Test Document

for

Master Invoice

Version 1.0

Prepared by

Group 1:		Group Name: while (1)
Cezan Vispi Damania	230310	cezanvd23@iitk.ac.in
Ch V Sai Koushik	230312	skoushik23@iitk.ac.in
Challa Kethan	230317	kethanc23@iitk.ac.in
Challa Umesh Varun	230318	cumesh23@iitk.ac.in
Chilamakuri Kundan Sai	230330	ckundans23@iitk.ac.in
Gudi Praneeth Sai	230425	gudips23@iitk.ac.in
Kishore Senthil Kumar	230566	kishores23@iitk.ac.in
Sai Saketh Mogillapalli	230895	saisaketh23@iitk.ac.in
Srijani Gadupudi	231033	srijanig23@iitk.ac.in
Vundela Obula Reddy	231178	voreddy23@iitk.ac.in

Course: CS253

Mentor TA: Hemang Mohanlal Khatri

Date: 05-04-2025

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Cezan Vispi Damania Ch V Sai Koushik Challa Kethan Challa Umesh Varun Chilamakuri Kundan Sai Gudi Praneeth Sai Kishore Senthil Kumar Sai Saketh Mogillapalli Srijani Gadupudi Vundela Obula Reddy	The first version of the test document	06/04/2025

1 Introduction

1.1 Test Strategy:

System testing and integration testing were performed manually to assess the overall functionality, usability, and compliance of the integrated software product with specified requirements. Automated testing was primarily utilized for unit testing of individual components. We utilized **Django's test framework** for automation to ensure that each component functioned correctly on its own. As a result, the testing approach combines both manual and automated methods to ensure comprehensive test coverage.

1.2 Testing Timeline:

During the development phase, preliminary tests were carried out to verify the core functionalities and ensure the software's operational readiness. The majority of in-depth testing was conducted after implementation. As a result, necessary adjustments could be made with minimal disruption to the overall software architecture.

1.3 Testers:

The developers conducted the testing process. To maintain objectivity and eliminate potential bias during testing, developers were assigned to test components they had not been directly involved in developing.

1.4 Coverage Criteria:

Manual system testing covered all user stories, functional requirements, and use case scenarios to ensure comprehensive validation of the software from an end-user perspective. For unit testing, automated testing focused on achieving function, branch and statement coverage.

1.5 Tools used for Testing:

We used the Django testing framework for unit tests and employed Dbeaver for integration testing, accessing the database and analysing how data is being stored and modified. Apache HTTP server for system testing to simulate heavy traffic and test concurrency.

2 Unit Testing

1. AUTHENTICATION

a)REGISTER

In this unit we have tested whether new user can be created using new username and new emailID.

Unit Details:[User class(Django inbuilt)]

Test Owner: While(1)

Test Date:28-03-2025

Test Results:

- The user was able to proceed to verify otp page on giving new username and emailID.
- The user was not able to proceed with using already existing emailID or username.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```
def setUp(self):
    self.existing_user = User.objects.create_user(
        username='existinguser',
        email='existing@example.com',
        password='testpassword123'
    )
    self.url = reverse('login2')
```

initialising test database

```
def test_successful_registration(self):
    response = self.client.post(self.url, {
        'register': '1',
        'username': 'newuser',
        'email': 'newuser@example.com',
        'password1': 'securePassword123!',
        'password2': 'securePassword123!',
    })
    self.assertRedirects(response, reverse('verify_otp'))
    #check that user is not created yet
    self.assertFalse(User.objects.filter(username='newuser').exists())
    #check that the session contains the registration data
    self.assertIn('register_data', self.client.session)
    self.assertEqual(self.client.session['register_data']['username'], 'newuser')
```

Test whether the page redirects to verify_otp page after entering new username, emailID.

```python
def test_registration_with_existing_username(self):
    response = self.client.post(self.url, {
        'register': '1',
        'username': 'existinguser',
        'email': 'uniqueemail@example.com',
        'password1': 'securePassword123!',
        'password2': 'securePassword123!',
    })
    self.assertContains(response, 'This username is already taken.')

def test_registration_with_existing_email(self):
    response = self.client.post(self.url, {
        'register': '1',
        'username': 'anotheruser',
        'email': 'existing@example.com',
        'password1': 'securePassword123!',
        'password2': 'securePassword123!',
    })
    self.assertContains(response, "This email is already registered.")
```

Test whether it will give error if we use already existing emailID or username for registering.

b)LOGIN

In this unit, we have tested whether the user was able to log in successfully on entering all credentials(i.e, username, password) correctly for a registered user.

Unit Details: [User class(Django inbuilt), user_login() function]

Test Owner: While(1)

Test Date:28-03-2025

Test Results:

- The user was able to log in successfully on entering the correct credentials(both username, password).
- The user was not able to log in with invalid credentials.
- The user was not able to submit the login form without filling in both the credentials.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```python
def setUp(self):
    self.client = Client()
    self.login_url = reverse('login2')
    self.home_url = reverse('home')
    self.user = User.objects.create_user(
        username='testuser',
        email='test@example.com',
        password='StrongPass123'
    )
```

Initialising test database

```python
def test_get_login_page(self):
    response = self.client.get(self.login_url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'user/login.html')
    self.assertIn('login_form', response.context)
```

Test whether the view is using login.html template

```python
def test_login_successful(self):
    data = {
        'login': '1',
        'username': 'testuser',
        'password': 'StrongPass123',
    }
    response = self.client.post(self.login_url, data)
    self.assertRedirects(response, self.home_url)
```

Test whether user is able to login successfully by entering correct credentials.

```python
def test_login_nonexistent_user(self):
    data = {
        'login': '1',
        'username': 'doesnotexist',
        'password': 'whatever',
    }
    response = self.client.post(self.login_url, data)
    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertTrue(any("Login failed" in str(m) for m in messages))

def test_login_wrong_password(self):
    data = {
        'login': '1',
        'username': 'testuser',
        'password': 'wrongpassword',
    }
    response = self.client.post(self.login_url, data)
    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertTrue(any("Login failed" in str(m) for m in messages))
```

Test whether the login is failing by entering wrong credentials

```python
def test_login_empty_fields(self):
    data = {
        'login': '1',
        'username': '',
        'password': '',
    }
    response = self.client.post(self.login_url, data)
    self.assertEqual(response.status_code, 200)
    messages = list(get_messages(response.wsgi_request))
    self.assertTrue(any("Login failed" in str(m) for m in messages))
```

Test that form can’t be submitted with empty fields.

2. INVENTORY

a)ADD PRODUCT

In this unit we have tested whether a new product is getting added and the views are redirecting.

