Emilia Code Fix

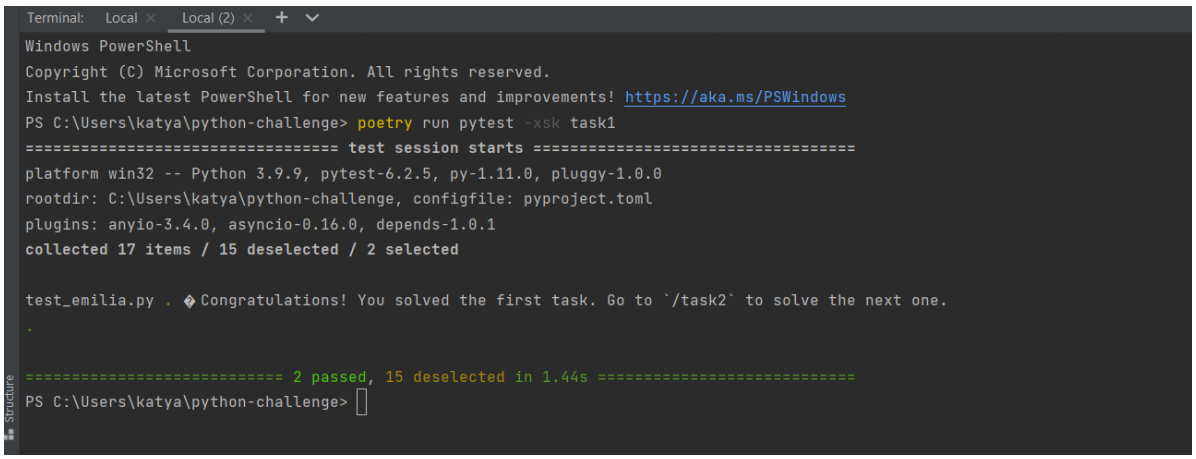
Task-1:

"🤷: The first step of a meaningful conversation is a friendly greeting. Therefore, your first task will be to improve our greeting functionality at `/task1/greet/{name}`. Currently, this route is only available in English GB. But not all of our customers speak English. Could you add a query parameter `language` to also support German DE and Spanish ES? Valid values should be `de`, `en`, and `es`! In case the user passes a different language, we should respond, that we don't speak it. Just go to `emilia.py` and implement the requested changes. The exact requirements for all tasks are defined by the test in `test_emilia.py`. Run 'pytest -xsk task1' to verify you solved this task correctly ✓.\n"

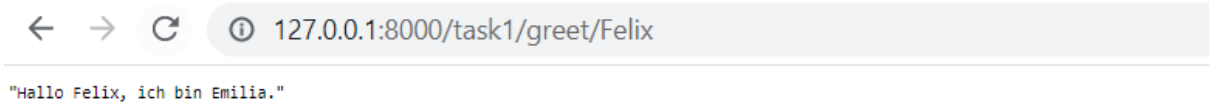
Task Solution:

Now in the first task, Emilia greets the users in 3 available languages so the code is added using the 'if-else' statements, where a return statement is given for each of the languages selected by the user and also there is a default case.

The code passes all the test cases given, using the command `poetry run pytest -xsk task1`, as the code is fixed for the `emila.py` python file, the output is given as:



