SOFTWARE ENGINEERING PROJECT

Milestone V

GROUP 11

JEMMA MARIYA GEORGE
Roll No. 21f1001937

BARUN KUMAR SINHA
Roll No. 21f1002021

VIDUSHI TALWAR
Roll No. 21f1004809

Test Driven Development [TDD]

Test Driven Development is used especially in Agile Processes.

Herein, we first write the tests for the functionality that we wish to implement. Thereafter we design and develop that functionality.

The first thing we do is express the requirement / feature / user story in the form of a test and then we run this created test. And since we haven't yet implemented the feature in question, the test will fail. So we deliberately create a test and make it fail. Next, our goal is to somehow make this test pass & to accomplish this, we create the minimum code to meet the needs of the test.

Finally we refactor the code, i.e. we try to improve the quality, make the code more modular and elegant while still satisfying the test case scenarios defined.

Thus the mantra for Test Driven Development can be summarized as:

- **RED:** Write a Test and See it Fail
- **GREEN:** Write code and See it Pass
- **REFACTOR:** Make the code better

Test Cases

Note: The test cases have been designed in the following format:

[Page Being tested,
Inputs,
Expected Output,
Actual Output,
Result – Success/Fail]

Dashboard – Load Tickets API:

- ## Testing for server error when the database has not been created
  [User Dashboard,
  Category – “Python”,
  {“message”: “Internal Server Error”},
  {“message”: “Internal Server Error”},
  Success]
- ## testing for tickets when table not creating
  [User Dashboard,
  Category – “Python”,
  {“message”: “Tickets do not exist for the given category”},
  {“message”: “Internal Server Error”},
  Fail]
- ## Testing for tickets when empty table is created
  [User Dashboard,
  Category – “Python”,
  {“message”: “Tickets do not exist for the given category”},
  {“message”: “Tickets do not exist for the given category”},
  Success]
- ## Testing for tickets when table has data but query hasn’t been defined
  [User Dashboard,
  Category – “Python”,
  [{'Ticket_id': 'Python/1', 'Created_by': 1, 'Solved_by': 2, 'Category': 'Python', 'Query': 'There is an error in the GA 4 Q7.. Could you please look into it?', 'Solution': 'Dear Student, the quetsion has been updated. Please check.', 'Created_at': '03/23/2023 @ 4:31am', 'Solved_at': '03/23/2023 @ 10:11am', 'IsResolved': 'True', 'AddFAQ': 'False', 'Pin': 'False', 'Report': 'False', 'Likes': 3}],
  {“message”: “Internal Server Error”},
  Fail]

- ## Testing for tickets when table has data & query has been defined
  [User Dashboard,
  Category – "Python",
  [{'Ticket_id': 'Python/1', 'Created_by': 1, 'Solved_by': 2, 'Category': 'Python', 'Query': 'There is an error in the GA 4 Q7.. Could you please look into it?', 'Solution': 'Dear Student, the quetsion has been updated. Please check.', 'Created_at': '03/23/2023 @ 4:31am', 'Solved_at': '03/23/2023 @ 10:11am', 'IsResolved': 'True', 'AddFAQ': 'False', 'Pin': 'False', 'Report': 'False', 'Likes': 3}],
  [{'Ticket_id': 'Python/1', 'Created_by': 1, 'Solved_by': 2, 'Category': 'Python', 'Query': 'There is an error in the GA 4 Q7.. Could you please look into it?', 'Solution': 'Dear Student, the quetsion has been updated. Please check.', 'Created_at': '03/23/2023 @ 4:31am', 'Solved_at': '03/23/2023 @ 10:11am', 'IsResolved': 'True', 'AddFAQ': 'False', 'Pin': 'False', 'Report': 'False', 'Likes': 3}],
  Success]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help display the tickets created in the various categories on the user dashboard:

```
class DashboardApi(Resource):

  def get(self, cat):
    dashboard = db.session.execute(f"SELECT * FROM ticket WHERE Category = '{cat}'").fetchall()
    dash = list(dashboard)
    dashboard = [dict(row) for row in dashboard]

    if dash == [] :
      abort(404, message="Tickets do not exist for the given category")

    return dashboard

api.add_resource(DashboardApi, "/dashboard/tickets/<cat>")
```

Admin - Select Tickets for FAQ API:

- ## Testing for server error when the ticket with said id does not exist
  [FAQ Document,
  Ticket_id – "Python91",
  {"message": "Query with given id doesn't exist"},
  {"message": "Internal Server Error"},
  Fail]
- ## Testing for server error when the ticket with said id does not exist after defining query
  [FAQ Document,

Ticket_id – “Python91”,
{“message”: “Query with given id doesn't exist”},
{“message”: “Query with given id doesn't exist”},
Success]

- ## Testing for server error when the query with said id does exist
  [FAQ Document,
  Ticket_id – “Python1”,
  {'Ticket_id': 'Python1', 'status': 'Query AddFAQ status updated!'},
  {“message”: “Internal Server Error”},
  Fail]
- ## Testing for server error when the ticket with said id does exist after defining the query
  [FAQ Document,
  Ticket_id – “Python1”,
  {'Ticket_id': 'Python1', 'status': 'Query AddFAQ status updated!'},
  {'Ticket_id': 'Python1', 'status': 'Query AddFAQ status updated!'},
  Success]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help display the admins select the tickets to be added to the FAQ document:

```
add_faq = reqparse.RequestParser(bundle_errors=True)
add_faq.add_argument("Ticket_id", type=str, required=True, help="No ticket id provided")

class AdminApi(Resource):

  def put(self):

    args = add_faq.parse_args()
    Ticket_id = args.get("Ticket_id")
    tic = db.session.execute(f"SELECT * FROM ticket WHERE Ticket_id = '{Ticket_id}'")
    tic = [dict(row) for row in tic]

    if tic == []:
      abort(404, message="Query with given id doesn't exist")

    tru = True
    db.session.execute(f"UPDATE ticket SET AddFAQ = '{tru}' WHERE Ticket_id = '{Ticket_id}'")
    db.session.commit()

    response = { "Ticket_id": Ticket_id, "status": "Query AddFAQ status updated!" }

    return response

api.add_resource(AdminApi, "/admin/add_ticket_to_faq")
```

Admin - Delete reported tickets API:

- ## Testing for server error when the ticket with said id does not exist
  [User Dashboard,
  Ticket_id – “Python91”,
  {“message”: “Query with given id doesn't exist”},
  {“message”: “Internal Server Error”},
  Fail]
- ## Testing for server error when the ticket with said id does not exist after defining query
  [User Dashboard,
  Ticket_id – “Python91”,
  {“message”: “Query with given id doesn't exist”},
  {“message”: “Query with given id doesn't exist”},
  Success]
- ## Testing for server error when the ticket with said id does exist
  [User Dashboard,
  Ticket_id – “English7”,
  {'Ticket_id': 'English7', 'status': 'Query Deleted Successfully!'},
  {“message”: “Internal Server Error”},
  Fail]
- ## Testing for server error when the ticket with said id does exist but is not reported after defining query
  [User Dashboard,
  Ticket_id – “English7”,
  {'message': "Operation not allowed as Query hasn't been reported yet."},
  {'Ticket_id': 'English7', 'status': 'Query Deleted Successfully!'},
  Fail]
- ## Testing for server error when the ticket with said id does exist but is not reported after defining logic
  [User Dashboard,
  Ticket_id – “English7”,
  {'message': "Operation not allowed as Query hasn't been reported yet."},
  {'message': "Operation not allowed as Query hasn't been reported yet."},
  Success]
- ## Testing for server error when the ticket with said id does exist and has been reported after defining query
  [User Dashboard,
  Ticket_id – “English7”,
  {'Ticket_id': 'English7', 'status': 'Query Deleted Successfully!'},
  {'Ticket_id': 'English7', 'status': 'Query Deleted Successfully!'},
  Success]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help display the admins delete the tickets reported by the Support Staff:

```
del_ticket = reqparse.RequestParser(bundle_errors=True)
del_ticket.add_argument("Ticket_id", type=str, required=True, help="No ticket id provided")

class Delete_ticket(Resource):

    def delete(self):

        args = del_ticket.parse_args()
        Ticket_id = args.get("Ticket_id")
        tru = True
        tic = db.session.execute(f"SELECT * FROM ticket WHERE Ticket_id = '{Ticket_id}'")

        tic = [dict(row) for row in tic]

        if tic == []:
            abort(404, message="Query with given id doesn't exist")
        elif tic[0]['Report'] != 'True':
            abort(405, message="Operation not allowed as Query hasn't been reported yet.")

        db.session.execute(f"DELETE FROM ticket WHERE Ticket_id = '{Ticket_id}'")
        db.session.commit()

        response = { "Ticket_id": Ticket_id, "status": "Query Deleted Successfully!" }

        return response

api.add_resource(Delete_ticket, "/admin/delete_ticket")
```

