

**Lebanese University - Faculty of Sciences**

**Section - I**

**Department of Computer Science**

2025-2026

### Save Campus - Surplus Food Redistribution System

**I3306 Database II**

**Prepared by**

**Hasan Kheireddin 110712**

**Mohammad Alayan 111470**

**Acknowledgements**

We would sincerely like to express our gratitude to our professor, **Dr. Dbouk**, for their invaluable guidance throughout the Database II course.

We are also thankful to the Lebanese University, Faculty of Sciences, for providing the resources and environment that made this work possible.

Finally, We extend our appreciation to our colleagues for the constructive discussions during the design phase of the "Save Campus" system.

**Hasan Kheireddin**

**Mohammad Alayan**

**Table of Content**

[Save Campus - Surplus Food Redistribution System 1](#_ld8nq7kqlkfg)

[**Introduction 4**](#_vl8df0f6410a)

[**CHAPTER 1: Database Application Architecture 5**](#_2z0i2bqu9aqe)

[1. Introduction 5](#_8i0aao59zdtv)

[1.2 User Client (Web/Mobile Application): 5](#_qj3bgi812kor)

[1.3 Application/Web Server: 5](#_t1k3r6rlgxnj)

[1.4 Database Server 5](#_v9hog7aor66k)

[**CHAPTER 2 : Database Modeling 7**](#_mxnr4fntw1i4)

[1. Introduction 7](#_hcglgostruec)

[2. Business Requirements 7](#_mnsxcej43cx)

[3. Entity Relationship Diagram 8](#_2mreyyg2e0fs)

[4. Database Schema 8](#_da6znnu16b96)

[**CHAPTER 3: Services and Interface modeling 9**](#_u0exoe9zw7n7)

[**1. Introduction 9**](#_jcks6un8iwac)

[**2. Interface 9**](#_t9n773s44dlh)

[2.1 Authentication 9](#_feiv6lc6ddoz)

[2.2 Staff Dashboard and Meal Posting Service 10](#_yvbvtla3ptph)

[2.3 Student Meal Browser & Claim Service 11](#_hxlpqnoxi9n)

[2.4 Mock Scenario 13](#_1pylafj5pfpn)

[**CHAPTER 4: Application Implementation and testing 16**](#_nee9axhwpuv6)

[**1. DDL scripts 16**](#_k4qq1k86taj9)

[**2. DML scripts 17**](#_xx015a1x738n)

[**3. Indexing Script 20**](#_bx21djalfiu5)

[**4. Procedures and Functions 20**](#_6gy3kmja9yxo)

[**5. Triggers 24**](#_7mb75r6bxrzt)

[**6. Cursors 24**](#_93xggcbiy5k4)

[**7. Security Privilege 27**](#_ak3f97t8bpor)

[**Conclusion: 28**](#_6cnvdrvih9se)

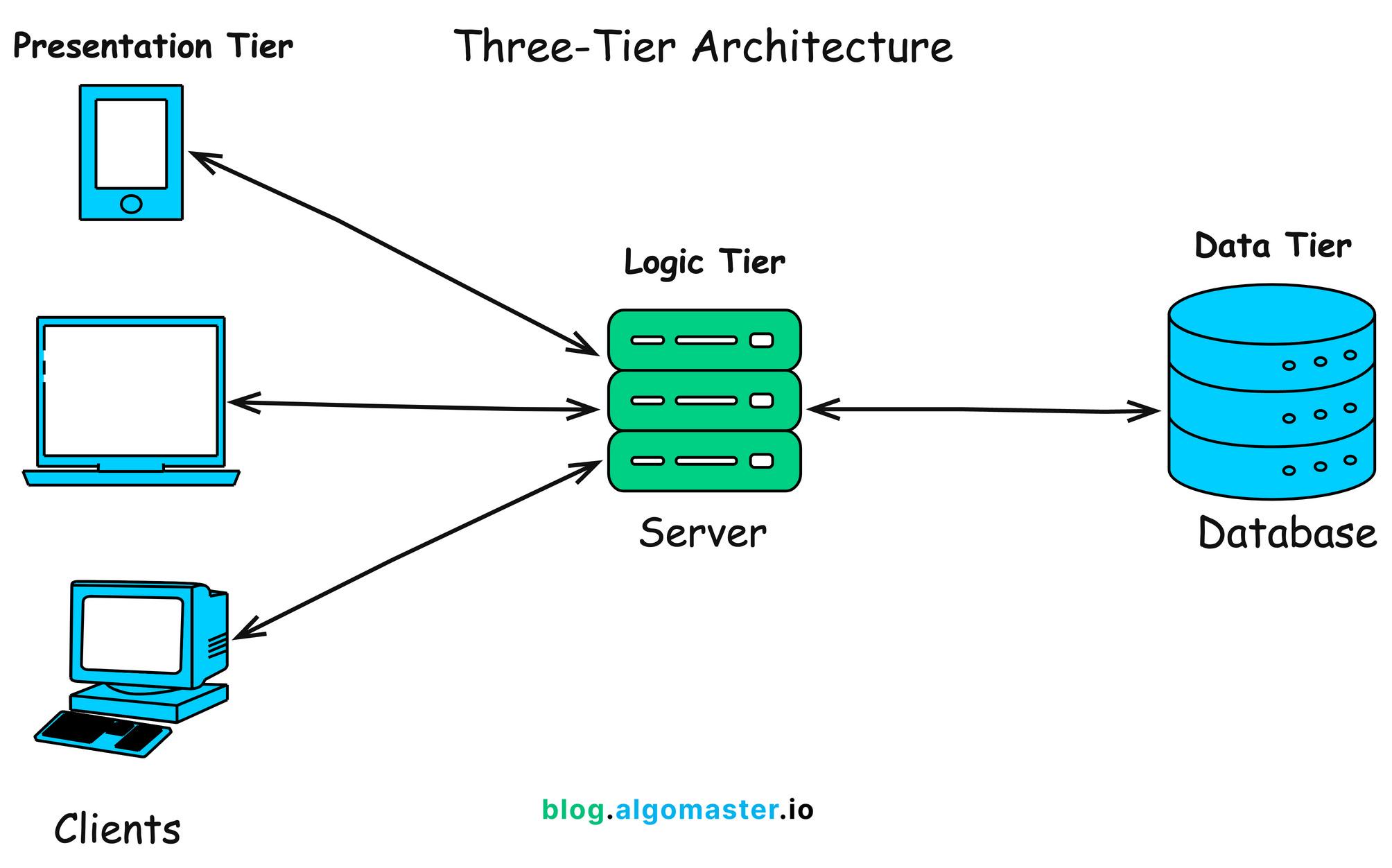
# Introduction

Save Campus is a solution that targets food wastage in campuses by creating an online interface where staff can list their surplus food, and students can request food in return. As part of this endeavor, this project will apply a relational database management system in order to handle users, food, and claims effectively and efficiently. Also, this report will detail the database implementation process and functionality.

# CHAPTER 1: Database Application Architecture

## 1. Introduction

The system uses a client-server network topology. This means that staff and students use a front-end application (client) that sends requests to a back-end server. This server then responds to the requests by communicating with the database to store and retrieve data.

Figure 1. Client-Server Architecture (Adapted from algomaster 2025).

## 1.2 User Client (Web/Mobile Application):

Staff can post and oversee meals by logging in. Students can look through the meals that are offered and make claims. It displays data from the server and manages user input.

## 1.3 Application/Web Server:

It is the place where the business logic is maintained. It deals with the authentication of the user, the handling of the food postings, the claims, and the communication with the database from the client’s end.

## 1.4 Database Server

This is the central data storage facility. It holds all user data and data related to claims and food in a secure manner. The database server uses a database management system that supports transactions for claims and complex queries for reporting.

**DBMS Chosen:** **MySQL** (You can choose PostgreSQL or any other relational DBMS).

# CHAPTER 2 : Database Modeling

## 1. Introduction

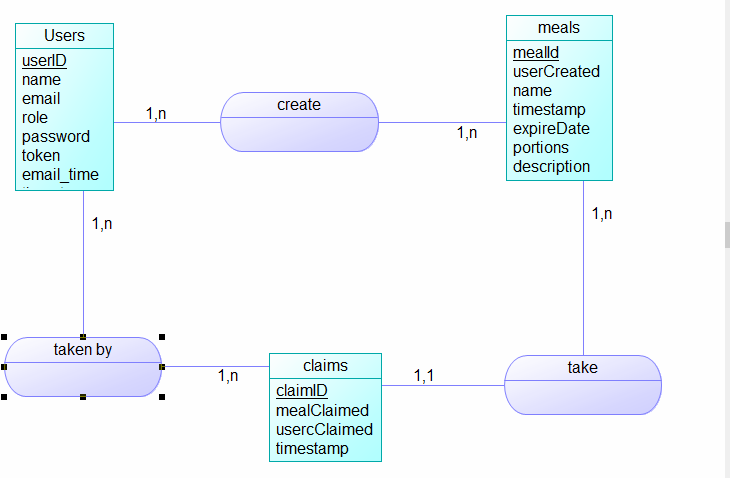
The database design methodology employed is that of three phases, these being the phase of accumulating business requirements, the phase of conceptual entity-relationship diagramming, and the final phase of logical mapping to the relational schema model.

## 2. Business Requirements

The Save Campus system should implement the following mandatory functions:

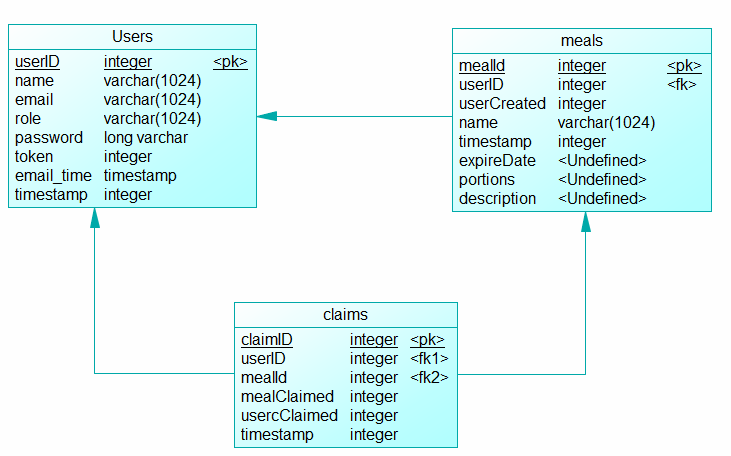
1. User management should include a differentiation between the Staff and Student type of users. They share common fields, including name and email. But differ in authorization.
2. Meal Management: Staff can create, edit, or delete the meal posting details. These details include description, amount, time, and expiry status.
3. Claim Management: Students can browse available meals, meaning that the meals have not been claimed by setting isClaimed to False, and place a claim. A student can only place a claim on a single meal. The system will track the time the student placed a claim on the meal.
4. Integrity Rules: A person cannot own two meals at the same time until they choose to take the previous one. A meal announced in the system by a staff member cannot be owned by the same staff member.

## 3. Entity Relationship Diagram



## 

## 4. Database Schema



# 

# 

# CHAPTER 3: Services and Interface modeling

# 1. Introduction

This chapter introduces the operational services provided by the Save Campus application, modeling the user interface. It has dual operational services based on user roles:

1. Staff Services: Login, posting meals, edit and delete meal functionality, and status viewing.
2. Student Services: Authentication, viewing available meals, claiming meals, and personal claims management.

These are accessed using a web-based interface that has been designed to be simple and intuitive. The wire frames shown below give an illustration of the major user flows that are in the interface.

# 2. Interface

## 2.1 Authentication

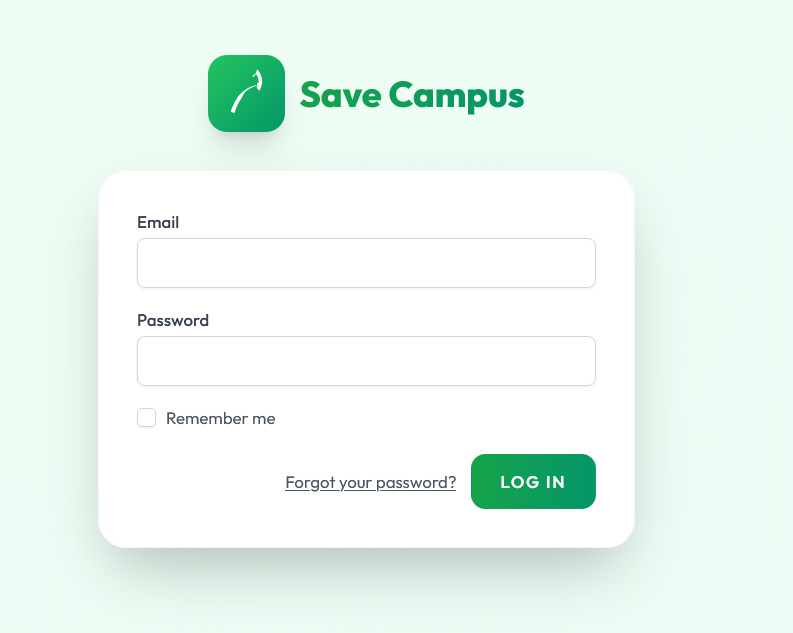
Description: All users are required to log in. The login details are verified against the user table based on the role attribute.