Unit Details: [Inventory class, add_inventory() function]

Test Owner: While(1)

Test Date:29-03-2025

Test Results:

- On clicking add_product, the user gets a product-Form.

- The User was able to add a new product successfully and redirect to the same page on filling all details(i.e., item ID, product name, quantity, cost price, sale price, max retail price, and GST).
- The User could not add a new product on missing any of the details(i.e., item ID, product name, quantity, cost price, sale price, max retail price, and GST).
- The user can not add a new product with an already existing item ID.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(
        username='testuser',
        password='testpass123'
    )
    self.url = reverse('add_inventory')
```

initialising test database.

```
def test_get_request_returns_form(self):
    response = self.client.get(self.url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'inventory/add_inventory.html')
    self.assertIn('form', response.context)
    self.assertEqual(response.context.get('message', ''), '')
```

Testing whether the form is displaying.

```
def test_post_valid_inventory_creates_product_and_redirects(self):
    valid_data = {
        'item_id': '12345',
        'product_name': 'Test Product',
        'quantity': 10,
        'cost_price': 50,
        'sale_price': 100,
        'max_retail_price': 120,
        'gst': 18
    }
    response = self.client.post(self.url, valid_data)
    self.assertEqual(response.status_code, 302)
    inventory_exists = Inventory.objects.filter(
        item_id='12345', user=self.user
    ).exists()
    self.assertTrue(inventory_exists)
```

Test whether it is creating a new product and redirecting to the same page on filling all details in the form.

```python
def test_post_invalid_inventory_does_not_create_product(self):
    invalid_data = {
        'item_id': '',  # Missing required value
        'product_name': '',
        'quantity': '',
        'cost_price': 50,
        'sale_price': 100,
        'max_retail_price': 120,
        'gst': 18
    }
    response = self.client.post(self.url, invalid_data)
    self.assertEqual(response.status_code, 200)
    self.assertIn('form', response.context)
    form = response.context['form']
    self.assertTrue(form.errors)
    print("Form Errors:", form.errors)
    inventory_exists = Inventory.objects.filter(
        item_id='12345', user=self.user
    ).exists()
    self.assertFalse(inventory_exists)
```

Testing that it is not creating a new product with unfilled details.

```python
def test_unique_item_id_per_user(self):
    # Create an inventory item with a specific item_id.
    Inventory.objects.create(
        user=self.user,
        item_id='12345',
        product_name='Test Product',
        quantity=10,
        cost_price=50,
        sale_price=100,
        max_retail_price=120,
        gst=18,
        total_qty_sold=0
    )
    # Attempt to create another inventory item with the same item_id.
    with self.assertRaises(IntegrityError):
        Inventory.objects.create(
            user=self.user,
            item_id='12345',
            product_name='Duplicate Product',
            quantity=5,
            cost_price=60,
            sale_price=120,
            max_retail_price=150,
            gst=18,
            total_qty_sold=0
        )
```

Test that the user cannot be able to add a new product with an already existing item ID.

b)VIEW INVENTORY

In this unit we have tested whether it is giving the list of items in inventory and on editing or deleting the item it is getting updated.

Unit Details: [Inventory class, inventory_list() function, edit_inventory() function, delete_inventory() function]

Test Owner: While(1)

Test Date:29-03-2025

Test Results:

- The user was able to get all the inventory items in the inventory list.
- The user was successfully able to search any item with its itemID or product name.
- The user was successfully able to delete any item in the inventory.
- The user was successfully able to edit details(i.e, item ID, product name, quantity, cost price, sale price, max retail price, and GST) of any item in the inventory.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.item1 = Inventory.objects.create(
        user=self.user,
        item_id="12345",
        product_name="Test Product 1",
        quantity=10,
        cost_price=50,
        sale_price=100,
        max_retail_price=120,
        gst=18,
        total_qty_sold=0
    )
    self.item2 = Inventory.objects.create(
        user=self.user,
        item_id="67890",
        product_name="Test Product 2",
        quantity=5,
        cost_price=30,
        sale_price=80,
        max_retail_price=100,
        gst=18,
        total_qty_sold=0
    )
```

initialising test database with two inventory items.

```python
def test_inventory_list_view(self):
    response = self.client.get(reverse('inventory_list'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'inventory/inventory.html')
    self.assertContains(response, "Test Product 1")
    self.assertContains(response, "Test Product 2")
```

Test whether all the items are fetching.

```python
def test_inventory_search_by_product_name(self):
    response = self.client.get(reverse('inventory_list'), {'query': 'Test Product 1'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Product 1")
    self.assertNotContains(response, "Test Product 2")


def test_inventory_search_by_item_id(self):
    response = self.client.get(reverse('inventory_list'), {'query': '67890'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Product 2")
    self.assertNotContains(response, "Test Product 1")
```

Test for search operation by product name and itemID.

```python
def test_edit_inventory(self):
    edit_url = reverse('edit_inventory', args=[self.item1.id])
    updated_data = {
        'item_id': '12345',
        'product_name': 'Updated Product',
        'quantity': 15,
        'cost_price': 55,
        'sale_price': 110,
        'max_retail_price': 130,
        'gst': 18
    }
    response = self.client.post(edit_url, updated_data)
    self.assertEqual(response.status_code, 302)
    self.item1.refresh_from_db()
    self.assertEqual(self.item1.product_name, 'Updated Product')
    self.assertEqual(self.item1.quantity, 15)
    self.assertEqual(self.item1.sale_price, 110)
```

Test that edit inventory is updating the item details.

```python
def test_delete_inventory(self):
    delete_url = reverse('delete_inventory', args=[self.item1.id])
    response = self.client.post(delete_url)
    self.assertEqual(response.status_code, 302)
    item_exists = Inventory.objects.filter(id=self.item1.id).exists()
    self.assertFalse(item_exists)
```

Test whether “delete” is deleting the item from the inventory.

3. INWARD SUPPLY

a)ADD SUPPLIER

In this unit we have checked whether a new supplier is getting added and the views are redirecting correctly.

Unit Details: [Supplier class, add_supplier function]

Test Owner: While(1)

Test Date:29-03-2025

Test Results:

- On clicking add_supplier, the user gets a SupplierForm.
- The User was able to add a new supplier successfully and redirect to the same page on filling all details(i.e., firm_name, person name, phone number, email id, address)

- The User was not able to add a new supplier on missing any of the details(i.e., firm_name, person name, phone number, email id, address)
- The user can not add a new supplier with an already existing firm name.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(
        username='testuser',
        password='testpass123'
    )
    self.url = reverse('add_supplier')
```

initialising test database.