FAQ – Adding Selected queries to doc API:

- ## Testing for server error when the ticket with said id already exists in the doc
  [FAQ Document,
  Ticket_id – "Python91",
  {"message": "Query already exists in FAQs!"},
  {"message": "Internal Server Error"},
  Fail]
- ## Testing for server error when the ticket with said id already exists in the doc after defining logic
  [FAQ Document,
  Ticket_id – "Python91",
  {"message": "Query already exists in FAQs!"},
  {"message": "Query already exists in FAQs!"},

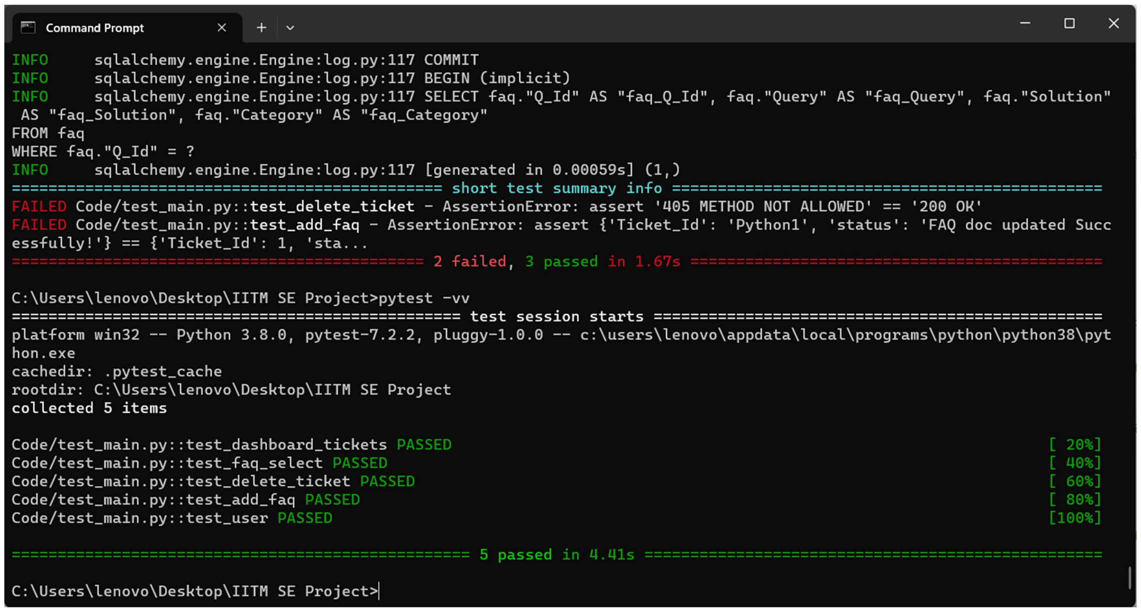
Success]

- ## Testing for server error when the query with said id does not exist in the doc
  [FAQ Document,
  Ticket_id – "Python1",
  {'Ticket_id': 'Python1', 'status': 'FAQ doc updated successfully!'},
  {"message": "Internal Server Error"},
  Fail]
- ## Testing for server error when the ticket with said id does not exist in the doc after defining the query
  [FAQ Document,
  Ticket_id – "Python1",
  {'Ticket_id': 'Python1', 'status': 'FAQ doc updated successfully!'},
  {'Ticket_id': 'Python1', 'status': 'FAQ doc updated successfully!'},
  Success]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help add the queries to the FAQ document:

```
create_faq = reqparse.RequestParser(bundle_errors=True)
create_faq.add_argument("Ticket_id", type=str, required=True, help="No ticket id provided")
create_faq.add_argument("Query", type=str, required=True, help="No query provided")
create_faq.add_argument("Solution", type=str, required=True, help="No solution provided")
create_faq.add_argument("Category", type=str, required=True, help="No category provided")

class FAQApi(Resource):

  def post(self):

    args = create_faq.parse_args()
    Ticket_id = args.get("Ticket_id")
    Query = args.get("Query")
    Solution = args.get("Solution")
    Category = args.get("Category")
    tic = db.session.execute(f"SELECT * FROM faq WHERE Query = '{Query}'")
    tic = [dict(row) for row in tic]

    if tic != []:
      abort(409, message="Query already exists in FAQs!")

    query_faq = faq(Query = Query, Solution = Solution, Category = Category)
    db.session.add(query_faq)
    db.session.commit()

    Q_Id = query_faq.Q_Id
    response = { "Ticket_Id" : Ticket_id, "status" : "FAQ doc updated Successfully!" }

    return response
```