The Task-1 browser output for the FastAPI application is given as,



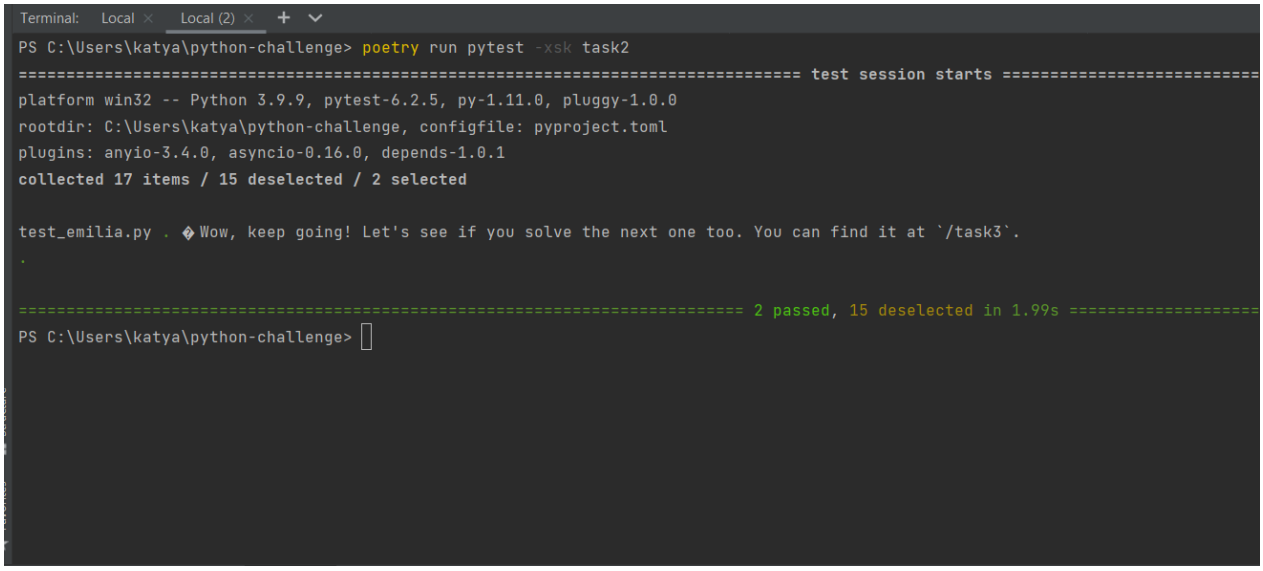
Task-2:

"🤷: Sometimes best practices can be annoying. Python uses 'snake_case' 🐍 for variables names, but from a typical API you might expect keys in 'camelCase' 🐫. Can you write a 'camelize' function that transforms a snake_case-string to a camelCase-string? The boilerplate code is already there! This function will be exposed at `/task2/camelize` and expects a JSON object as payload.\n"

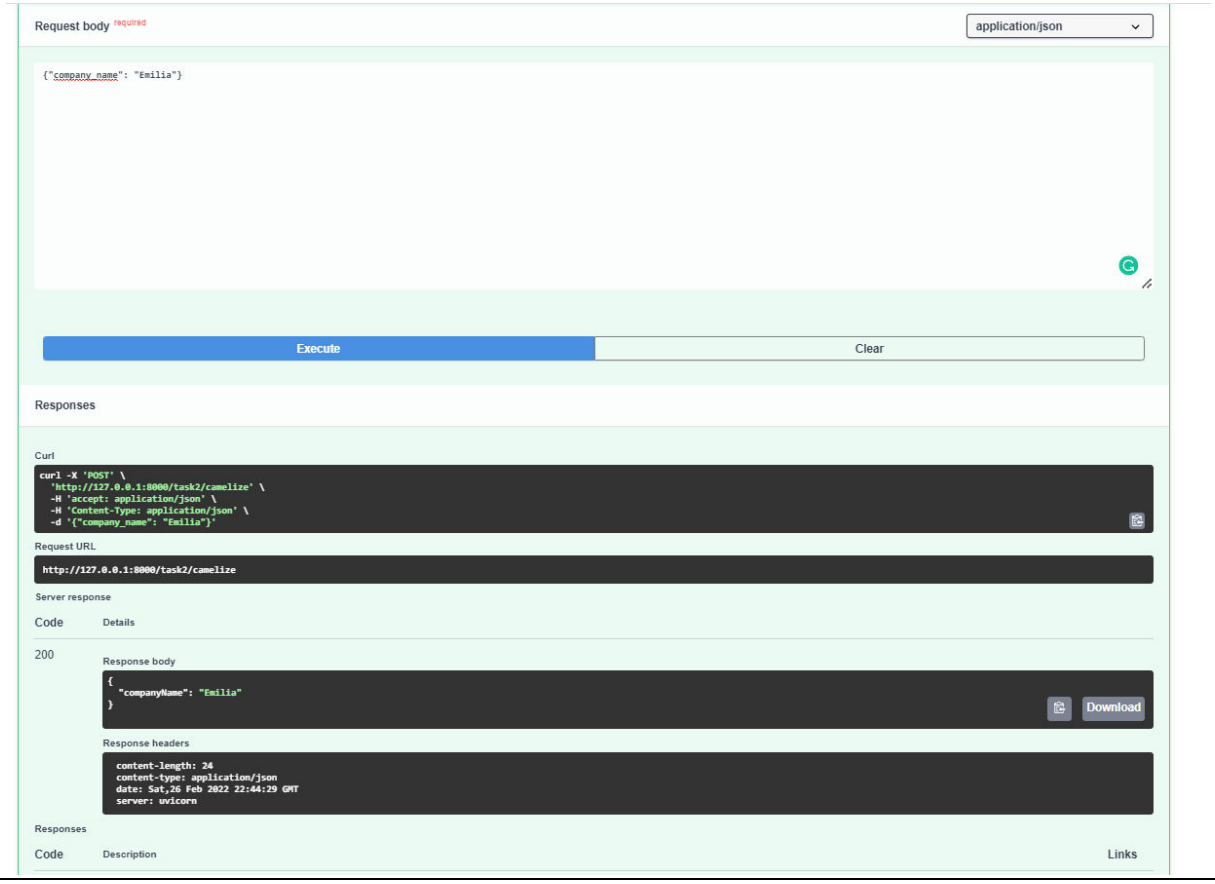
Task Solution:

