Option 4: Add New Species
- Option 5: View Conservation Statistics
- Option 0: Exit

4. Perform operations on the species data

5. View results after each operation

6. Exit program when finished

Execution Steps to Follow:

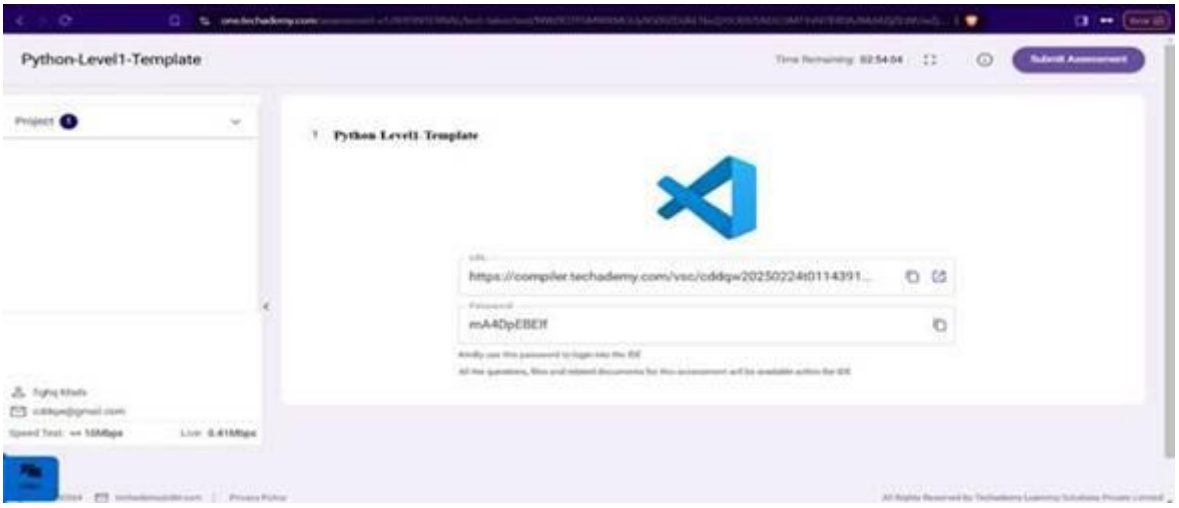
- All actions like build, compile, running application, running test cases will be through Command Terminal.
- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal
- This editor Auto Saves the code
- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) .
- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
- To launch application: python3 filename.py
- To run Test cases: python3 -m unittest

Screen shot to run the program

To run the application

python3 filename.py

To run the testcase python3 -m unittest



● Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.