api.add_resource(FAQApi, "/FAQ/add_ticket")



```
INFO     sqlalchemy.engine.Engine:log.py:117 COMMIT
INFO     sqlalchemy.engine.Engine:log.py:117 BEGIN (implicit)
INFO     sqlalchemy.engine.Engine:log.py:117 SELECT faq."Q_Id" AS "faq_Q_Id", faq."Query" AS "faq_Query", faq."Solution"
 AS "faq_Solution", faq."Category" AS "faq_Category"
FROM faq
WHERE faq."Q_Id" = ?
INFO     sqlalchemy.engine.Engine:log.py:117 [generated in 0.00059s] (1,)
=============================================== short test summary info ================================================
FAILED Code/test_main.py::test_delete_ticket - AssertionError: assert '405 METHOD NOT ALLOWED' == '200 OK'
FAILED Code/test_main.py::test_add_faq - AssertionError: assert {'Ticket_Id': 'Python1', 'status': 'FAQ doc updated Succ
essfully!'} == {'Ticket_Id': 1, 'sta...
============================================= 2 failed, 3 passed in 1.67s ==============================================

C:\Users\lenovo\Desktop\IITM SE Project>pytest -vv
================================================== test session starts =================================================
platform win32 -- Python 3.8.0, pytest-7.2.2, pluggy-1.0.0 -- c:\users\lenovo\appdata\local\programs\python\python38\pyt
hon.exe
cachedir: .pytest_cache
rootdir: C:\Users\lenovo\Desktop\IITM SE Project
collected 5 items

Code/test_main.py::test_dashboard_tickets PASSED                                                                  [ 20%]
Code/test_main.py::test_faq_select PASSED                                                                         [ 40%]
Code/test_main.py::test_delete_ticket PASSED                                                                      [ 60%]
Code/test_main.py::test_add_faq PASSED                                                                            [ 80%]
Code/test_main.py::test_user PASSED                                                                               [100%]

=================================================== 5 passed in 4.41s ==================================================

C:\Users\lenovo\Desktop\IITM SE Project>
```

Sign Up – Helping users access the software API:

- [
  Sign up Api being Tested,
  Inputs : { “user_name” : “John Doe”, “User_password” : “12345”, “User_email” : “abc@xyz.com”, “User_type” : ‘Student” },
  Expected outputs : [{ Status_code : 201, Description : “ User added successfully “, }, { Status_code : 409, Description : “ user with same name or email already exist “ }],
  Actual Output : { Status_code : 201, Description : “ User added successfully “, },
  Success
  ]
- [
  Sign up Page being Test,
  Inputs : { “user_name” : “John Doe”, “User_password” : “12345”, “User_email” : “abc@xyz.com” “User_type” : ‘Student” },
  Expected outputs : [{ Status_code : 201, Description : “ User added successfully “, }, { Status_code : 409, Description : “ user with same name or email already exist “ }],
  Actual Output : { Status_code : 409, Description : “user with same name or email already exist “, },
  Success
  ]

Resolve – Helping support team solve tickets API:

- [
  Resolve Ticket being Tested,
  Inputs : { "ticket_id": 1, "ticket_query": "Whats is syallbus for quiz 2", "ticket_answer": "week 1-8”, “User_type” : “support staff” },
  Expected outputs : [{ Status_code : 200, Message : “ticket resolve” }, { Status_code : 400 Message : “Bad request ! Only a support staff can resolve a ticket” }],
  Actual Output : { Status_code : 200, Message : “ticket resolve” },
  Success
  ]
- [
  Resolve Ticket being Tested,
  Inputs : { "ticket_id": 1, "ticket_query": "Whats is syallbus for quiz 2", "ticket_answer": "week 1-8”, “User_type” : “Student” },

