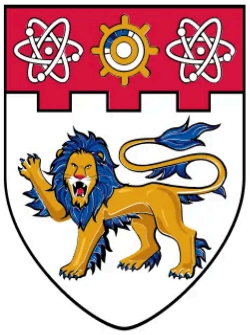


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 - Software Engineering
Lab 4 Deliverables

Lab Group	SCEB
Team	Team 1
Members	AKANKSHA MATHUR (U2323265C)
	HU HAN (U2320036H)
	PHUA NAOMI (U2422699H)
	KARTHIK RAJ S/O MOHAN(U2320917J)
	LI YUJIA (U2320574C)
	WANG KE (U2320276E)

Table of Contents

1. Black Box Testing……1

1.1 Profile Module……1

1.2 Equivalence Class and Boundary Value Testing……1

1.3 Test Cases and Results……2

2. White Box Testing……5

2.1 GetProfileByUserId……5

2.1.1 Control Flow Graph……5

2.1.2 Basic Path Testing……6

2.1.3 Test Cases and Results……6

2.2 ChangeUserTypeByUserId**……7**

2.2.1 Control Flow Graph……7

2.2.2 Basic Path Testing……8

2.2.3 Test Cases and Results……8

1. Black Box Testing

1.1 Control class to test - Profile Module

The Profile Control Class is responsible for managing the profile within the application including viewing, updating personal information, searching user records, and changing user types with restrictions. When a user logs in, their information, like email and password, will be collected and validated before they can view their profile. The user profile consists of the user's name and the type of restriction.

For update operations, a user accesses their profile to update their personal details, which is then validated by the controller to maintain data integrity. The controller ensures correct formatting and security monitoring, particularly of sensitive information, prior to storing the user-specific information in the database. Apart from profile updates, for actions handled by Administrators and Moderators, the controller imposes strict role-based access restrictions such as updating the type of user restriction.

1.2 Equivalence Class and Boundary Value Testing

Equivalence Class Testing

Equivalence Class Testing is a black-box testing technique that divides the input domain of a program into classes or groups of data from which test cases can be derived. The fundamental concept is that if one test case in an equivalence class detects a defect, other test cases in the same class would likely find the same defect.

User Types:

- Admin users (valid administrator privileges)
- MODERATOR users (valid moderator privileges)
- NORMAL users (standard user privileges)
- RESTRICTED users (limited privileges)
- Unauthenticated users (no privileges)
- Invalid user types (non-existent user types)

Target User Types:

- Target is ADMIN
- Target is MODERATOR
- Target is NORMAL
- Target is RESTRICTED
- Target does not exist

New Type:

- New type is ADMIN
- New type is MODERATOR
- New type is NORMAL
- New type is RESTRICTED

- New type is invalid/non-existent

User ID:
Valid: Valid existing user ID
Invalid: Valid format but non-existent user ID; Invalid format user ID; Empty user ID; Self user ID (current user)

Profile Data:
Valid: Valid profile data (all required fields present)
Invalid: Missing required fields; Invalid data types; Excessive data size; Empty profile data

Boundary Value Testing

Boundary Value Testing focuses on testing at the boundaries of input domains, based on the observation that errors tend to occur at the boundaries of input ranges rather than in the center.

User Type Boundaries:
Transition from NORMAL TO RESTRICTED; Transition from RESTRICTED to NORMAL; Transition from MODERATOR to ADMIN; Transition from NORMAL to MODERATOR;

User ID Boundaries:
- Valid: Minimum valid length for user ID; Maximum valid length for user ID;
- Invalid: Exceed maximum length;

Profile Data Boundaries:
- Valid: Minimum valid length for profile ID; Minimum valid length for profile ID;
- Invalid: Exceed maximum length;

1.3 Test Cases and Results

Test Case ID	Action	Input	Expected output	Output	Pass ?