Figure 3.1: The login page

## 2.2 Staff Dashboard and Meal Posting Service

Description: This is the core service for staff. It inserts a new record into the meal table with posted\_by set to the logged-in staff member's ID.

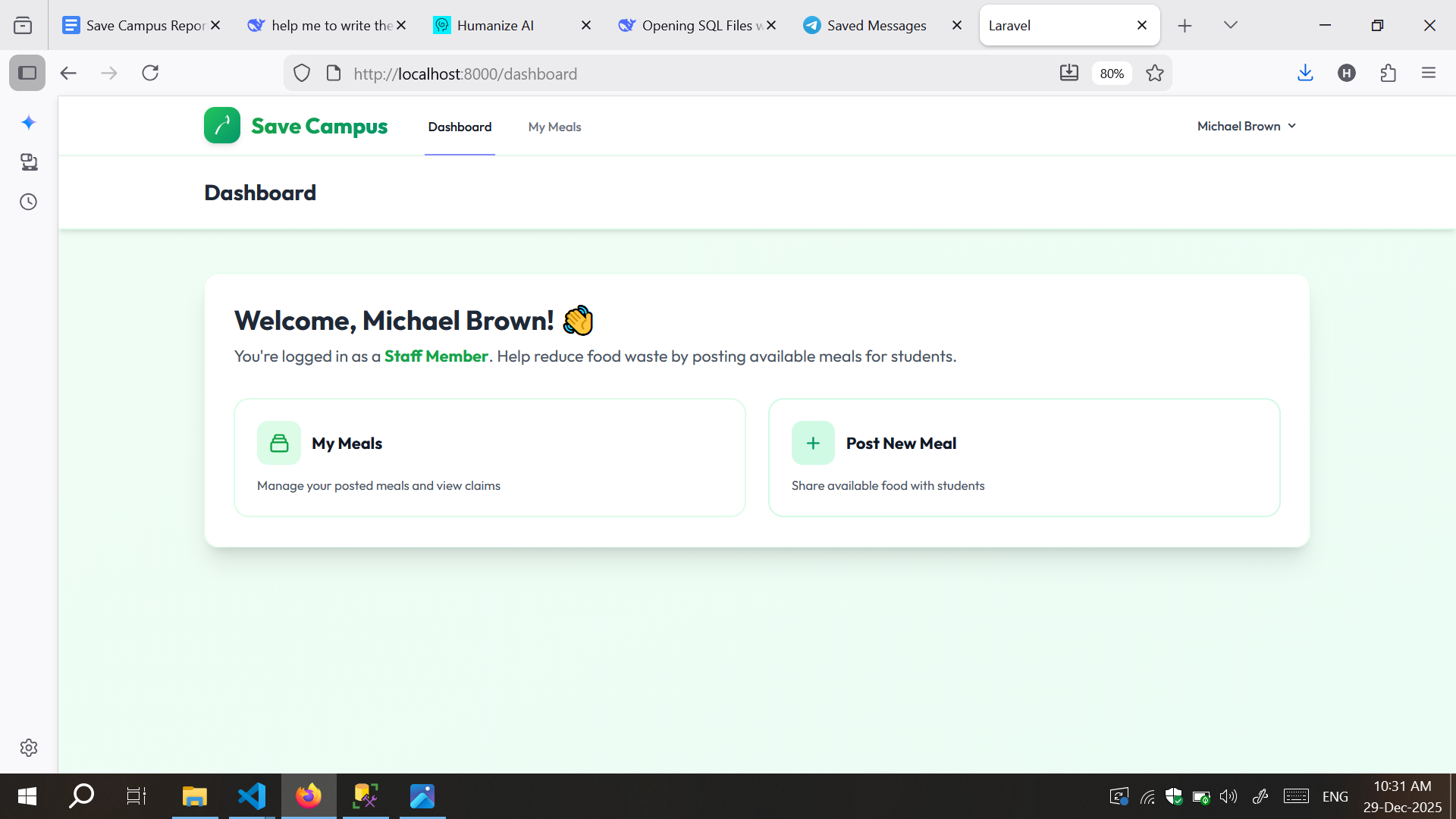


Figure 3.2.a Staff Dashboard

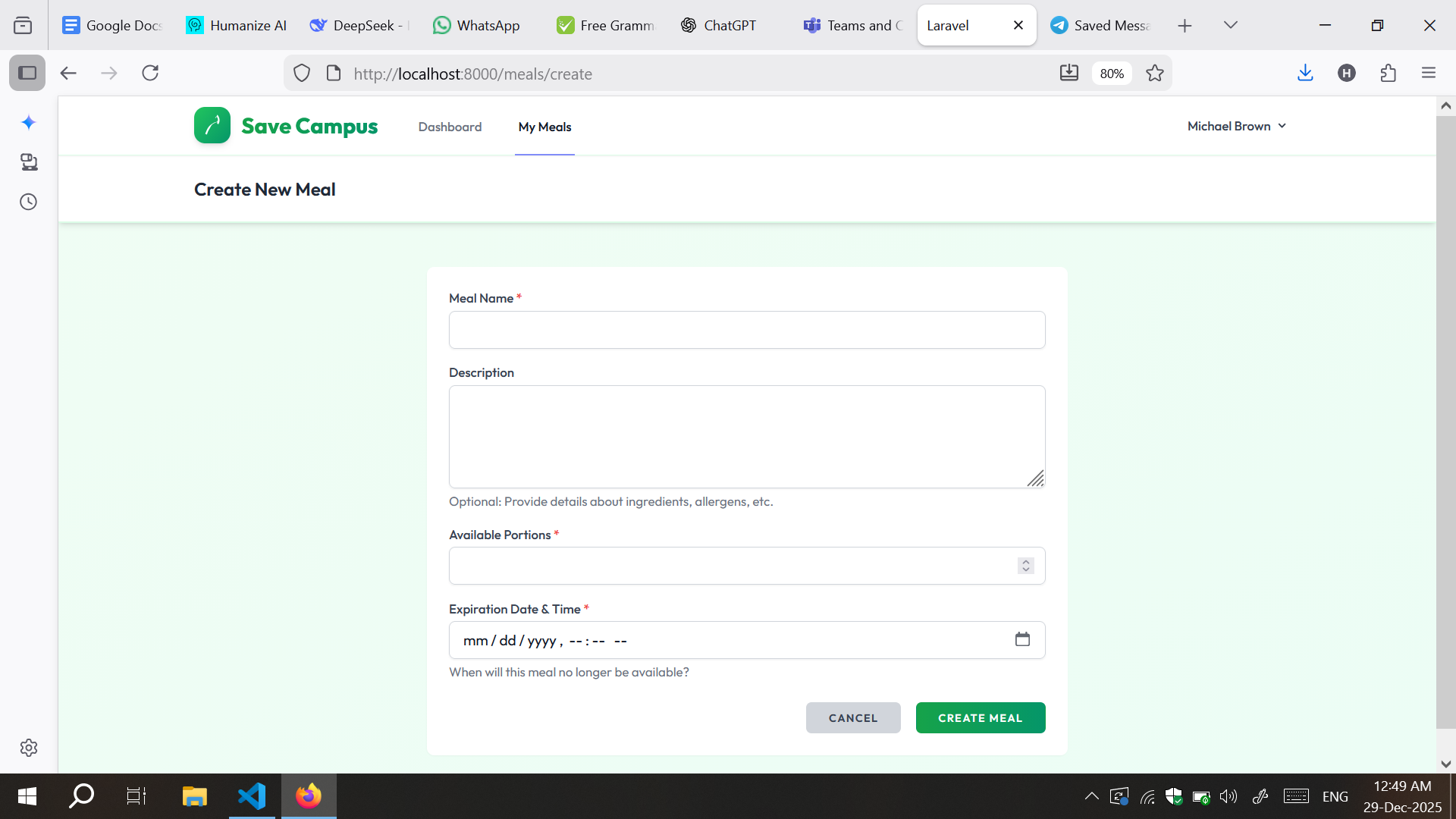


Figure 3.2.b Staff meal creation

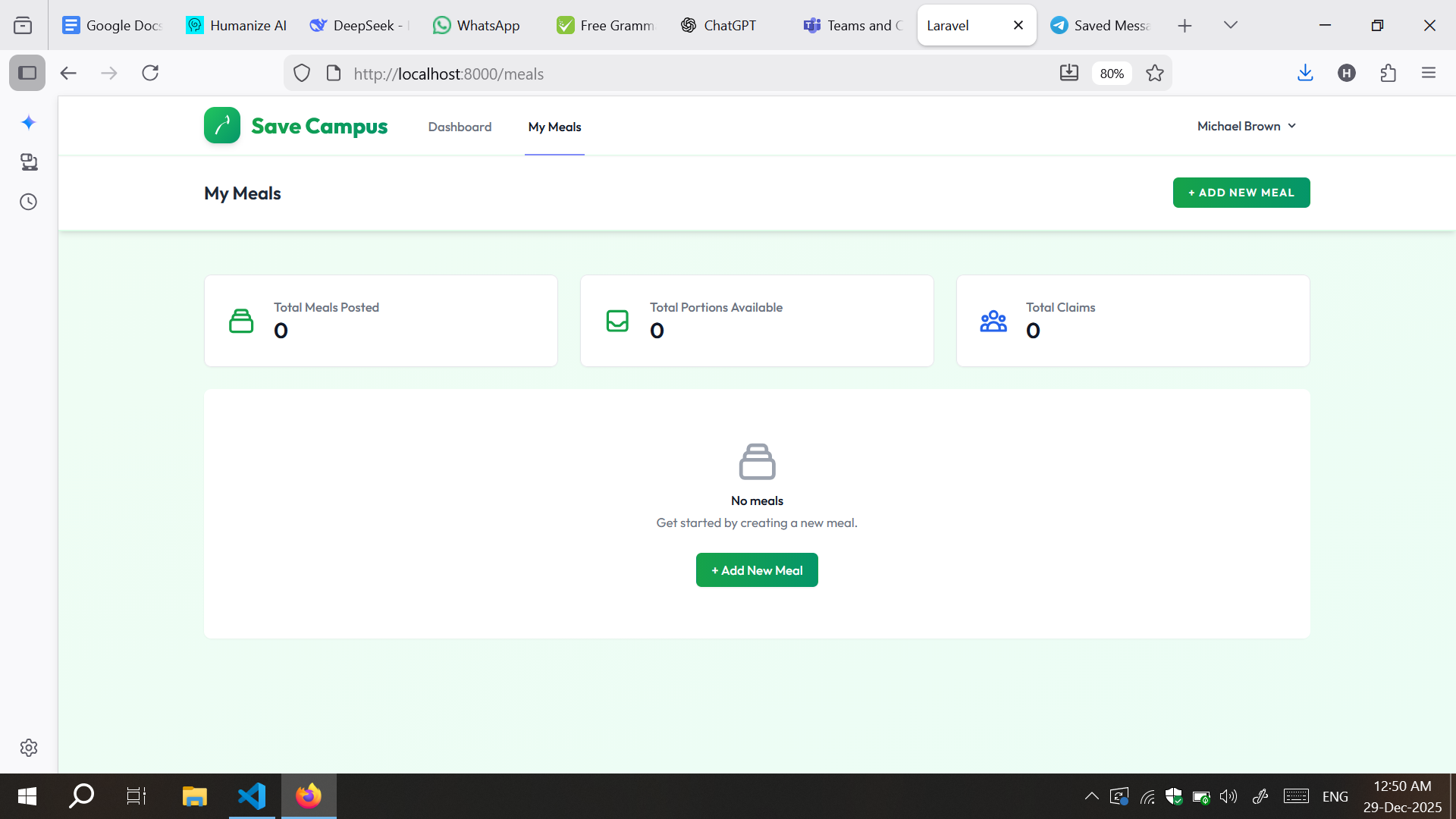


Figure 3.2.c: Staff meal list

## 2.3 Student Meal Browser & Claim Service

Description: This workflow is the core student interaction. It involves a sequence of services that ensure data integrity through a database transaction.