```
def test_get_request_returns_form(self):
    response = self.client.get(self.url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'inward_supply/add_supplier.html')
    self.assertIn('form', response.context)
    self.assertEqual(response.context.get('message', ''), '')
```

Testing whether the form is displaying.

```
def test_post_valid_supplier_creates_supplier_and_redirects(self):
    valid_data = {
        'firm_name': 'Test Firm',
        'person_name': 'Test Person',
        'phone_number': '1234567890',
        'email_id': 'supplier@example.com',
        'address': '123 Test Street'
    }
    response = self.client.post(self.url, valid_data)
    self.assertEqual(response.status_code, 302)
    supplier_exists = Supplier.objects.filter(
        email_id='supplier@example.com', user=self.user).exists()
    self.assertTrue(supplier_exists)
```

Test whether it is creating a new supplier and redirecting to the same page on filling all details in the form.

```python
def test_post_invalid_supplier_does_not_create_supplier(self):
    invalid_data = {
        'firm_name': '',  # Missing required value
        'person_name': '',  # Missing required value
        'phone_number': '1234567890',
        'email_id': 'supplier@example.com',
        'address': '123 Test Street'
    }
    response = self.client.post(self.url, invalid_data)
    self.assertEqual(response.status_code, 200)
    self.assertIn('form', response.context)
    form = response.context['form']
    self.assertTrue(form.errors)
    print("Form Errors:", form.errors)
    supplier_exists = Supplier.objects.filter(
        email_id='supplier@example.com', user=self.user).exists()
    self.assertFalse(supplier_exists)
```

Testing that it is not creating a new supplier with unfilled details.

```python
def test_unique_firm_name_per_user(self):
    # Create a supplier with a specific firm name.
    supplier1 = Supplier.objects.create(
        user=self.user,
        firm_name='Test Firm',
        person_name= 'Test Person',
        phone_number= '1234567890',
        email_id= 'supplier@example.com',
        address= '123 Test Street'
    )
    #Attempt to create another supplier with the same firm name.
    with self.assertRaises(IntegrityError):
        Supplier.objects.create(
            user=self.user,
            firm_name='Test Firm',
            person_name= 'Test Person',
            phone_number= '1234567890',
            email_id= 'supplier@example.com',
            address= '123 Test Street'
        )
```

Test that the user cannot be able to add a new supplier with an already existing firm name.

b)VIEW SUPPLIERS

In this unit we have tested whether it is giving the list of suppliers and on editing or deleting the supplier it is getting updated.

Unit Details: [Supplier class, view_suppliers() function, edit_supplier() function, delete_supplier() function]

Test Owner: While(1)

Test Date: 29-03-2025

Test Results:

- The user was able to get all the suppliers in the suppliers list.

- The user was successfully able to search any supplier with its supplier name or firm name.
- The user was successfully delete any supplier in the list.
- The user was successfully able to edit details(i.e., firm_name, person name, phone number, email id, address) of any supplier.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.supplier1 = Supplier.objects.create(
        user=self.user, firm_name="Test Firm 1",
        person_name="Test Person 1",
        phone_number="1234567890",
        email_id="supplier1@example.com",
        address="123 Test Street",
        debit=100.0, total_sales=5000.0
    )
    self.supplier2 = Supplier.objects.create(
        user=self.user, firm_name="Test Firm 2",
        person_name="Test Person 2",
        phone_number="0987654321",
        email_id="supplier2@example.com",
        address="456 Sample Road",
        debit=200.0, total_sales=10000.0
    )
```

initialising test database with two suppliers.

```python
def test_view_suppliers_list(self):
    response = self.client.get(reverse('view_suppliers'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'inward_supply/view_suppliers.html')
    self.assertContains(response, "Test Firm 1")
    self.assertContains(response, "Test Firm 2")
```

Test whether all the suppliers are fetching.

```python
def test_search_suppliers_by_firm_name(self):
    response = self.client.get(reverse('view_suppliers'), {'query': 'Test Firm 1'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Firm 1")
    self.assertNotContains(response, "Test Firm 2")

def test_search_suppliers_by_person_name(self):
    response = self.client.get(reverse('view_suppliers'), {'query': 'Test Person 2'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Firm 2")
    self.assertNotContains(response, "Test Firm 1")
```

Test for search operation by firm name and supplier name.

```python
def test_edit_supplier(self):
    edit_url = reverse('edit_supplier', args=[self.supplier1.id])
    updated_data = {
        'firm_name': 'Updated Firm',
        'person_name': 'Updated Person',
        'phone_number': '9999999999',
        'email_id': 'updated@example.com',
        'address': '789 New Street'
    }
    response = self.client.post(edit_url, updated_data)
    self.assertEqual(response.status_code, 302)
    self.supplier1.refresh_from_db()
    self.assertEqual(self.supplier1.firm_name, 'Updated Firm')
    self.assertEqual(self.supplier1.person_name, 'Updated Person')
    self.assertEqual(self.supplier1.phone_number, '9999999999')
    self.assertEqual(self.supplier1.email_id, 'updated@example.com')
    self.assertEqual(self.supplier1.address, '789 New Street')
    #Ensure debit and total_sales remain unchanged
    self.assertEqual(self.supplier1.debit, 100.0)
    self.assertEqual(self.supplier1.total_sales, 5000.0)
```

Test that the edit supplier is updating the supplier details.

```python
def test_delete_supplier(self):
    """Test deleting a supplier."""
    delete_url = reverse('delete_supplier', args=[self.supplier1.id])
    response = self.client.post(delete_url)
    self.assertEqual(response.status_code, 302)
    self.assertFalse(Supplier.objects.filter(id=self.supplier1.id).exists())
```

Test that the delete supplier function is deleting the supplier.

c)ADD INWARD INVOICE BILL

In this unit we have tested whether an inward invoice bill is getting saved on filling the invoice bill form fully.

Unit Details: [Supplier class, InwardBill class, ProductEntry class, add_invoice() function]

Test Owner: While(1)

Test Date: 30-03-2025

Test Results:

- The user was able to add a new inward invoice bill successfully on filling all details(i.e. bill number, date,supplier, product, quantity).
- The user was not able to add a new bill with a duplicate bill number.
- The user was not able to add a new inward invoice bill with unfilled details(i.e., bill number, date, supplier, product, quantity).