TC1-1	Get profile	Current user is a new user without profile	{
			id: [profileId]
			name: “User_”+[UserId]
			type: “NORMAL”
			}
				{
				id: [profileId]
				name: “User_”+[UserId]
				type: “NORMAL”
				}
					Yes

TC1-2	Get profile	Current user is an existing user with an existing profile	{ id: [profileId] name: [custom_profile_name]	{ id: [profileId] name: [custom_p	Yes

			type: “NORMAL” } 	rofile_name] type: “NORMAL” }	
TC2-1	Update profile name	New name with length within [1,255]	Success	Success	Yes
TC2-2	Update profile name	New name with name length >=256	Error	Error	Yes
TC2-2	Update profile name	new name with name length==0, i.e., empty string “”	Error	Error	Yes
TC3-1	sSearch profile by user ID	user ID string existing in database	a valid profile object	a valid profile object	Yes
TC3-2	sSearch profile by user ID	user ID string not existing in database	Error	Error	Yes
TC5-1	Change user type	Current: ADMIN Target: NORMAL→MODERATOR	Success	Success	Yes
TC5-2	Change user type	Current: ADMIN Target: MODERATOR→NORMAL	Success	Success	Yes
TC5-3	Change user type	Current: ADMIN Target: NORMAL→RESTRICTED	Success	Success	Yes
TC5-4	Change user type	Current: ADMIN Target: RESTRICTED→NORMAL	Success	Success	Yes
TC5-5	Change user type	Current: ADMIN	Success	Success	Yes

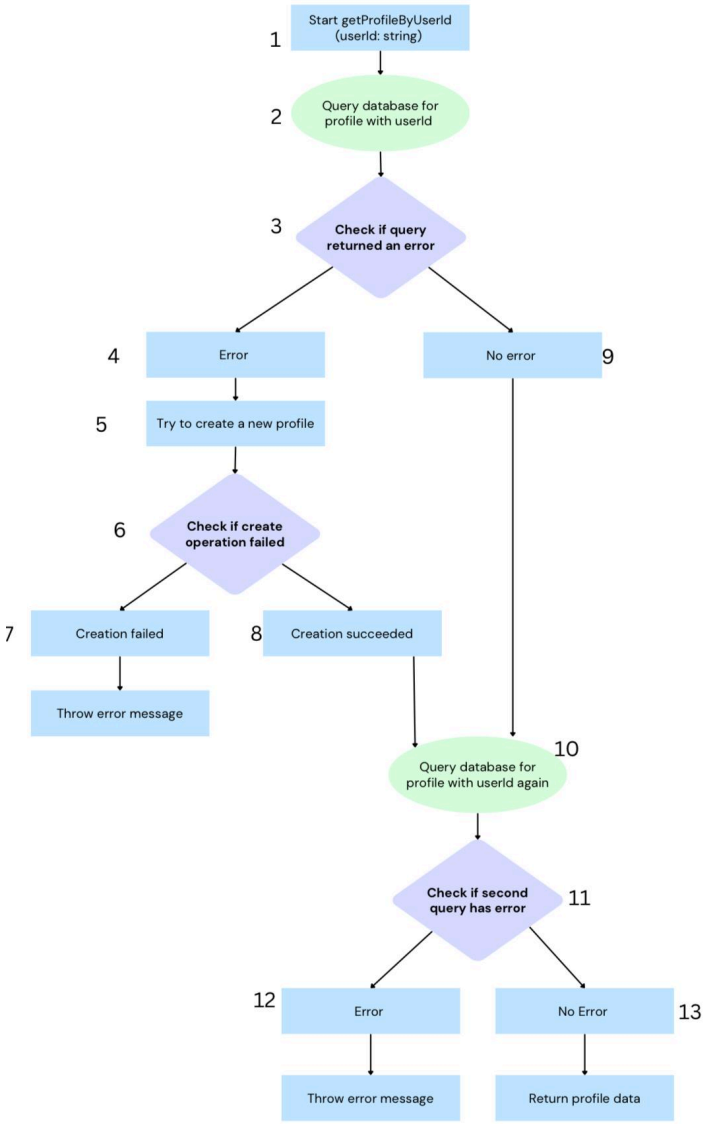
		Target: MODERATOR→RESTRICTED			
TC5-6	Change user type	Current: ADMIN Target: RESTRICTED→MODERATOR	Success	Success	Yes
TC5-7	Change user type	Current: MODERATOR Target: NORMAL→RESTRICTED	Success	Success	Yes
TC5-7	Change user type	Current: MODERATOR Target: RESTRICTED→NO	Success	Success	Yes