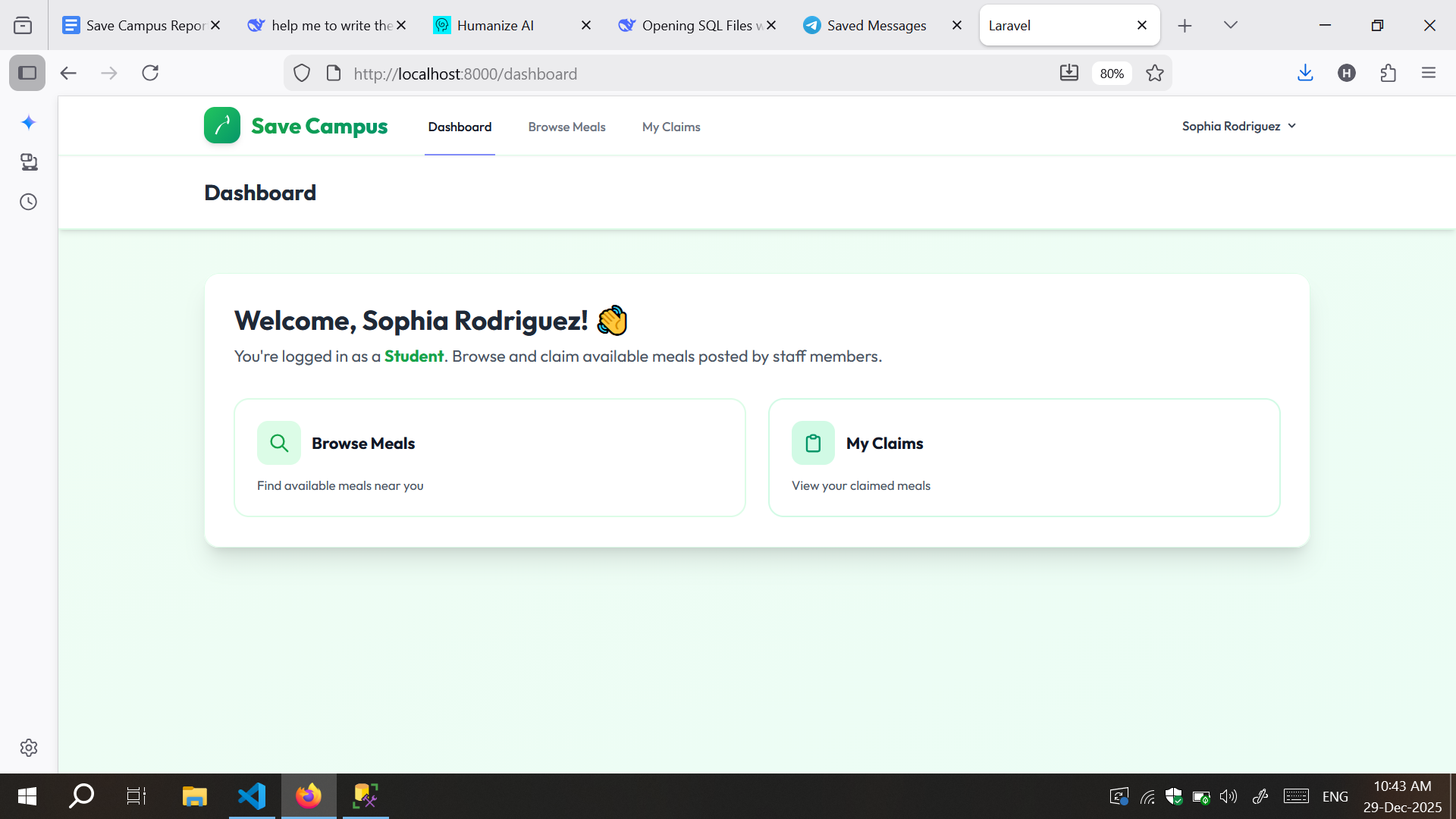


Figure 3.3.a Student Dashboard

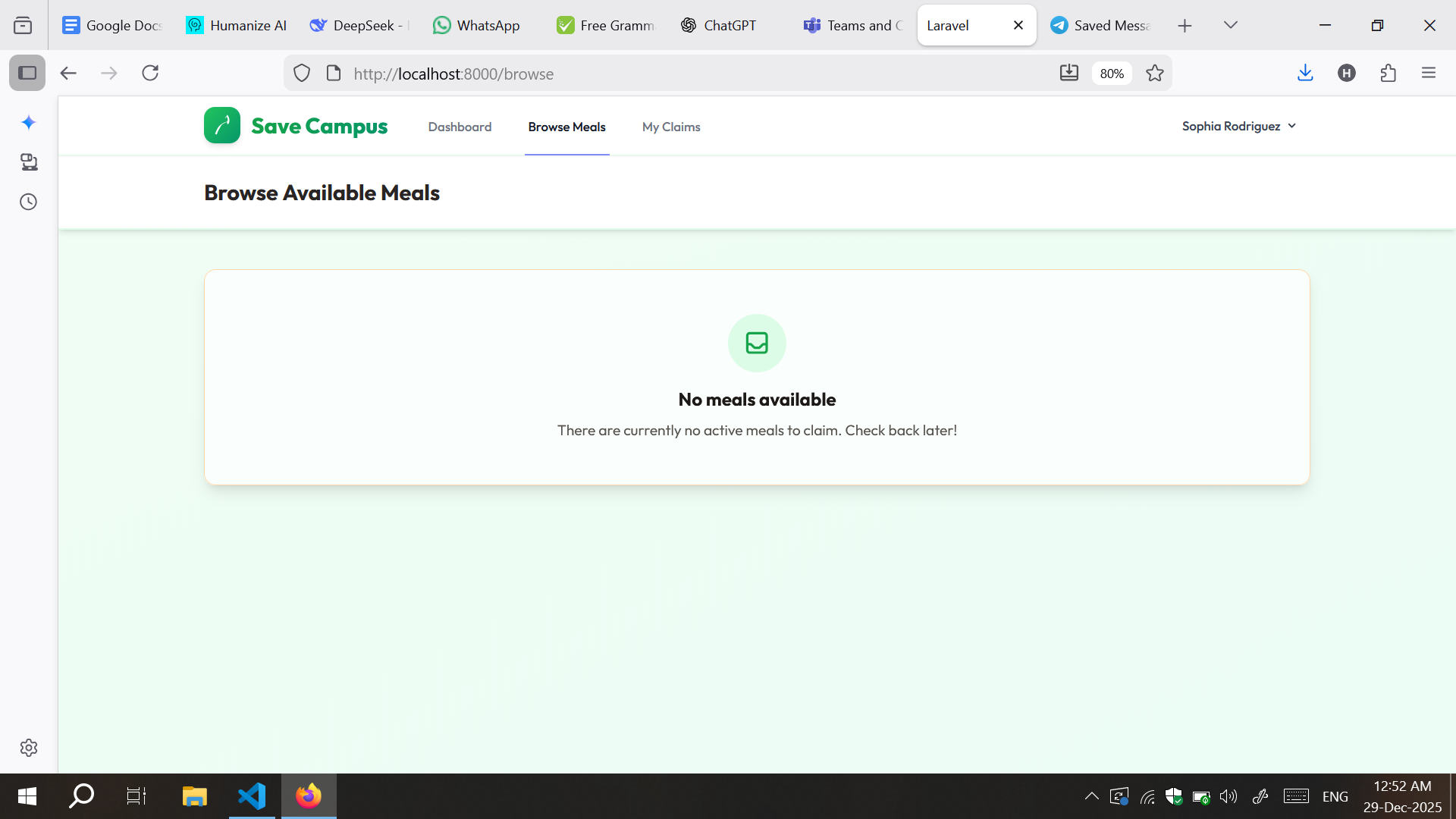


Figure 3.3.b Student Browse Meals Tab

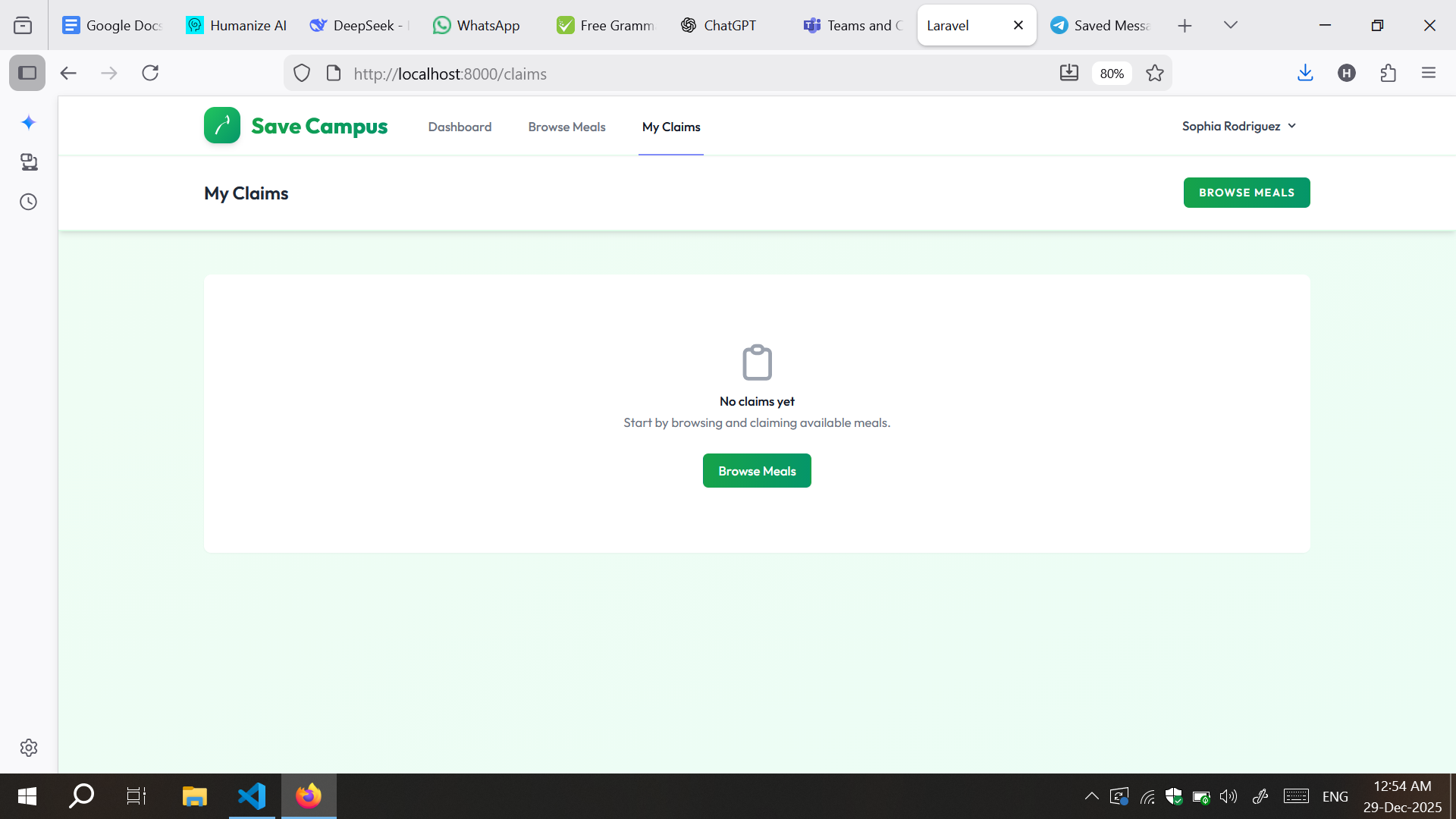


Figure 3.2.c Student Claims Meal Tab

## 2.4 Mock Scenario

Figure 3.3.a Staff create meal(pizza)