In this task the input string in `snake_case` format changes to `camelCase`. To this the approach used is, splitting the string over the delimiter underscore and then joining the string by Capitalizing the starting letter of the word after the delimiter.

The code passes all the test cases given, using the command `poetry run pytest -xsk task2`, as the code is fixed for the 'emila.py' python file, the output is given as:



The task-2 output for the FastAPI application is given as,



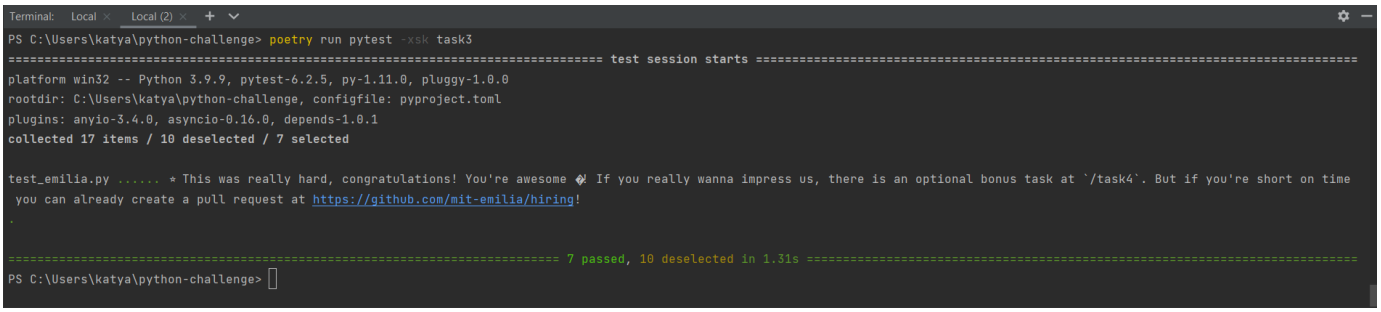
Task-3:

"🤳: Our goal is to increase the quality of life of elderly people by encouraging them to have more contact with other people through a rich and varied daily schedule ✅. Therefore we must understand the intentions of our users. For example, a user might want to call a family member or wants to set a reminder to have some tea with an old friend. We have different handlers to respond to the various user actions. Unfortunately, our code is broken □ as it uses a random handler for every incoming action. On top of that our responses are very rude 🖕 at the moment. Therefore, the first part of your task is to implement an intent recognition mechanism, which forwards the incoming actions to the right action handler. Afterward, you need to implement these action handlers. Make sure the responses are friendlier than they are now □. As always the exact requirements are set by the tests in `/test_emilia.py`. The number of tests is slightly higher than in the previous tasks, but don't let that discourage you!\n"