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```python
def setUp(self):
    """Set up a test user, supplier, and two inventory products."""
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.supplier = Supplier.objects.create(
        user=self.user,firm_name="Test Supplier",
        person_name="Supplier Person",phone_number="1234567890",
        email_id="supplier@example.com",
        address="123 Test Street",
    )
    self.inventory_product1 = Inventory.objects.create(
        user=self.user, item_id="P001",
        product_name="Test Product 1", quantity=10,
        cost_price=Decimal('100.00'), sale_price=Decimal('150.00'),
        max_retail_price=Decimal('200.00'), gst=Decimal('10.00'),
    )
    self.inventory_product2 = Inventory.objects.create(
        user=self.user, item_id="P002",
        product_name="Test Product 2", quantity=20,
        cost_price=Decimal('200.00'), sale_price=Decimal('250.00'),
        max_retail_price=Decimal('300.00'), gst=Decimal('5.00'),
    )
```

initialising test database with 1 supplier and 2 inventory items.

```python
def test_get_add_invoice_page(self):
    url = reverse('add-inward-bill')
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, "inward_supply/invoice_form.html")
```

Test whether the form is displaying.

```python
def test_post_valid_invoice_creates_invoice_and_product_entries(self):
    url = reverse('add-inward-bill')
    data = {
        'bill_number': 'INV001', 'date': '2025-04-03',
        'billed-to': str(self.supplier.id),
        'product_id[]': [str(self.inventory_product1.id), str(self.inventory_product2.id)],
        'quantity[]': ['5', '3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse('invoice_list'))

    invoice = InvoiceBill.objects.filter(user=self.user, bill_number='INV001').first()
    self.assertIsNotNone(invoice)
    self.assertEqual(invoice.supplier, self.supplier)
    product_entries = ProductEntry.objects.filter(invoice=invoice)
    self.assertEqual(product_entries.count(), 2)

    expected_amount1 = ((self.inventory_product1.cost_price*Decimal(5))*
                        (1 + (self.inventory_product1.gst / Decimal(100))))
    expected_amount2 = ((self.inventory_product2.cost_price *Decimal(3))*
                        (1 + (self.inventory_product2.gst / Decimal(100))))

    pe1 = product_entries.filter(product_name=self.inventory_product1.product_name).first()
    self.assertIsNotNone(pe1)
    self.assertEqual(pe1.quantity, 5)
    self.assertEqual(Decimal(pe1.amount), expected_amount1)

    pe2 = product_entries.filter(product_name=self.inventory_product2.product_name).first()
    self.assertIsNotNone(pe2)
    self.assertEqual(pe2.quantity, 3)
    self.assertEqual(Decimal(pe2.amount), expected_amount2)

    total_invoice_amount = expected_amount1 + expected_amount2
    self.assertEqual(Decimal(invoice.get_total_amount()), total_invoice_amount)
```

Test whether the bill is saved with all fields filled.

```python
def test_post_invoice_missing_required_field_shows_error(self):
    url = reverse('add-inward-bill')
    data = {
    # 'bill_number' is intentionally omitted
    'date': '2025-04-03',
    'billed-to': str(self.supplier.id),
    'product_id[]': [str(self.inventory_product1.id), str(self.inventory_product2.id)],
    'quantity[]': ['5', '3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "This field is required", msg_prefix="Missing bill number should trigger an error")
```

Test whether the bill is not added with missing fields.

```python
def test_post_duplicate_bill_number_shows_error(self):
    url = reverse('add-inward-bill')
    InvoiceBill.objects.create(user=self.user, bill_number='INV002', date='2025-04-03')
    data = {
        'bill_number': 'INV002',
        'date': '2025-04-03',
        'product_id[]': [str(self.inventory_product1.id)],
        'quantity[]': ['3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Bill number already exists")
```

Test whether the bill is not added with a duplicate bill number.

d) INVOICE LIST

In this unit, we have tested whether all the inward invoice bills are in the list and whether the view bill is showing the bill details.

Unit Details:[Supplier class, InwardBill class, ProductEntry class, invoice_list () function, out_invoice_detail()function]

Test Owner: While(1)

Test Date:30-03-2025

Test Results:

- The user was able to see the list of all inward bills successfully.
- The user successfully got the details of a particular detail by clicking on view bill.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
def setUp(self):
    self.client = Client()
    self.user = get_user_model().objects.create_user(username='testuser', password='testpass123')
    self.client.login(username='testuser', password='testpass123')

    self.supplier = Supplier.objects.create(
        user=self.user,firm_name='Test Supplier',
        person_name='Alice', phone_number='1234567890',
        email_id='alice@gmail.com',address='123 Main St'
    )
    self.invoice1 = InvoiceBill.objects.create(
        user=self.user,
        date='2025-04-03',
        bill_number='INV100',
        supplier=self.supplier,
    )
    self.invoice2 = InvoiceBill.objects.create(
        user=self.user,
        date='2025-04-04',
        bill_number='INV200',
        supplier=self.supplier,
    )
    ProductEntry.objects.create(invoice=self.invoice1, product_name='Product A', quantity=4, amount=40.0)
    ProductEntry.objects.create(invoice=self.invoice1, product_name='Product B', quantity=2, amount=30.0)
    ProductEntry.objects.create( invoice=self.invoice2, product_name='Product C', quantity=5, amount=75.0)
```

initialising test database with 1 supplier and 2 inward invoice bills

```python
def test_invoice_list_shows_all_user_invoices(self):
    url = reverse('invoice_list')
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, "inward_supply/view_bill.html")

    self.assertContains(response, 'INV100')
    self.assertContains(response, 'INV200')
    self.assertContains(response, self.supplier.firm_name)
```

Test whether all the inward invoice bills are showing in the list.

```python
def test_invoice_detail_shows_correct_invoice_and_products(self):
    url = reverse('invoice_detail', args=[self.invoice1.bill_number])
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, "inward_supply/invoice_detail.html")

    self.assertContains(response, self.invoice1.bill_number)
    self.assertContains(response, self.supplier.firm_name)

    self.assertContains(response, 'Product A')
    self.assertContains(response, 'Product B')
    self.assertNotContains(response, 'Product C')
```

Test whether the details of the inward invoice bill are showing on clicking view bill.

4. OUTWARD SUPPLY

a)ADD RETAILER

In this unit we have checked whether a new retailer is getting added and the views are redirecting correctly.

Unit Details: [Supplier class, add_supplier function]

Test Owner: While(1)

Test Date:29-03-2025

Test Results:

- On clicking add_retailer, the user gets a retailer form.
- The User was able to add a new retailer successfully and redirect to the same page on filling all details(i.e., firm_name, person name, phone number, email id, address)
- The User was not able to add a new retailer on missing any of the details(i.e., firm_name, person name, phone number, email id, address)
- The user can not add a new retailer with an already existing firm name.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(
        username='testuser',
        password='testpass123'
    )
    self.url = reverse('add_retailer')
```

initialising test database.