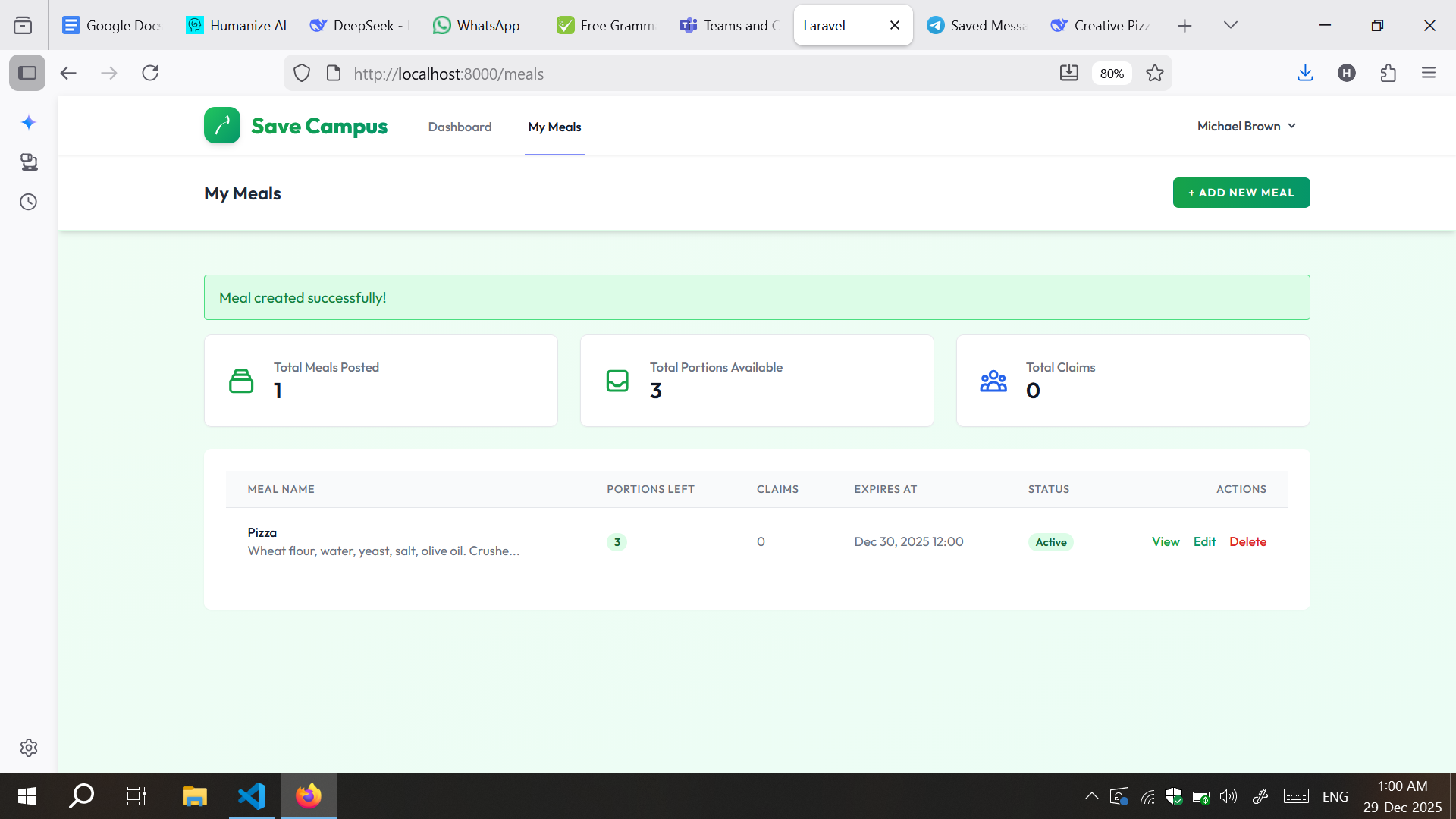


Figure 3.3.b Staff meals list

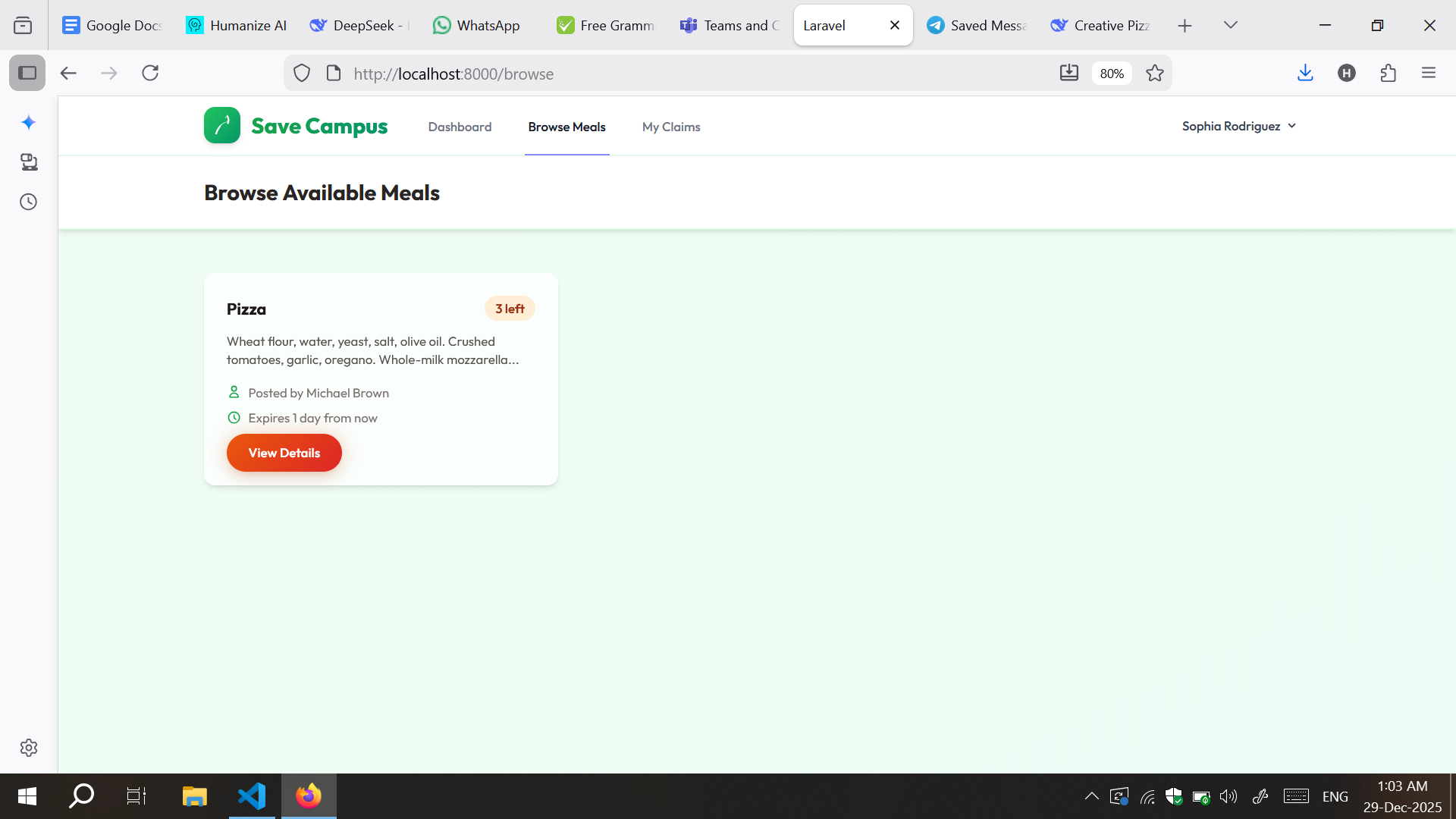


Figure 3.3.c Student Browse meal(pizza)