2. White Box Testing

2.1 GetProfileByUserId

The getProfileByUserId(userId) method is designed to retrieve a user's profile based on their unique user ID. It begins by querying the database for the profile. If the profile exists and no error is returned, it immediately returns the profile data. However, if the initial query fails (indicating the profile does not exist), the method attempts to create a new default profile for the user. If the profile creation fails, an error message is thrown. If creation is successful, the method performs a second query to retrieve the newly created profile. Should this second query also fail, an error is returned. Otherwise, the method returns the profile data. This approach ensures that every user has a profile, either pre-existing or automatically generated, and includes appropriate error handling at each critical step.

2.1.1 Control Flow Graph



2.1.2 Basic Path Testing

Path Analysis

1. Path 1: Profile exists - Successful Path

1 → 2 → 3 → 9 →10 → 11 → 13

2. Path 2: Profile does not exist - Successful creation

1 → 2 → 3 → 4 → 5 → 6 → 8 → 10 → 11 → 13

3. Path 3: Profile does not exist - Failed creation (standard error)

1 → 2 →3 → 4→5 →6 → 7

4. Path 4: Profile creation succeeds but second query fails

1→2→3→4→5→6→8→10→11→12

2.1.3 Test Cases and Results

Test Cases for Each Path

1. Path 1 Test: Provide an existing userId

2. Path 2 Test: Provide a valid userId that doesn't have a profile yet

3. Path 3 Test: Mock a standard error during profile creation

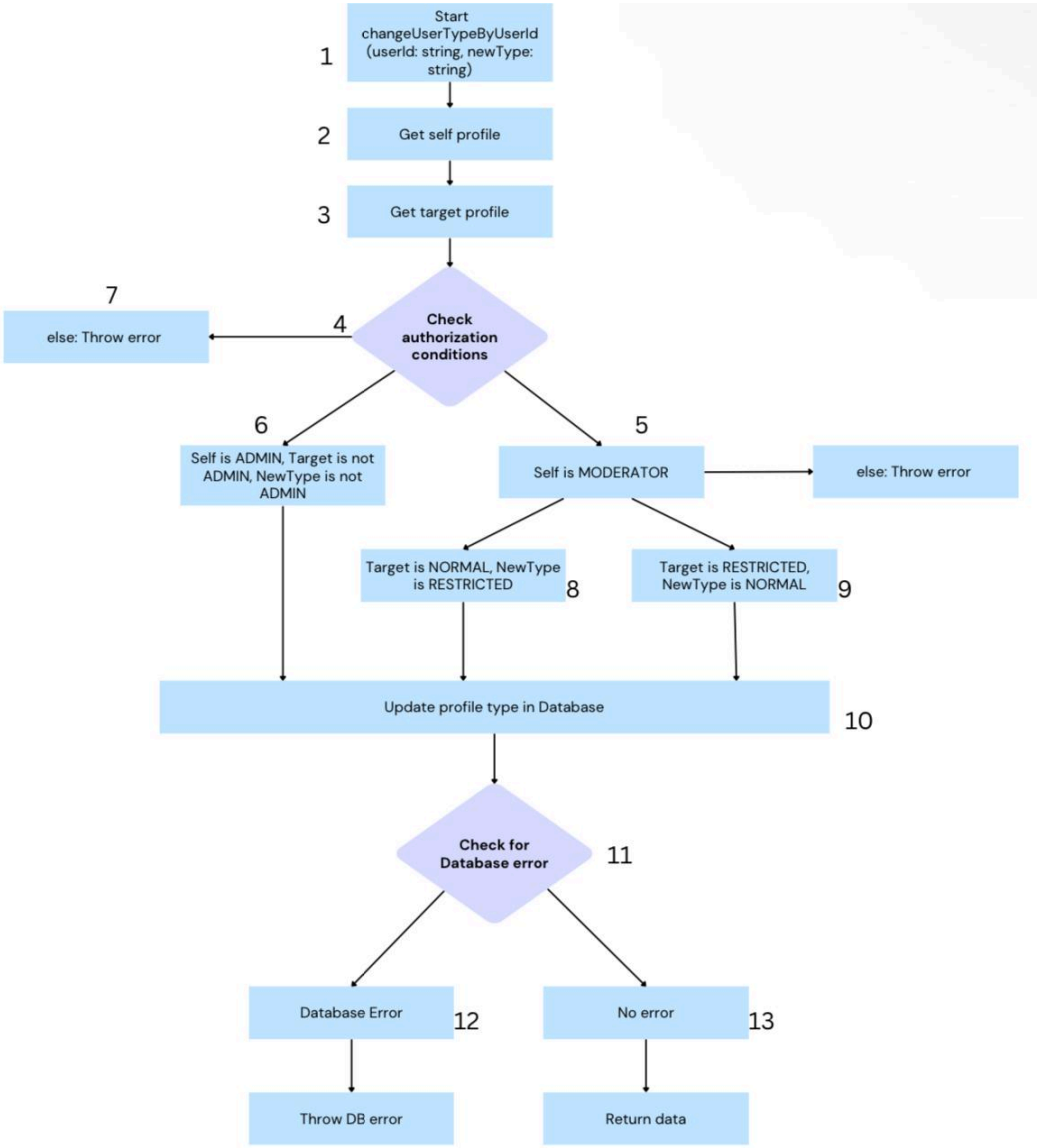
4. Path 4 Test: Mock successful creation but failure on second query