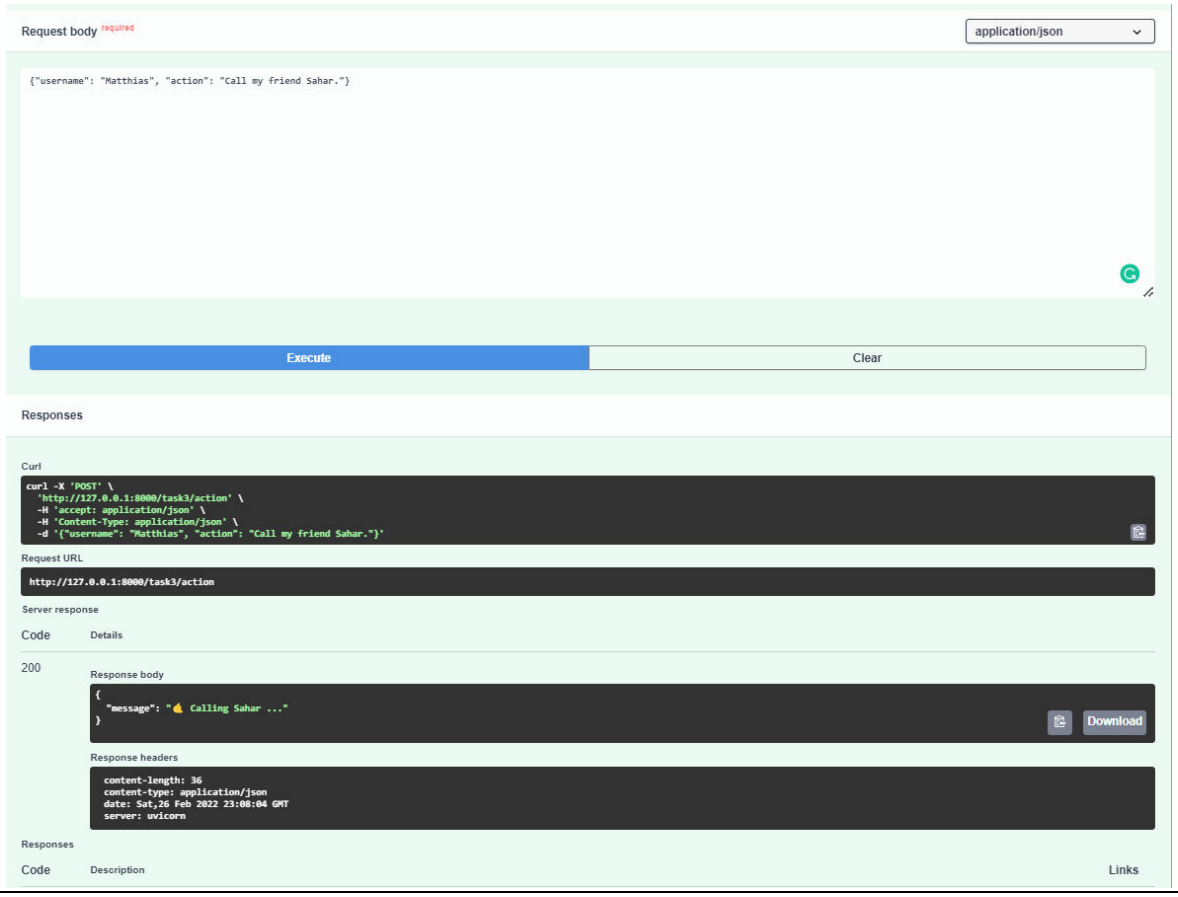
Task Solution:

Now in this task, the user actions are handled by Emilia, these actions can be related to calling, reminder, timer, and unknown actions. In each action, the action request is passed to which a message response is received as output. Here the input calls are managed by the handler as there are multiple functions under a single endpoint.