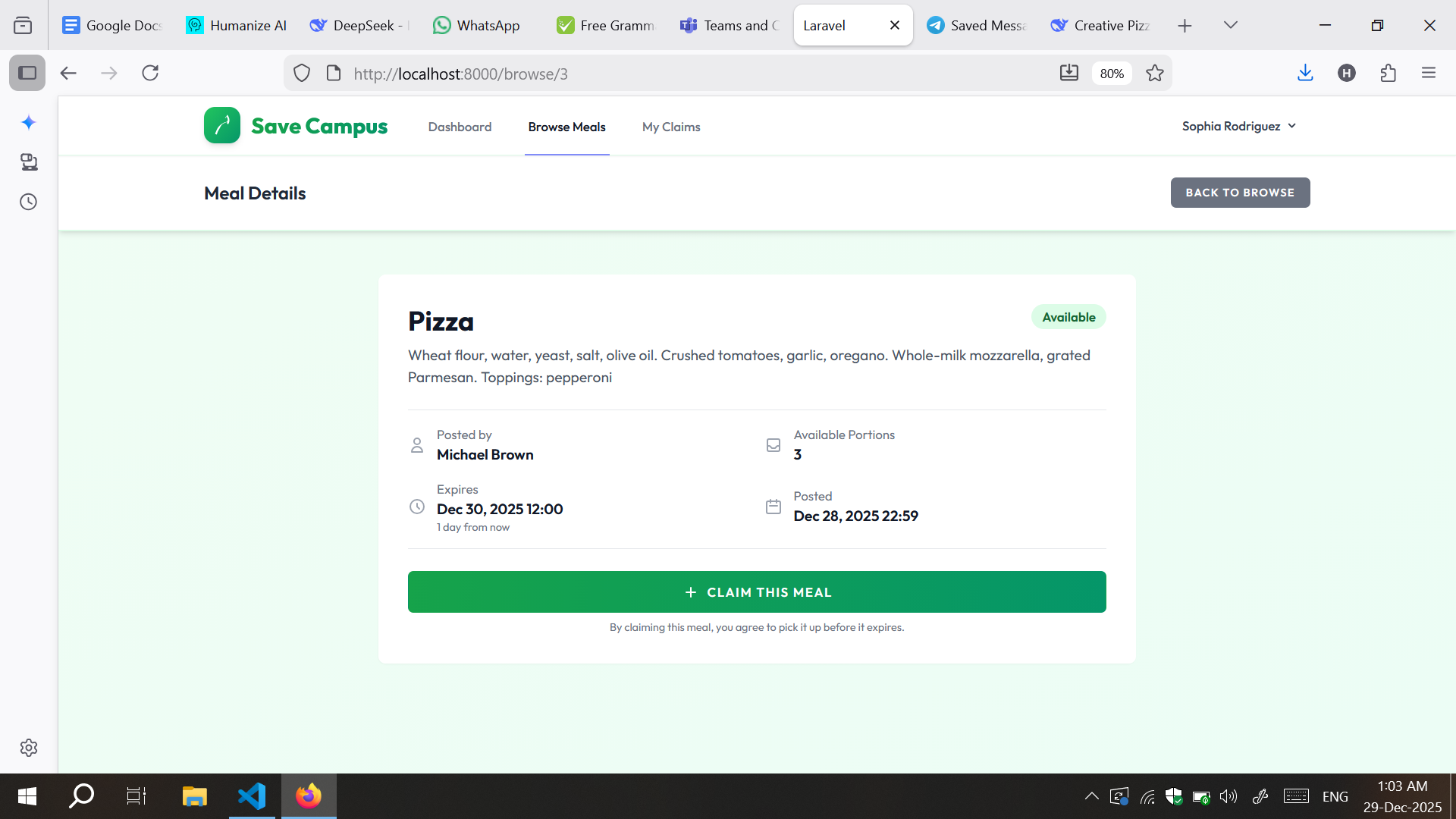


Figure 3.3.d Student meal claim detail

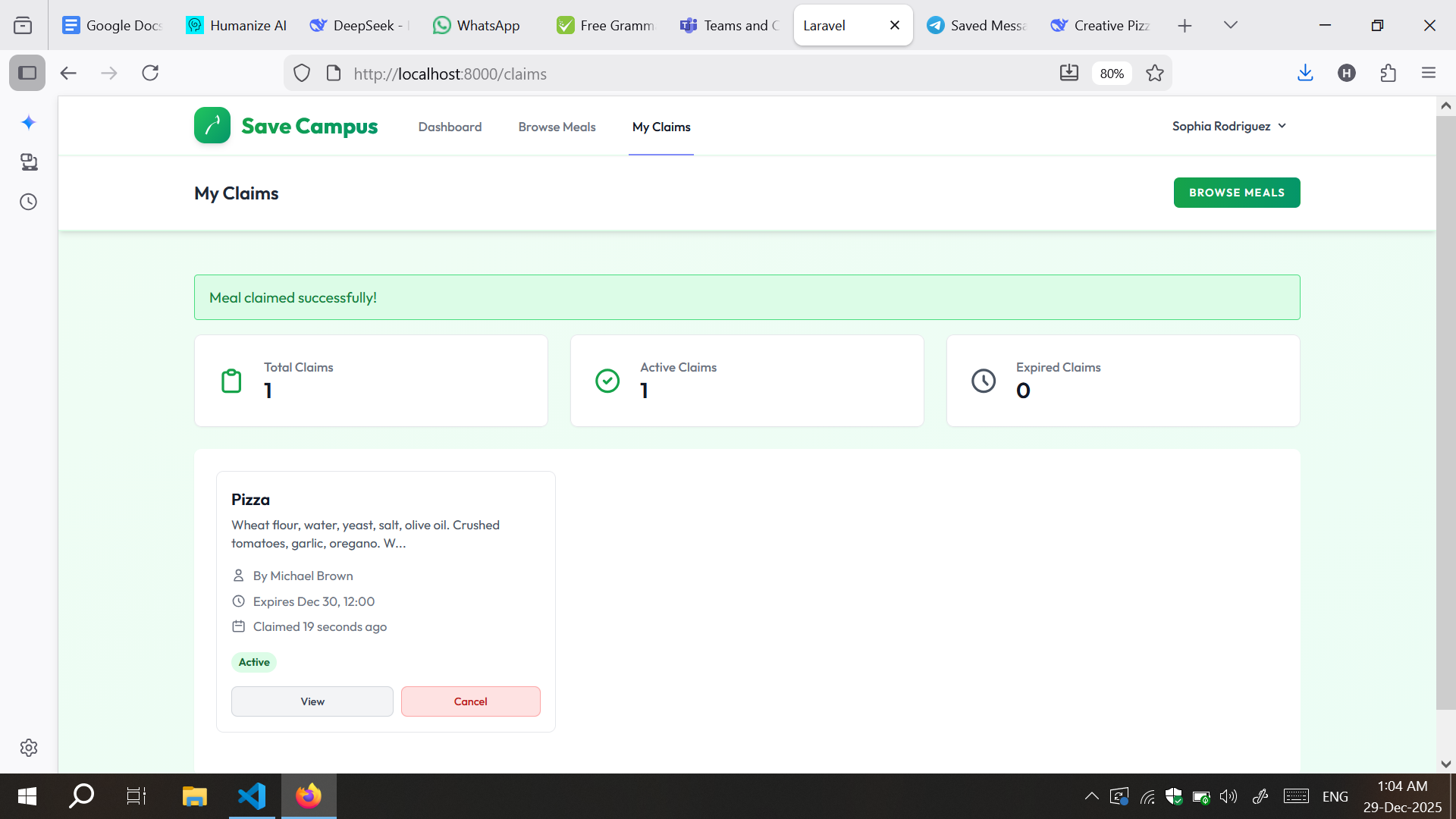


Figure 3.3.e Student claims tab

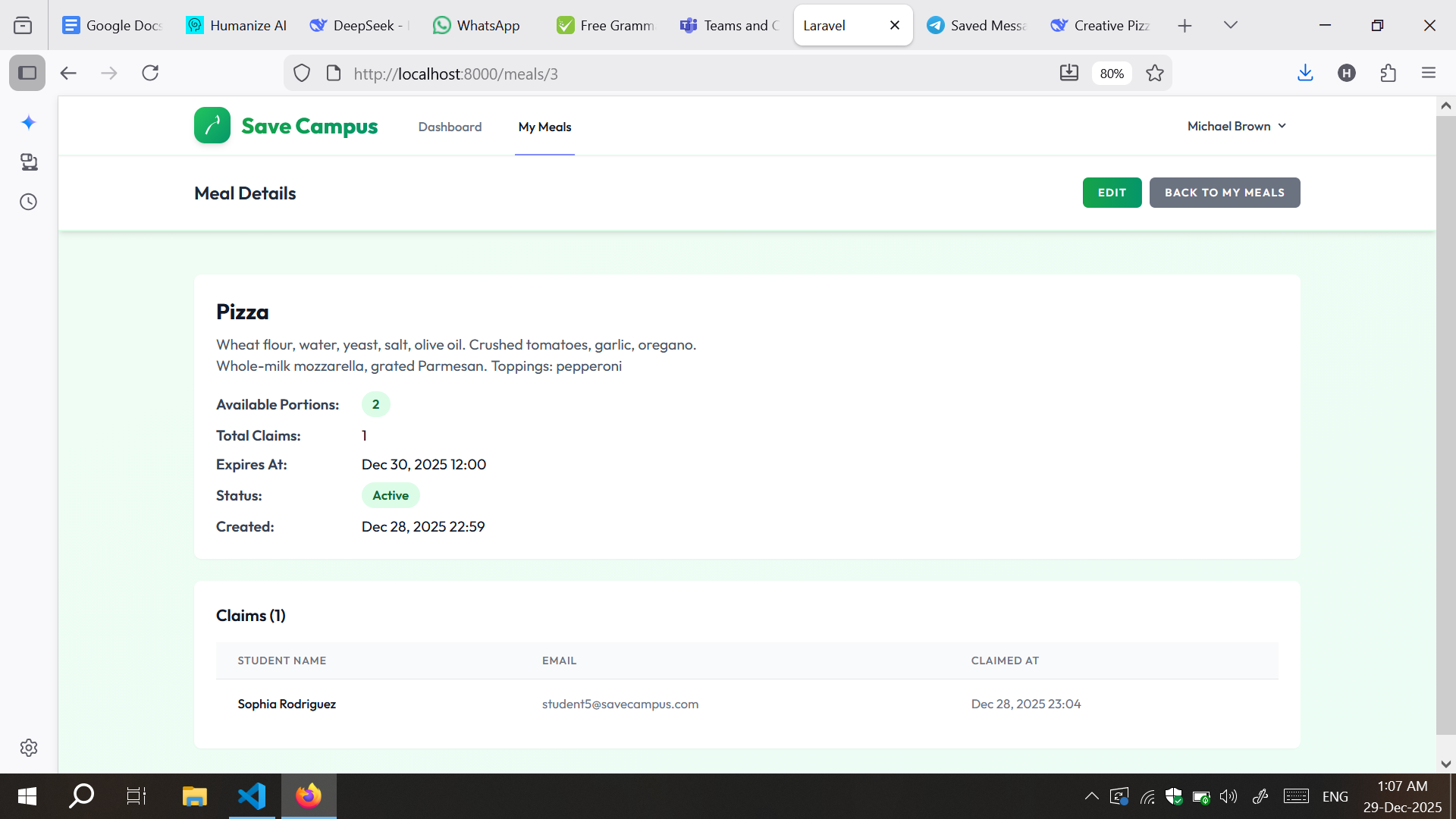


Figure 3.3.f meal detail updated

# CHAPTER 4: Application Implementation and testing

# 1. DDL scripts

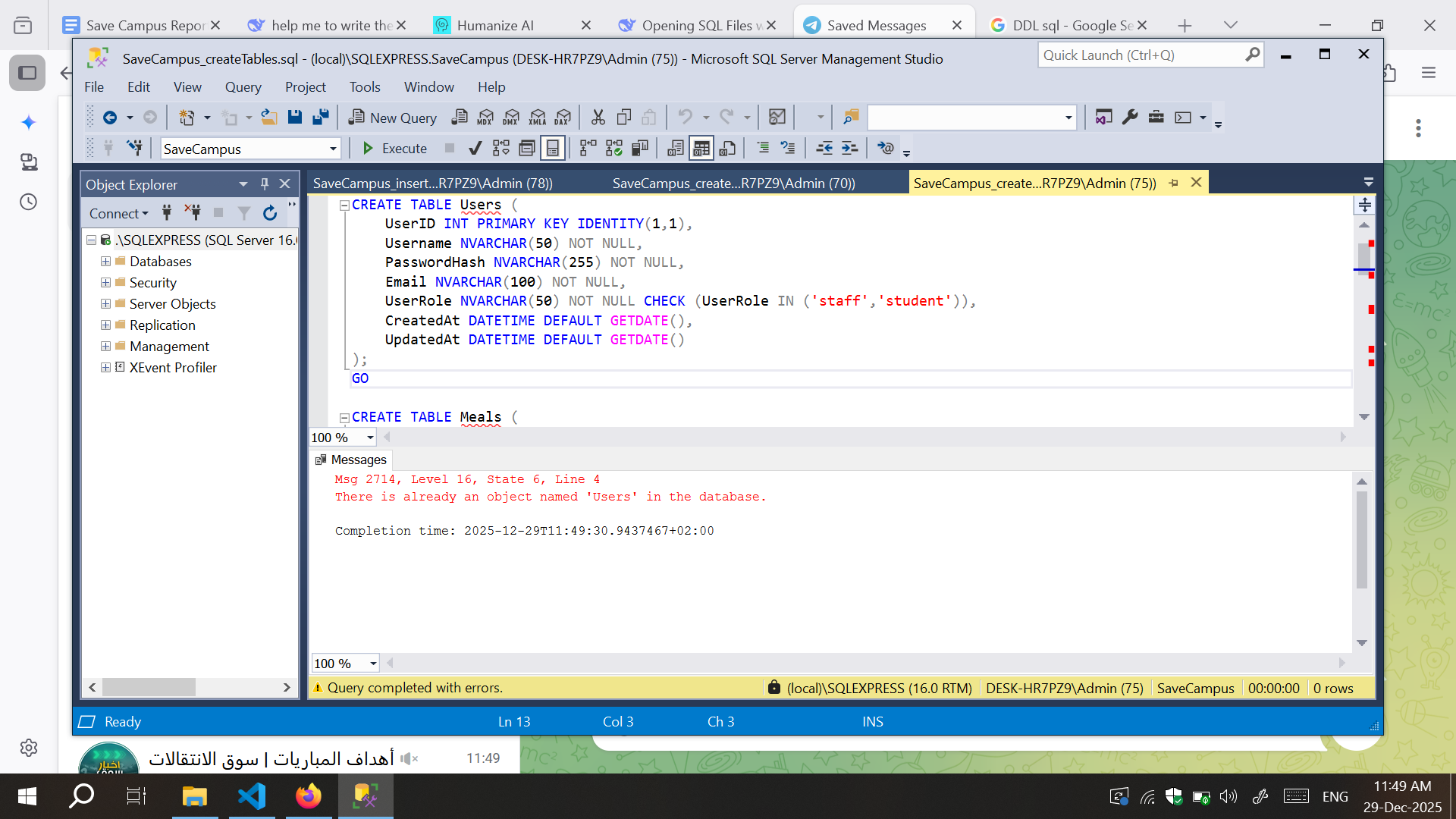