```
def test_get_request_returns_form(self):
    response = self.client.get(self.url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'outward_supply/add_retailer.html')
    self.assertIn('form', response.context)
    self.assertEqual(response.context.get('message', ''), '')
```

Testing whether the form is displaying.

```
def test_post_valid_retailer_creates_retailer_and_redirects(self):
    valid_data = {
        'firm_name': 'Test Retailer',
        'person_name': 'Retailer Person',
        'phone_number': '1234567890',
        'email_id': 'retailer@example.com',
        'address': '456 Retail Street'
    }
    response = self.client.post(self.url, valid_data)
    self.assertEqual(response.status_code, 200)
    retailer_exists = Retailer.objects.filter(
        email_id='retailer@example.com', user=self.user).exists()
    self.assertTrue(retailer_exists)
```

Testing whether it is creating new retailer and redirecting to the same page on filling all details in the form.

```python
def test_post_invalid_retailer_does_not_create_retailer(self):
    invalid_data = {
        'firm_name': '',  # Missing required field
        'person_name': '',  # Missing required field
        'phone_number': '1234567890',
        'email_id': 'retailer@example.com',
        'address': '456 Retail Street'
    }
    response = self.client.post(self.url, invalid_data)
    self.assertEqual(response.status_code, 200)
    self.assertIn('form', response.context)
    form = response.context['form']
    self.assertTrue(form.errors)
    print("Form Errors:", form.errors)
    retailer_exists = Retailer.objects.filter(
        email_id='retailer@example.com', user=self.user).exists()
    self.assertFalse(retailer_exists)
```

Testing that its is not creating a new supplier with unfilled details.

```python
def test_unique_firm_name_per_user(self):
    #Create a retailer with a specific firm name.
    Retailer.objects.create(
        user=self.user,
        firm_name='Test Retailer',
        person_name='Retailer Person',
        phone_number='1234567890',
        email_id='retailer@example.com',
        address='456 Retail Street'
    )
    #Attempt to create another retailer with the same firm name.
    with self.assertRaises(IntegrityError):
        Retailer.objects.create(
            user=self.user,
            firm_name='Test Retailer',
            person_name='Retailer Person',
            phone_number='1234567890',
            email_id='retailer2@example.com',
            address='789 Another Street'
        )
```

Test that the user cannot be able to add a new retailer with an already existing firm name.

b)VIEW RETAILERS

In this unit we have tested whether it is giving the list of retailers and on editing or deleting the retailer it is getting updated.

Unit Details: [Retailer class, view_retailers() function, edit_retailer() function, delete_retailer() function]

Test Owner: While(1)

Test Date:29-03-2025

Test Results:

- The user was able to get all the retailers in the retailers list.
- The user was successfully able to search any retailer with its retailer name or firm name.
- The user was successfully delete any retailer in the list.
- The user was successfully able to edit details(i.e., firm_name, person name, phone number, email id, address) of any retailer.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
def setUp(self):
    """Set up a test user and sample retailers."""
    self.user = get_user_model().objects.create_user(
        username='testuser',
        password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.retailer1 = Retailer.objects.create(
        user=self.user, firm_name="Test Retailer 1",
        person_name="Retailer Person 1", phone_number="1111111111",
        email_id="retailer1@example.com", address="111 Test Street"
    )
    self.retailer2 = Retailer.objects.create(
        user=self.user, firm_name="Test Retailer 2",
        person_name="Retailer Person 2", phone_number="2222222222",
        email_id="retailer2@example.com", address="222 Sample Road"
    )
```

initialising test database with two retailers.

```python
def test_view_retailers_list(self):
    response = self.client.get(reverse('view_retailers'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'outward_supply/view_retailers.html')
    self.assertContains(response, "Test Retailer 1")
    self.assertContains(response, "Test Retailer 2")
```

Test whether all the retailers are fetching.

```python
def test_search_retailers_by_firm_name(self):
    response = self.client.get(reverse('view_retailers'), {'query': 'Test Retailer 1'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Retailer 1")
    self.assertNotContains(response, "Test Retailer 2")

def test_search_retailers_by_person_name(self):
    response = self.client.get(reverse('view_retailers'), {'query': 'Retailer Person 2'})
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Test Retailer 2")
    self.assertNotContains(response, "Test Retailer 1")
```

Test for search operation by firm name and retailer name.

```python
def test_edit_retailer(self):
    edit_url = reverse('edit_retailer', args=[self.retailer1.id])
    updated_data = {
        'firm_name': 'Updated Retailer',
        'person_name': 'Updated Person',
        'phone_number': '8888888888',
        'email_id': 'updatedretailer@example.com',
        'address': '888 Updated Street'
    }
    response = self.client.post(edit_url, updated_data)
    self.assertEqual(response.status_code, 302)
    self.retailer1.refresh_from_db()
    self.assertEqual(self.retailer1.firm_name, 'Updated Retailer')
    self.assertEqual(self.retailer1.person_name, 'Updated Person')
    self.assertEqual(self.retailer1.phone_number, '8888888888')
    self.assertEqual(self.retailer1.email_id, 'updatedretailer@example.com')
    self.assertEqual(self.retailer1.address, '888 Updated Street')
```

Test that the edit retailer is updating the retailer details.

```python
def test_delete_retailer(self):
    """Test deleting a retailer."""
    delete_url = reverse('delete_retailer', args=[self.retailer1.id])
    response = self.client.post(delete_url)
    self.assertEqual(response.status_code, 302)
    self.assertFalse(Retailer.objects.filter(id=self.retailer1.id).exists())
```

Test that the delete retailer is deleting the retailer.

c)ADD OUTWARD INVOICE BILL

In this unit we have tested whether outward invoice bill is getting saved on filling the invoice bill form fully.

Unit Details: [Retailer class, Outward_Invoice class, ProductEntry class, add_out_invoice() function]

Test Owner: While(1)

Test Date:30-03-2025

Test Results:

- The user was able to add new outward invoice bill successfully on filling all details(i.e. bill number, date,retailer, product, quantity).
- The user was not able to add a new bill with a duplicate bill number.
- The user was not able to add new outward invoice bill with unfilled details(i.e. bill number, date,retailer, product, quantity).