Test ID	Path covered	Input condition	Expected Output	Output
TGP-1	Path1	Provide an existing userID	Return profile data with type, name	Return profile data with type, name
TGP-2	Path2	Provide a valid userId that doesn't have a profile yet	Creates new profile, then returns it	Creates new profile, then returns it
TGP-3	Path3	Mock a standard error during profile creation	Throw creation error message	Throw creation error message
TGP-4	Path4	Mock successful creation but failure on second query	Throw second query error message	Throw second query error message

2.2 ChangeUserTypeByUserId

The changeUserTypeByUserId(userId, newType) method changes the user type using their unique user ID. Firstly, it gets the self-profile and then the target profile. If self is not authorised, an error message is thrown. If self is an Admin, they can only change a non-admin user's type, and the new type must also not be Admin. If the self is a Moderator, they are only allowed to toggle a target user's type between Normal and Restricted. Otherwise, an error is thrown. If the authorisation conditions are met, the system proceeds to update the target user's profile type in the database. Then, it checks for any database error. If an error occurs, it throws a database error. If not, it returns the updated data. This will ensure that the user role changes are strictly for authorised users and any errors are handled appropriately.

2.2.1 Control Flow Graph



2.2.2 Basic Path Testing

Path Analysis

1. Path1 (success) :

1 → 2 → 3 → 4 → 5 → 7 → 8 → 9

2. Path 2 (unauthorized failure) :

1 → 2 → 3 → 4 → 6

3. Path 3 (database error failure):

1 → 2 → 3 → 4 → 5 → 7 → 8 → 10

4. Path 4 (self profile retrieval failure):

1 → 2

5. Path 5 (target profile retrieval failure):

1 → 2 → 3

2.2.3 Test Cases and Results

Test ID	Path covered	Input Conditions	Expected output	Output
CUT-1-1	Path 1	selfType = ADMIN, targetType = NORMAL, newType = RSTRICTED	success	success
CUT-1-2	Path 1	selfType = ADMIN, targetType = MODERATOR, newType = NORMAL	success	success
CUT-1-3	Path 1	selfType = ADMIN, targetType = RESTRICTED, newType =	success	success

		MODERATOR		
CUT-2	Path 2	selfType = MODERATOR, targetType = NORMAL, newType = RESTRICTED	success	success
CUT-3	Path 3 (unauthorized failure)	selfType = ADMIN, targetType = NORMAL, newType = ADMIN	error	error
CUT-4	Path 4 (unauthorized failure - moderator)	selfType = MODERATOR, targetType = NORMAL, newType = MODERATOR	error	error
CUT-5	Path 5 (database error failure)	valid authorization, simulate DB error	error	error
CUT-6	Path 6 (self profile retrieval failure)	Mock getprofile() failure	error	error
CUT-7	Path 7 (target profile retrieval failure)	Mock getProfileByUserId() failure	error	error