Figure 4.1.a Create Users Table

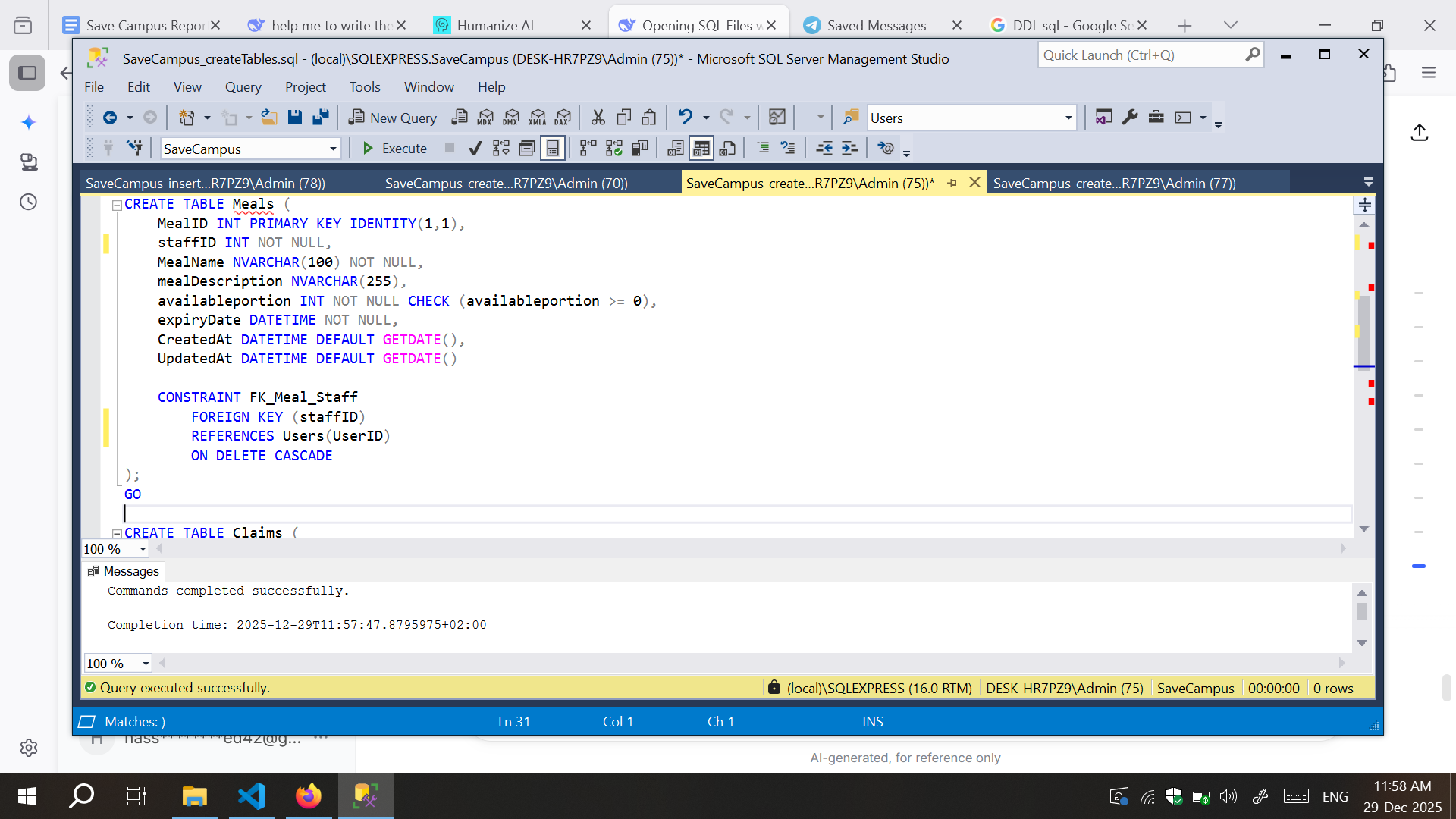


Figure 4.1.b Create Meals Table

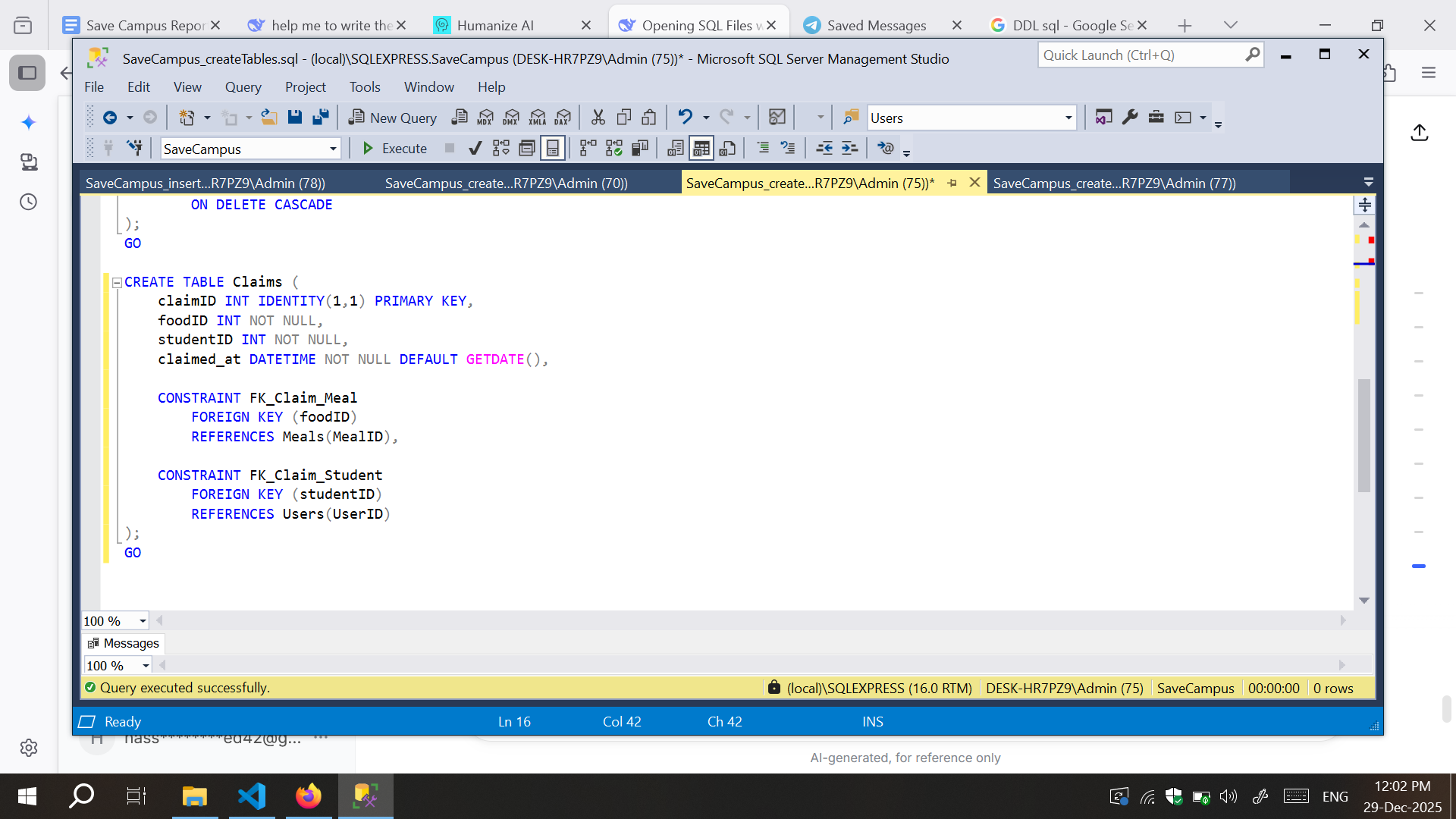


Figure 4.1.c Create Claims Table

# 2. DML scripts

### 

Figure 4.2.a Add Staff Users



Figure 4.2.b Add Student Users

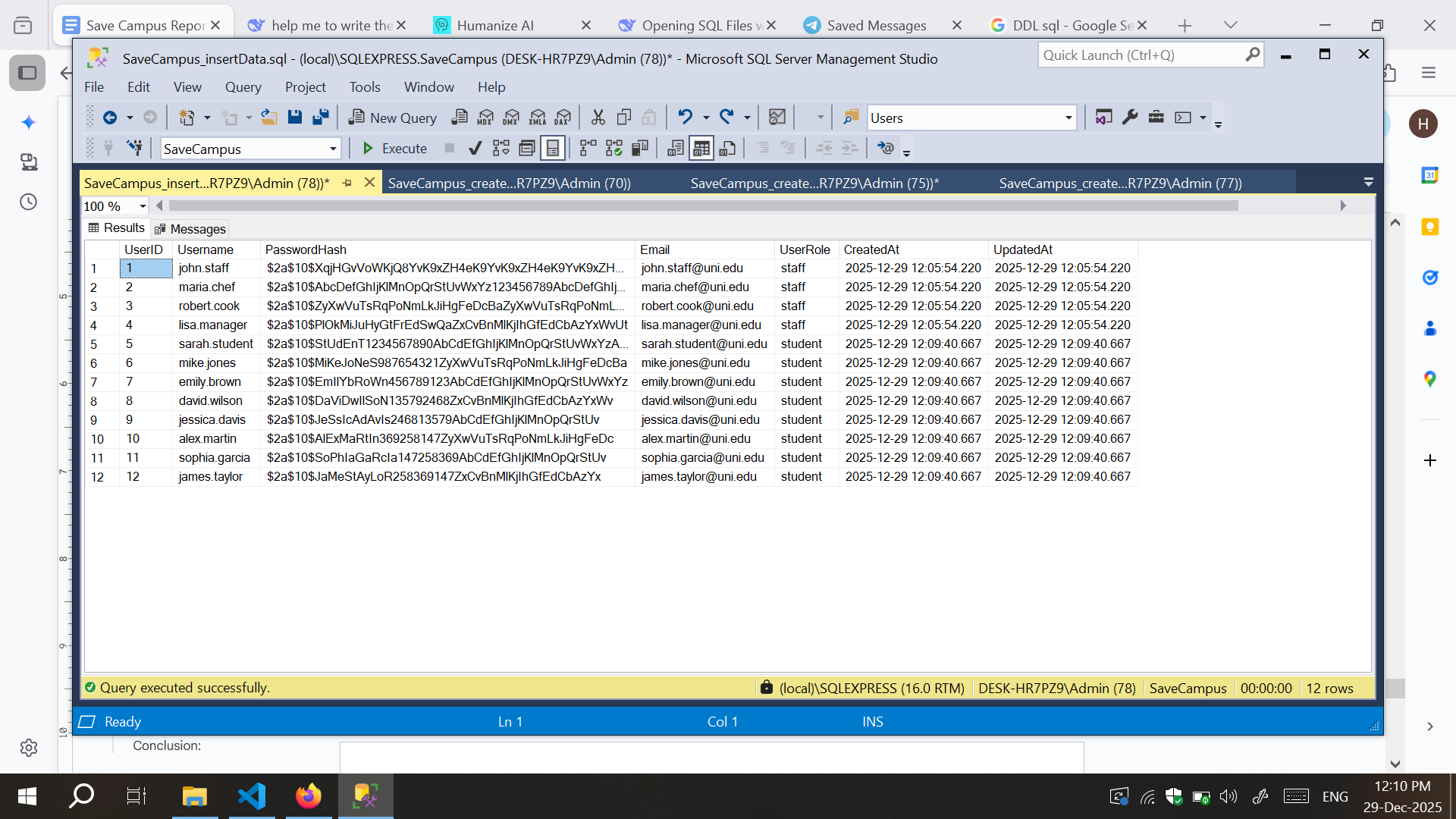