Expected outputs : [{ Status_code : 200, Message : “ticket resolve” }, { Status_code : 400 Message : “Bad request ! Only a support staff can resolve a ticket” }],

Actual Output : { Status_code : 400 Message : “Bad request ! Only a support staff can resolve a ticket” },

Success

]

Create Ticket – Helping students create tickets API:

- ## Creating a valid user ticket
  [
  Create Ticket(create_ticket endpoint),
  {"Ticket_id": "T001", "Created_by": 1, "Solved_by": null, "Category": "Software", "Query": "How to install Python", "Created_at": "2022-03-23 10:00:00", "Solved_at": null, "IsResolved": false, "AddFAQ": false, "Pin": false, "Report": false, "Likes": 0},
  {"userid": 1, "ticket_id": "T001", "status": "Ticket Added Successfully!"},
  {"userid": 1, "ticket_id": "T001", "status": "Ticket Added Successfully!"}
  Success
  ]
- ## Creating an invalid user ticket
  [
  Create Ticket(create_ticket endpoint),
  {"Ticket_id": "T002", "Created_by": 1, "Solved_by": null, "Category": "Software", "Query": "How to install Python", "Created_at": "2022-03-23 10:00:00", "Solved_at": null, "IsResolved": false, "AddFAQ": false, "Pin": false, "Report": false, "Likes": 0},
  {"status": "Ticket with same query already exists"},
  {"status": "Ticket with same query already exists"},
  Fail
  ]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help students create tickets & put their queries forth the support team:

```
@app.route('/Student/create_ticket///', methods=['POST'])
def create_ticket(userid, category, query):
        # Extract ticket details from request body
        data = request.get_json()
        ticket_id = data['Ticket_id']
        created_by = data['Created_by']
        solved_by = data['Solved_by']
        created_at = data['Created_at']
        solved_at = data['Solved_at']
        is_resolved = data['IsResolved']
        add_faq = data['AddFAQ']
        pin = data['Pin']
        report = data['Report']
        likes = data.get('Likes', 0)
        # Check if ticket with same query already exists
        if Ticket.query.filter_by(category=category, query=query).first():
                return jsonify({'status': 'Ticket with same query already exists'}), 403
        # Create new ticket
```

```
ticket = Ticket(
            ticket_id=ticket_id,
            created_by=created_by,
            solved_by=solved_by,
            category=category,
            query=query,
            created_at=created_at,
            solved_at=solved_at,
            is_resolved=is_resolved,
            add_faq=add_faq,
            pin=pin,
            report=report,
            likes=likes
)
db.session.add(ticket)
db.session.commit()

# Return success message
return jsonify({'userid': userid, 'ticket_id': ticket_id, 'status': 'Ticket Added Successfully!'}),
200
```

Like Ticket – Helping students +1 / like existing tickets API:

- ## Like a valid ticket
  [
    Like Ticket(like_ticket endpoint),
    {"Likes": 1},
    {"userid": 1, "ticket_id": "T001", "status": "Ticket Liked Successfully!"},
    {"userid": 1, "ticket_id": "T001", "status": "Ticket Liked Successfully!"},
    Success
  ]
- ## Like an invalid ticket
  [
    Like Ticket(like_ticket endpoint),
    {"Likes": 1},
    {"status": "Query with given id doesn't exist!"},
    {"status": "Query with given id doesn't exist!"},
    Fail
  ]

As a result of creating & analyzing the test cases, the following API endpoint was designed to help students +1 the existing queries to help prevent duplications:

```
@app.route('/Student/like_ticket/', methods=['PUT'])
def like_ticket(ticket_id):
        # Get the request body
        request_body = request.get_json()
        # Get the likes from the request body
        likes = request_body.get('Likes')
        # Check if the ticket exists in the database
        ticket = Ticket.query.filter_by(ticket_id=ticket_id).first()
        if not ticket:
                return jsonify({'status': 'Query with given id doesn\'t exist!'}), 404
        # Update the likes of the ticket
        ticket.likes += likes

        # Commit the changes to the database
        db.session.commit()

        # Return a success message
        return jsonify({'userid': ticket.created_by, 'ticket_id': ticket.ticket_id, 'status': 'Ticket Liked
        Successfully!'}), 200
```