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```python
def setUp(self):
    """Set up a test user, retailer, and two inventory products."""
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.retailer = Retailer.objects.create(
        user=self.user, firm_name="Test Retailer",
        person_name="Retailer Person", phone_number="1234567890",
        email_id="retailer@example.com",
        address="456 Retailer Street"
    )
    self.inventory_product1 = Inventory.objects.create(
        user=self.user, item_id="P001",
        product_name="Test Product 1", quantity=10,
        cost_price=Decimal('150.00'), sale_price=Decimal('200.00'),
        max_retail_price=Decimal('250.00'),
        gst=Decimal('10.00'),
    )
    self.inventory_product2 = Inventory.objects.create(
        user=self.user, item_id="P002",
        product_name="Test Product 2", quantity=20,
        cost_price=Decimal('250.00'), sale_price=Decimal('300.00'),
        max_retail_price=Decimal('350.00'),
        gst=Decimal('5.00'),
    )
```

initialising test database with 1 retailer and 2 products.

```python
def test_get_add_out_invoice_page(self):
    url = reverse('add-outward-bill')
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, "outward_supply/add_outward_invoice.html")
```

Test whether the form is displaying.

```python
def test_post_valid_out_invoice_creates_invoice_and_product_entries(self):
    url = reverse('add-outward-bill')
    data = {
        'bill_number': 'INV100',
        'date': '2025-04-03',
        'billed-to': str(self.retailer.id),
        'discount': '10',
        'product_id[]': [str(self.inventory_product1.id), str(self.inventory_product2.id)],
        'quantity[]': ['5', '3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse('out_invoice_list'))
    invoice = Outward_Invoice.objects.filter(user=self.user, bill_number='INV100').first()
    self.assertIsNotNone(invoice)
    self.assertEqual(invoice.retailer, self.retailer)
    product_entries = ProductEntry.objects.filter(invoice=invoice)
    self.assertEqual(product_entries.count(), 2)

    expected_amount1 = ((self.inventory_product1.sale_price * Decimal(5))*
                        (1 + (self.inventory_product1.gst / Decimal(100))))
    expected_amount2 = ((self.inventory_product2.sale_price * Decimal(3))*
                        (1 + (self.inventory_product2.gst / Decimal(100))))

    pe1 = product_entries.filter(product_name=self.inventory_product1.product_name).first()
    self.assertIsNotNone(pe1)
    self.assertEqual(pe1.quantity, 5)
    self.assertEqual(Decimal(pe1.amount), expected_amount1)
    pe2 = product_entries.filter(product_name=self.inventory_product2.product_name).first()
    self.assertIsNotNone(pe2)
    self.assertEqual(pe2.quantity, 3)
    self.assertEqual(Decimal(pe2.amount), expected_amount2)
```

Test whether the bill is saving with all fields filled.

```python
def test_post_duplicate_bill_number_shows_error(self):
    url = reverse('add-outward-bill')
    # Pre-create an invoice with a specific bill number
    Outward_Invoice.objects.create(user=self.user, bill_number='INV200', date='2025-04-03', discount=0)
    data = {
        'bill_number': 'INV100',
        'date': '2025-04-03',
        'discount': '0',
        'billed-to': str(self.retailer.id),
        'product_id[]': [str(self.inventory_product1.id)],
        'quantity[]': ['3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "Bill number already exists")
```

Test whether the bill is not added with a duplicate bill number.

```python
def test_post_out_invoice_missing_required_field_shows_error(self):
    url = reverse('add-outward-bill')
    data = {
        # 'bill_number' is intentionally omitted
        'date': '2025-04-03',
        'discount': '5',
        'billed-to': str(self.retailer.id),
        'product_id[]': [str(self.inventory_product1.id), str(self.inventory_product2.id)],
        'quantity[]': ['5', '3']
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, 200)
    self.assertContains(response, "This field is required", msg_prefix="Missing bill number should trigger an error")
    self.assertEqual(Outward_Invoice.objects.count(), 0, "Invoice should not be created if required fields are missing.")
```

Test whether the bill is not added with missing fields.

d)OUT INVOICE LIST

In this unit we have tested whether all the outward invoice bills are in the list and whether the view bill is showing the bill details.

Unit Details:[Retailer class, Outward_Invoice class, ProductEntry class, out_invoice_list () function, out_invoice_detail()function]

Test Owner: While(1)

Test Date:30-03-2025

Test Results:

- The user was able to see the list of all outward bills successfully.
- The user successfully got the details of a particular detail by clicking on view bill.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
class OutwardInvoiceListViewTests(TestCase):
    def setUp(self):
        self.client = Client()
        self.user = get_user_model().objects.create_user(username='testuser', password='testpass123')
        self.client.login(username='testuser', password='testpass123')
        self.retailer = Retailer.objects.create(
            user=self.user, firm_name='Test Firm',
            person_name='John Doe', phone_number='9876543210',
            email_id='john@example.com', address='123 Market Street',
        )
        self.invoice1 = Outward_Invoice.objects.create(
            user=self.user, date='2025-04-03',
            bill_number='INV001', retailer=self.retailer,
            discount=10.0,
            profit=5.0,
        )
        self.invoice2 = Outward_Invoice.objects.create(
            user=self.user, date='2025-04-03',
            bill_number='INV002', retailer=self.retailer,
            discount=5.0,
            profit=3.0,
        )
        ProductEntry.objects.create(invoice=self.invoice1, product_name='PRODUCT A', quantity=10, amount=50.0)
        ProductEntry.objects.create(invoice=self.invoice1, product_name='PRODUCT B', quantity=5, amount=25.0)
        ProductEntry.objects.create(invoice=self.invoice2, product_name='PRODUCT C', quantity=8, amount=32.0)
```

initialising test database with 1 retailer, 2 outward invoice bills.

```python
def test_invoice_list_shows_all_user_invoices_with_details(self):
    response = self.client.get(reverse('out_invoice_list'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, "outward_supply/outward_invoice_list.html")

    self.assertContains(response, 'INV001')
    self.assertContains(response, 'INV002')
    self.assertContains(response, self.retailer.firm_name)
```

Test whether the list contains all outward invoice bills.

```python
def test_out_invoice_detail_shows_correct_invoice_and_products(self):
    self.client.login(username='testuser', password='testpass')
    url = reverse('out_invoice_detail', args=[self.invoice1.bill_number])
    response = self.client.get(url)

    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'outward_supply/out_invoice_detail.html')
    self.assertContains(response, self.invoice1.bill_number)
    self.assertNotContains(response, self.invoice2.bill_number)
    self.assertContains(response, 'PRODUCT A')
    self.assertContains(response, 'PRODUCT B')
    self.assertNotContains(response, 'PRODUCT C')
    self.assertContains(response, self.retailer.firm_name)
```

Test whether the view bill giving the bill details correctly.