The code passes all the test cases given, using the command `poetry run pytest -xsk task3`, as the code is fixed for the `emila.py` python file, the output is given as:



The task-3 output for the FastAPI application is given as,



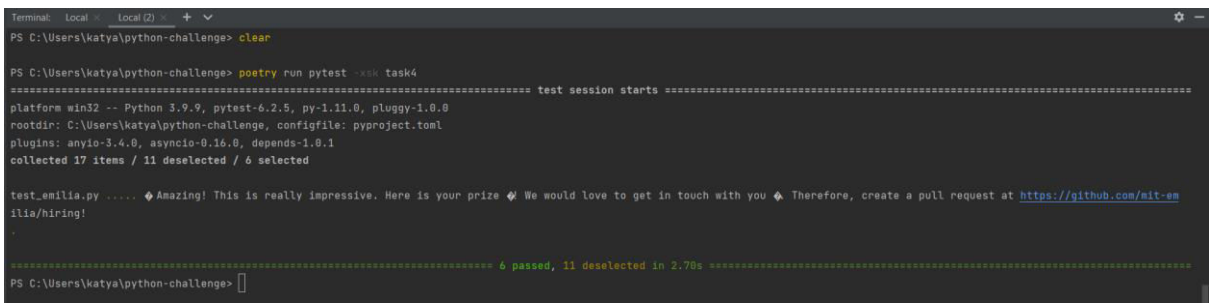
Task-4:

"(Bonus) : In our last task we have two users Stefan and Felix. Both of them have a secret stored in our database . Currently, our application is not very secure! An imposter could easily switch out the `username` field to pretend to be somebody else. Let's change that! Ideally, a secret should be accessible to the user it belongs to. Therefore if a user requests a secret from `/users/{username}/secret` route, we should verify that the user is logged in and that the `username` matches. For the login process, we created some boilerplate code and a (not fully working) login route at `/task4/token`. Your task is to implement the user verification logic and to make sure to return the right HTTP status in case a user is not authenticated or authorized. As always, check if you solved the task correctly by running 'pytest -xsk task4'. Have fun, you're almost there !\n"

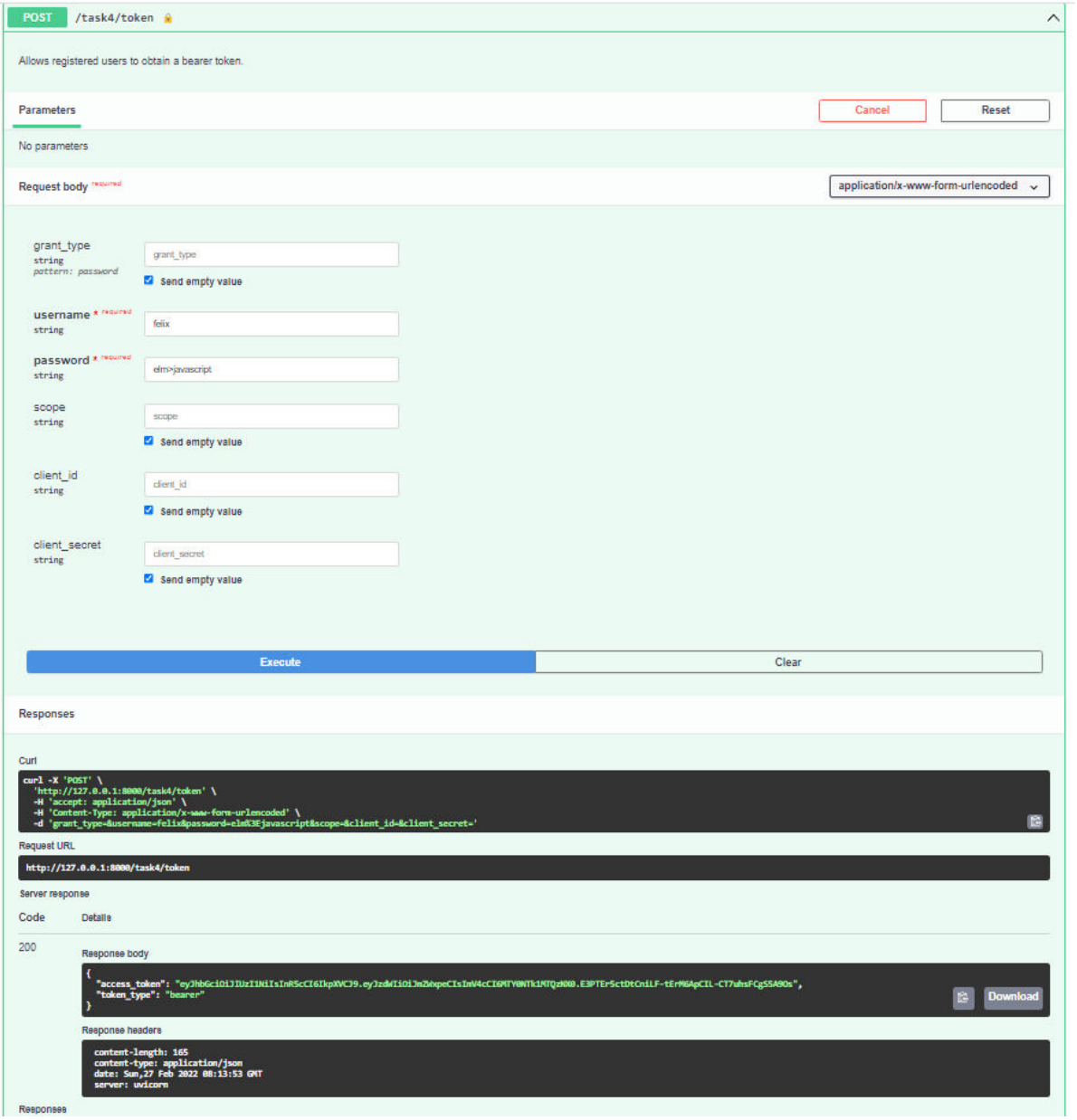
Task Solution:

The code is fixed by modifying the login() function functionality where now the password is also checked using the `verify_password` functionality and only those users that are registered and have valid login details can login to the system else the Emilia system raises an error. Now the system also verifies that the current_user and login_user both are the same and can only access their own secrets to which the function `get_current_user()` and `read_user_secret()` are modified in the code.

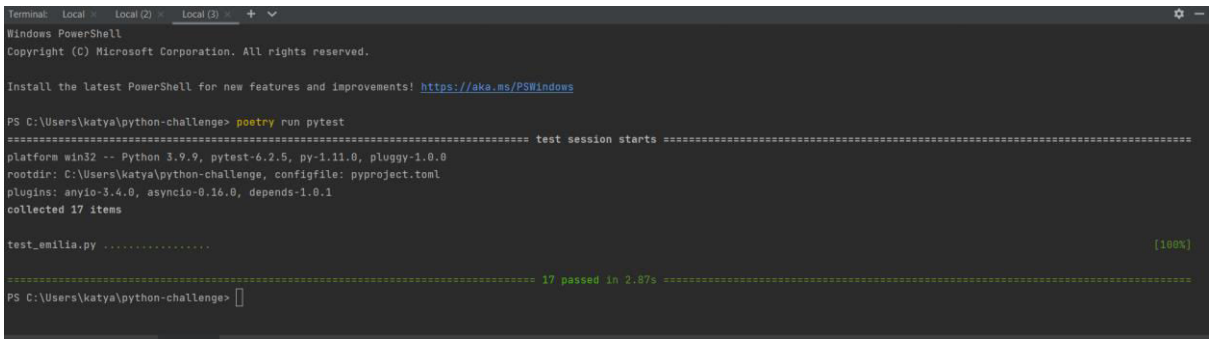
The code passes all the test cases given, using the command `poetry run pytest -xsk task4`, as the code is fixed in the `emila.py` python file, the output is given as:



The task-4 output for the FastAPI application is given as,



Now all the 4 tasks are done. So, the test cases are checked for the complete code by running the command `poetry run pytest` so this runs all the test cases for the Emilia and the result for this is,



All the test cases are passed for the Code. So, the code is fixed for Emilia. (Also added comments to the code for better understanding of the code)