Figure 4.2.c Result of Users table

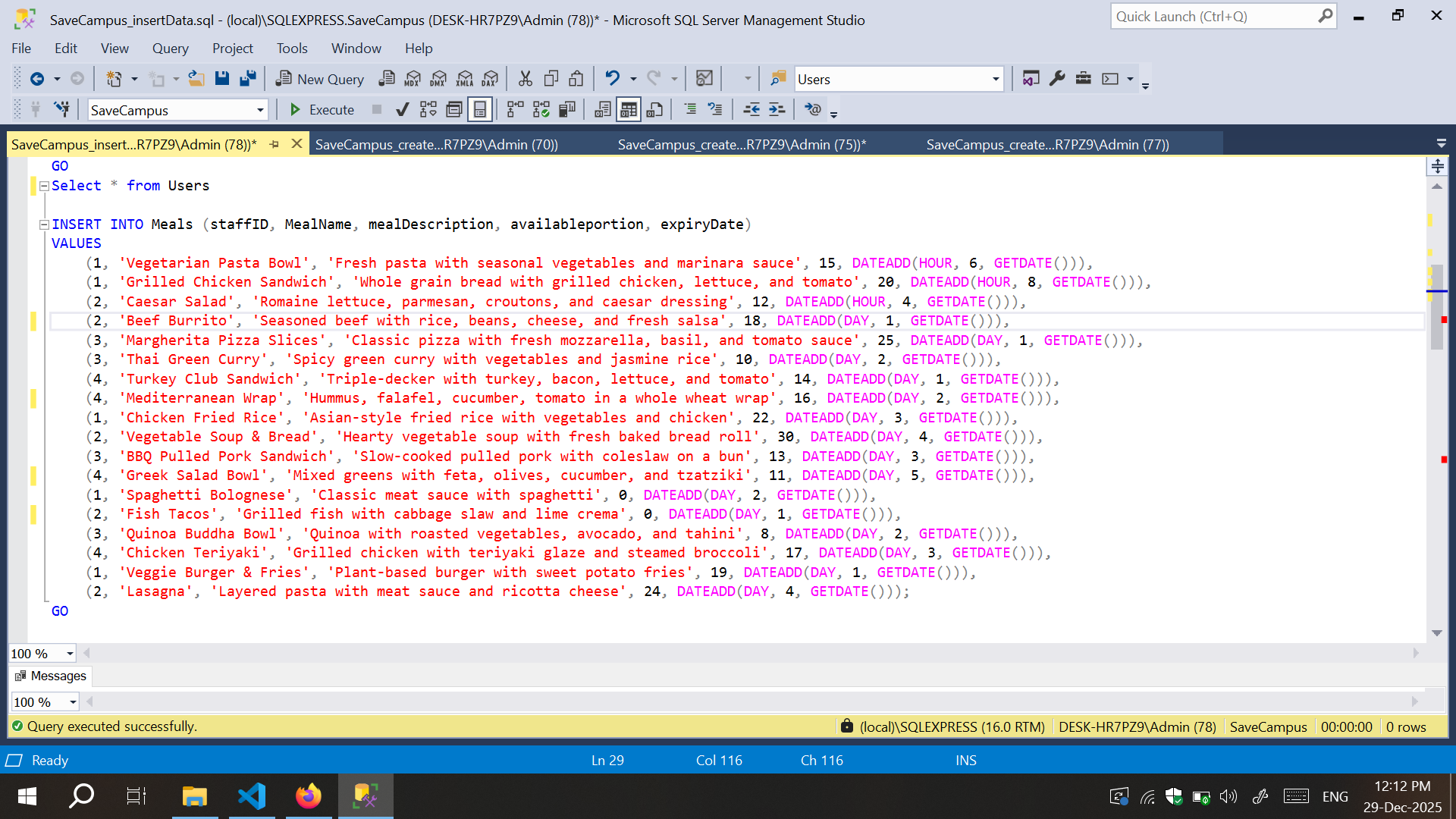


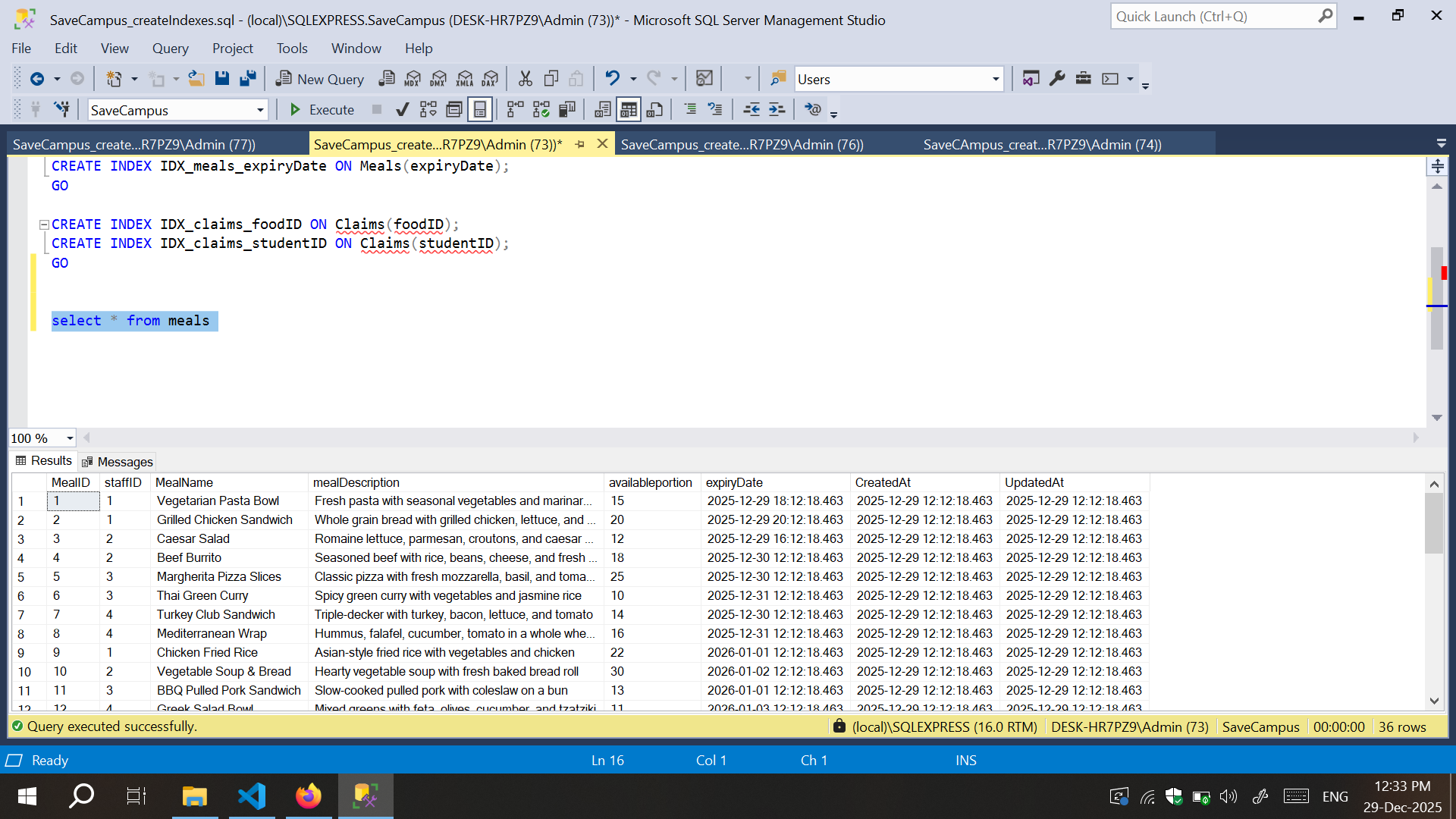
Figure 4.2.d Insert Meals

Figure 4.2.e Meals

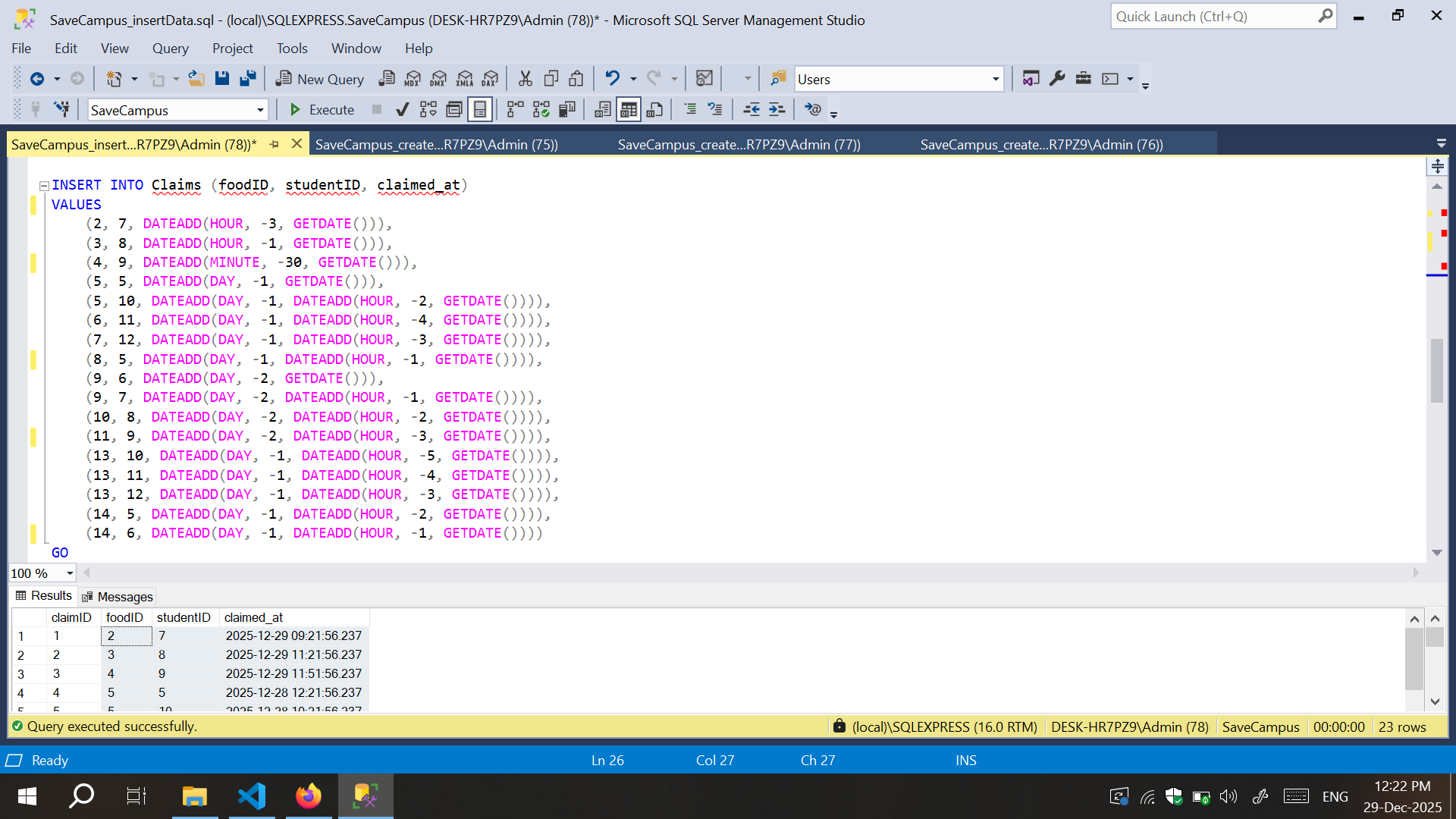


Figure 4.2.f Make claims

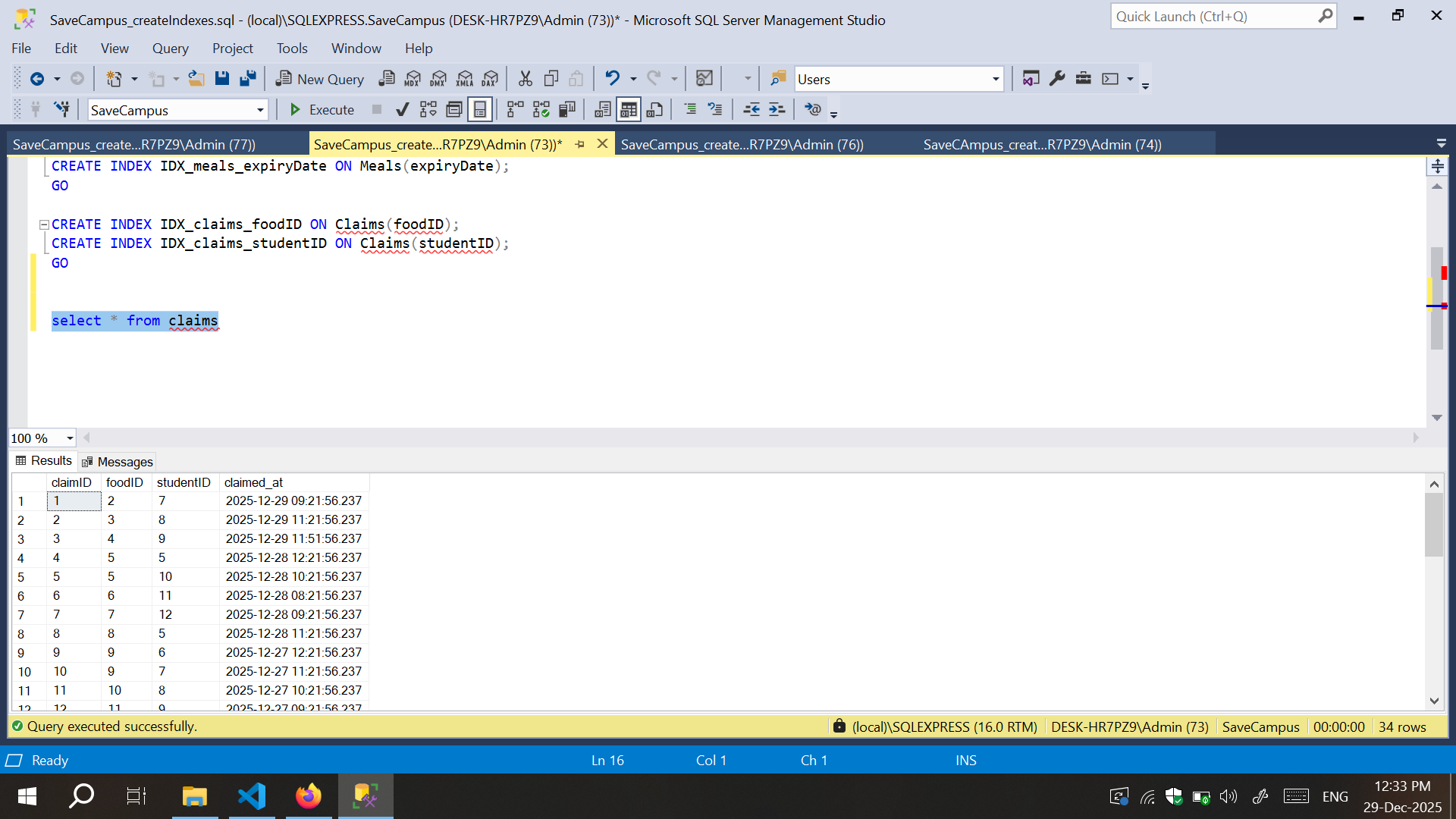
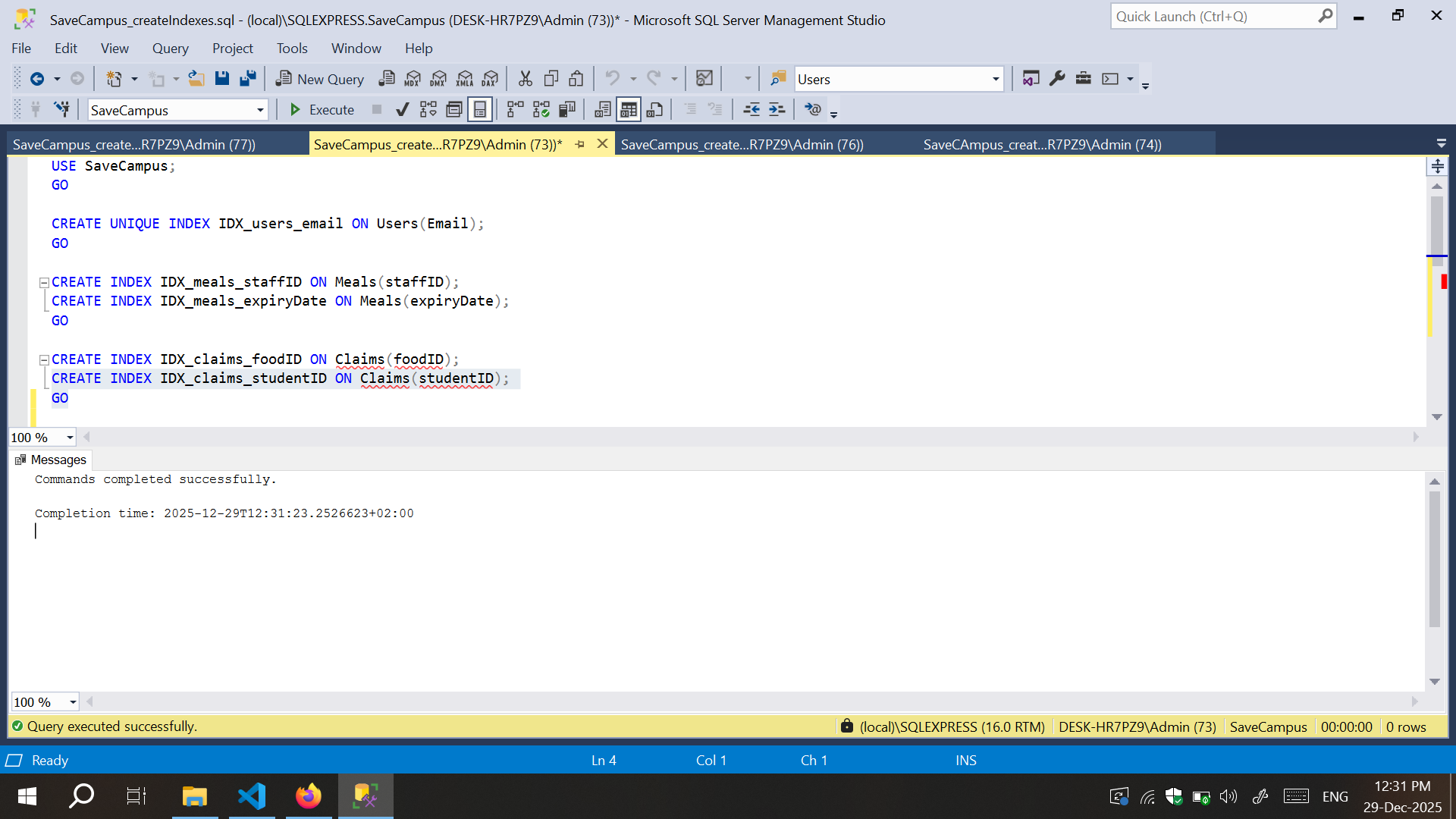