5. TRANSACTIONS

a)ADD INWARD TRANSACTION

In this unit, we have checked whether a new inward transaction is getting added and the views are redirecting correctly.

Unit Details: [Transaction class, update_transaction_supplier() function]

Test Owner: While(1)

Test Date:31-03-2025

Test Results:

- On clicking add_inward_transaction, the user gets an add_inward_transaction form.
- The User was able to add a new inward transaction(s) successfully and redirect to the view_transaction_history page.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```python
def setUp(self):
    """Set up a test user and sample suppliers."""
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.supplier1 = Supplier.objects.create(
        user=self.user,
        firm_name="Supplier 1",
        person_name="Person 1",
        phone_number="1111111111",
        email_id="supplier1@example.com",
        address="111 Test Street",
        debit=500.0,
        total_sales=1000.0
    )
    self.supplier2 = Supplier.objects.create(
        user=self.user,
        firm_name="Supplier 2",
        person_name="Person 2",
        phone_number="2222222222",
        email_id="supplier2@example.com",
        address="222 Sample Road",
        debit=800.0,
        total_sales=2000.0
    )
```

initialising test database with two suppliers.

```python
def test_view_add_transaction_page(self):
    response = self.client.get(reverse('update_transaction_supplier'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'transactions/add_inward_transaction.html')
```

Test whether the user is getting the add_inward_transaction form.

```python
def test_update_transaction_supplier(self):
    update_url = reverse('update_transaction_supplier')
    data = {
        'firm_name[]': ['Supplier 1', 'Supplier 2'],
        'amount_paid[]': ['100', '200'],
        'remarks[]': ['Paid in cash', 'Bank transfer'],
        'add_date': '2025-04-03'
    }
    response = self.client.post(update_url, data)
    self.assertRedirects(response, reverse('view_transaction_history'))

    transactions = Transaction.objects.filter(user=self.user, add_date='2025-04-03')
    self.assertEqual(transactions.count(), 2)

    trans_supplier1 = transactions.filter(firm_name="Supplier 1").first()
    self.assertIsNotNone(trans_supplier1)
    self.assertEqual(trans_supplier1.payment, 100.0)
    self.assertEqual(trans_supplier1.remarks, 'Paid in cash')
    self.assertEqual(trans_supplier1.person_name, 'Person 1')

    trans_supplier2 = transactions.filter(firm_name="Supplier 2").first()
    self.assertIsNotNone(trans_supplier2)
    self.assertEqual(trans_supplier2.payment, 200.0)
    self.assertEqual(trans_supplier2.remarks, 'Bank transfer')
    self.assertEqual(trans_supplier2.person_name, 'Person 2')
```

Test whether the transaction is saving successfully

.

b)ADD OUTWARD TRANSACTION

In this unit, we have checked whether a new outward transaction is getting added and if the views are redirecting correctly.

Unit Details: [Transaction class, update_transaction_retailer() function]

Test Owner: While(1)

Test Date:31-03-2025

Test Results:

- On clicking add_outward_transaction, the user gets an add_outward_transaction form.
- The User was able to add a new inward transaction(s) successfully and redirect to the view_transaction_history page.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness by testing both valid and invalid inputs.

```
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.retailer1 = Retailer.objects.create(
        user=self.user,
        firm_name="Retailer 1",
        person_name="Retailer Person 1",
        phone_number="3333333333",
        email_id="retailer1@example.com",
        address="333 Retailer Street",
        credit=1000.0
    )
    self.retailer2 = Retailer.objects.create(
        user=self.user,
        firm_name="Retailer 2",
        person_name="Retailer Person 2",
        phone_number="4444444444",
        email_id="retailer2@example.com",
        address="444 Retailer Avenue",
        credit=1500.0
    )
```

initialising test database with two retailers.

```
def test_view_update_transaction_retailer_page(self):
    response = self.client.get(reverse('update_transaction_retailer'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'transactions/pending.html')
```

Test whether the user is getting the add_outward_transaction form.

```python
def test_update_transaction_retailer(self):
    update_url = reverse('update_transaction_retailer')
    data = {
        'firm_name[]': ['Retailer 1', 'Retailer 2'],
        'amount_paid[]': ['150', '250'],
        'remarks[]': ['Payment A', 'Payment B'],
        'add_date': '2025-04-03'
    }
    response = self.client.post(update_url, data)
    self.assertRedirects(response, reverse('view_transaction_history'))

    transactions = Transaction.objects.filter(user=self.user, add_date='2025-04-03')
    self.assertEqual(transactions.count(), 2)

    trans_retailer1 = transactions.filter(firm_name="Retailer 1").first()
    self.assertIsNotNone(trans_retailer1)
    self.assertEqual(trans_retailer1.payment, 150.0)
    self.assertEqual(trans_retailer1.remarks, 'Payment A')
    self.assertEqual(trans_retailer1.person_name, 'Retailer Person 1')
    self.assertEqual(trans_retailer1.type, 1)

    trans_retailer2 = transactions.filter(firm_name="Retailer 2").first()
    self.assertIsNotNone(trans_retailer2)
    self.assertEqual(trans_retailer2.payment, 250.0)
    self.assertEqual(trans_retailer2.remarks, 'Payment B')
    self.assertEqual(trans_retailer2.person_name, 'Retailer Person 2')
    self.assertEqual(trans_retailer2.type, 1)
```

Test whether the transaction is saving successfully.

c)TRANSACTION HISTORY

In this unit, we have checked whether the transaction history is fetching all the transactions.

Unit Details: [Transaction class, view_transaction_history() function]

Test Owner: While(1)
Test Date: 31-03-2025
Test Results:

- On clicking transaction history it is showing all the inward transactions and outward transactions(monetary) successfully.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```python
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.supplier = Supplier.objects.create(
        user=self.user,
        firm_name="Test Supplier",person_name="Supplier Person",
        phone_number="1111111111",email_id="supplier@example.com",
        address="Supplier Address",debit=100.0,total_sales=500.0
    )
    self.retailer = Retailer.objects.create(
        user=self.user, firm_name="Test Retailer", person_name="Retailer Person",
        phone_number="2222222222",email_id="retailer@example.com",
        address="Retailer Address", credit=1000.0
    )
    self.supplier_transaction = Transaction.objects.create(
        user=self.user,firm_name="Test Supplier",
        person_name="Supplier Person",add_date="2025-04-03",
        payment=50.0,remarks="Supplier transaction",type=0  # Supplier transaction
    )
    self.retailer_transaction = Transaction.objects.create(
        user=self.user,firm_name="Test Retailer",
        person_name="Retailer Person",add_date="2025-04-03",
        payment=75.0,remarks="Retailer transaction",type=1  # Retailer transaction
    )
```