Figure 4.2.g claims result

# 3. Indexing Script



# 4. Procedures and Functions

4.1 This ensures thread-safe claiming with proper concurrency control

CREATE PROCEDURE ClaimMeal

@meal\_id INT,

@user\_id INT

AS

BEGIN

SET NOCOUNT ON;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

BEGIN TRY

DECLARE @available INT;

DECLARE @userRole NVARCHAR(50);

DECLARE @expiryDate DATETIME;

-- Check if user exists and is a student

SELECT @userRole = UserRole

FROM Users

WHERE UserID = @user\_id;

IF @userRole IS NULL

BEGIN

RAISERROR('User not found.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @userRole != 'student'

BEGIN

RAISERROR('Only students can claim meals.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- Lock the meal row and check availability

SELECT @available = availableportion, @expiryDate = expiryDate

FROM Meals WITH (UPDLOCK, ROWLOCK)

WHERE MealID = @meal\_id;

IF @available IS NULL

BEGIN

RAISERROR('Meal not found.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @expiryDate < GETDATE()

BEGIN

RAISERROR('Meal has expired.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @available < 1

BEGIN

RAISERROR('No portions left.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- Check if student already claimed this meal

IF EXISTS (SELECT 1 FROM Claims WHERE foodID = @meal\_id AND studentID = @user\_id)

BEGIN

RAISERROR('You have already claimed this meal.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- Insert claim

INSERT INTO Claims (foodID, studentID, claimed\_at)

VALUES (@meal\_id, @user\_id, GETDATE());

-- Update available portions

UPDATE Meals

SET availableportion = availableportion - 1,

UpdatedAt = GETDATE()

WHERE MealID = @meal\_id;

COMMIT TRANSACTION;

PRINT 'Meal claimed successfully!';

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);

END CATCH

END;

GO

4.2 Add a new meal (for staff only)

CREATE PROCEDURE AddMeal

@staff\_id INT,

@meal\_name NVARCHAR(100),

@meal\_description NVARCHAR(255),

@available\_portions INT,

@expiry\_date DATETIME

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRANSACTION;

BEGIN TRY

DECLARE @userRole NVARCHAR(50);

-- Verify user is staff

SELECT @userRole = UserRole

FROM Users

WHERE UserID = @staff\_id;

IF @userRole IS NULL

BEGIN

RAISERROR('User not found.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @userRole != 'staff'

BEGIN

RAISERROR('Only staff members can add meals.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @available\_portions < 0

BEGIN

RAISERROR('Available portions must be non-negative.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

IF @expiry\_date <= GETDATE()

BEGIN

RAISERROR('Expiry date must be in the future.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- Insert the meal

INSERT INTO Meals (staffID, MealName, mealDescription, availableportion, expiryDate)

VALUES (@staff\_id, @meal\_name, @meal\_description, @available\_portions, @expiry\_date);

COMMIT TRANSACTION;

PRINT 'Meal added successfully!';

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);

END CATCH

END;

GO

4.4 Count claims for a specific meal

CREATE FUNCTION CountMealClaims (@meal\_id INT)

RETURNS INT

AS

BEGIN

DECLARE @count INT;

SELECT @count = COUNT(\*)

FROM Claims

WHERE foodID = @meal\_id;

RETURN ISNULL(@count, 0);

END;

GO

4.5 Get total meals posted by a staff member

CREATE FUNCTION GetStaffMealCount (@staff\_id INT)

RETURNS INT

AS

BEGIN

DECLARE @count INT;

SELECT @count = COUNT(\*)

FROM Meals

WHERE staffID = @staff\_id;

RETURN ISNULL(@count, 0);

END;

GO

# 5. Triggers

5.1 Update UpdatedAt timestamp on Users table modification

CREATE TRIGGER trg\_Users\_UpdateTimestamp

ON Users

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

UPDATE Users

SET UpdatedAt = GETDATE()

WHERE UserID IN (SELECT UserID FROM inserted);

END;

GO

5.2 Prevent deletion of meals with claims

CREATE TRIGGER trg\_Meals\_PreventDeleteWithClaims

ON Meals

INSTEAD OF DELETE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS (

SELECT 1

FROM deleted d, Claims c

WHERE d.MealID = c.foodID

)

BEGIN

RAISERROR('Cannot delete meals that have been claimed. Remove claims first.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- If no claims exist, allow deletion

DELETE FROM Meals

WHERE MealID IN (SELECT MealID FROM deleted);

END;

GO

# 6. Cursors

6.1 This cursor iterates through expired meals and marks them

CREATE PROCEDURE ExpireOldMeals

AS

BEGIN

SET NOCOUNT ON;

DECLARE @meal\_id INT;

DECLARE @meal\_name NVARCHAR(100);

DECLARE @available\_portions INT;

DECLARE @expired\_count INT = 0;

DECLARE @wasted\_portions INT = 0;

-- Declare cursor for expired meals

DECLARE meal\_cursor CURSOR FOR

SELECT MealID, MealName, availableportion

FROM Meals

WHERE expiryDate < GETDATE() AND availableportion > 0;

-- Open cursor

OPEN meal\_cursor;

-- Fetch first row

FETCH NEXT FROM meal\_cursor INTO @meal\_id, @meal\_name, @available\_portions;

-- Loop through all expired meals

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Log expired meal

PRINT 'Expiring meal: ' + @meal\_name + ' (ID: ' + CAST(@meal\_id AS NVARCHAR) +

') - Wasted portions: ' + CAST(@available\_portions AS NVARCHAR);

-- Update statistics

SET @expired\_count = @expired\_count + 1;

SET @wasted\_portions = @wasted\_portions + @available\_portions;

-- Set available portions to 0 for expired meals

UPDATE Meals

SET availableportion = 0,

UpdatedAt = GETDATE()

WHERE MealID = @meal\_id;

-- Fetch next row

FETCH NEXT FROM meal\_cursor INTO @meal\_id, @meal\_name, @available\_portions;

END

-- Close and deallocate cursor

CLOSE meal\_cursor;

DEALLOCATE meal\_cursor;

-- Print summary

PRINT '';

PRINT 'Summary:';

PRINT 'Total expired meals: ' + CAST(@expired\_count AS NVARCHAR);

PRINT 'Total wasted portions: ' + CAST(@wasted\_portions AS NVARCHAR);

END;

GO

6.2 Update meal statistics using cursor

CREATE PROCEDURE UpdateMealStatistics

AS

BEGIN

SET NOCOUNT ON;

DECLARE @meal\_id INT;

DECLARE @meal\_name NVARCHAR(100);

DECLARE @claim\_count INT;

DECLARE @staff\_name NVARCHAR(50);

PRINT '========================================';

PRINT 'MEAL STATISTICS REPORT';

PRINT '========================================';

PRINT '';

-- Declare cursor for all meals

DECLARE stats\_cursor CURSOR FOR

SELECT m.MealID, m.MealName, u.Username

FROM Meals m, Users u

WHERE m.staffID = u.UserID

ORDER BY m.CreatedAt DESC;

-- Open cursor

OPEN stats\_cursor;

-- Fetch first meal

FETCH NEXT FROM stats\_cursor INTO @meal\_id, @meal\_name, @staff\_name;

-- Loop through all meals

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Get claim count using our function

SET @claim\_count = dbo.CountMealClaims(@meal\_id);

-- Print meal statistics

PRINT 'Meal: ' + @meal\_name + ' (ID: ' + CAST(@meal\_id AS NVARCHAR) + ')';

PRINT 'Posted by: ' + @staff\_name;

PRINT 'Total Claims: ' + CAST(@claim\_count AS NVARCHAR);

PRINT '----------------------------------------';

-- Fetch next meal

FETCH NEXT FROM stats\_cursor INTO @meal\_id, @meal\_name, @staff\_name;

END

-- Close and deallocate cursor

CLOSE stats\_cursor;

DEALLOCATE stats\_cursor;

PRINT '';

PRINT 'Statistics report generated successfully!';

END;

GO

# 7. Security Privilege

CREATE ROLE staffRole;

CREATE ROLE studentRole;

GO

GRANT SELECT, INSERT, UPDATE, DELETE ON Users TO staffRole;

REVOKE INSERT ON Claims FROM staffRole;

REVOKE SELECT ON StudentClaimHistory FROM staffRole;

GO

GRANT SELECT ON ActiveMeals TO studentRole;

GRANT INSERT ON Claims TO studentRole;

GRANT SELECT ON StudentClaimHistory TO studentRole;

REVOKE INSERT, UPDATE, DELETE ON Meals FROM studentRole;

REVOKE SELECT ON StaffMealStats FROM studentRole;

GO

# Conclusion:

The **Save Campus** database has been fully implemented according to the designed schema. The scripts provided are production-ready and include necessary optimizations (indexes, transaction handling). The system meets all specified business requirements and is prepared for integration with a front-end application layer.