initialising database with 1 supplier, 1 retailer,1 inward transaction and 1 outward transaction.

```python
def test_view_transaction_history(self):
    url = reverse('view_transaction_history')
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'transactions/view_transaction_history.html')

    self.assertIn(self.supplier, response.context['suppliers'])
    self.assertIn(self.retailer, response.context['retailers'])
    transactions = response.context['transactions']

    self.assertEqual(transactions.count(), 2)
    self.assertIn(self.supplier_transaction, transactions)
    self.assertIn(self.retailer_transaction, transactions)

    supplier_trans = transactions.filter(firm_name="Test Supplier").first()
    retailer_trans = transactions.filter(firm_name="Test Retailer").first()
    self.assertIsNotNone(supplier_trans)
    self.assertIsNotNone(retailer_trans)
    self.assertEqual(supplier_trans.payment, 50.0)
    self.assertEqual(supplier_trans.remarks, "Supplier transaction")
    self.assertEqual(retailer_trans.payment, 75.0)
    self.assertEqual(retailer_trans.remarks, "Retailer transaction")
```

Test whether the transaction history is fetching all the transactions and the view_transaction_history page is directed

.d)PENDING TRANSACTIONS

In this unit, we have checked whether all the pending transactions are fetched.

Unit Details: [Transaction class, pending() function]

Test Owner: While(1)

Test Date:31-03-2025

Test Results:

- On clicking transaction history it is showing all the suppliers and retailers with pending transactions successfully.

Structural Coverage: Statement and Branch Coverage were used to ensure test completeness.

```
def setUp(self):
    self.user = get_user_model().objects.create_user(
        username='testuser', password='testpass123'
    )
    self.client.login(username='testuser', password='testpass123')
    self.supplier1 = Supplier.objects.create(
        user=self.user,
        firm_name="Supplier 1",
        person_name="Person 1",
        phone_number="1111111111",
        email_id="supplier1@example.com",
        address="111 Test Street",
        debit=500.0,
        total_sales=1000.0
    )
    self.retailer1 = Retailer.objects.create(
        user=self.user,
        firm_name="Retailer 1",
        person_name="Retailer Person 1",
        phone_number="2222222222",
        email_id="retailer1@example.com",
        address="222 Sample Road",
        credit=300.0,
        total_sales=1500.0
    )
```

initialising test database with 1 supplier and 1 retailer.

```
def test_pending_view(self):
    response = self.client.get(reverse('pending'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'transactions/pending.html')

    response = self.client.get(reverse('pending'))
    self.assertIn(self.supplier1, response.context['suppliers'])
    self.assertIn(self.retailer1, response.context['retailers'])
```

Test whether it is using pending _transaction template and fetching all the suppliers and retailers.

3 Integration Testing

3.1 Authentication

3.1.1 Registering a User

Module Details: This module integrates the registration process for new users, providing the necessary functionalities to create a new account.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

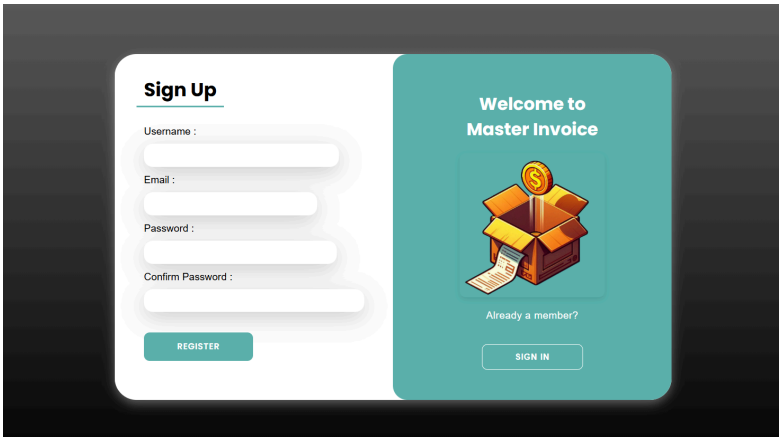
- **Test case-1:** An Invalid username, password are entered.
  Result :

1)When a new user enters a username that is already used by a previous user, it displays an error message: " A user with that username already exists."

2)When a user enters a password that is too common, the system displays a message "the password is too common" or when the username and password have most of the same characters from username or email it displays "The password is too similar to the username." or "The password is too similar to the email."

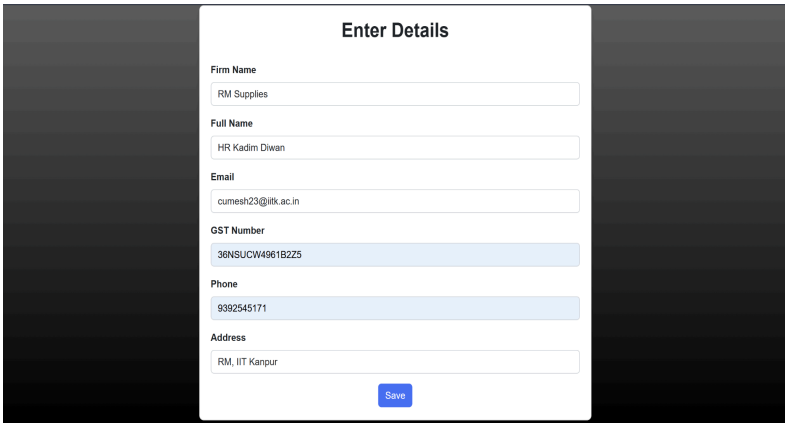
3) If an invalid email address format is entered, it displays an error message like "Please include '@' in the email address".

- **Test case-2:** After entering a valid username, password, and email address.



It is then redirected to Verify OTP

Then it is redirected to the Enter details page.



Then the user is added successfully and is redirected to the home page.
It is even updated in the database

superuser	username	first_name	last_name	email	is_staff	is_active	date_j
[ ]	goutham			kavurigouthamchandra@gmail.com	[ ]	[v]	2025-04-0
[ ]	ckund			ckundan999@gmail.com	[ ]	[v]	2025-04-0
[ ]	User2			skoushik23@iitk.ac.in	[ ]	[v]	2025-04-0
[ ]	obul			voreddy23@iitk.ac.in	[ ]	[v]	2025-04-0
[ ]	230895			c.kundansai@gmail.com	[ ]	[v]	2025-04-0
[ ]	User1			saikoushik.chundi@gmail.com	[ ]	[v]	2025-04-0
[ ]	umesh			umeshvarunchalla@gmail.com	[ ]	[v]	2025-04-0
[ ]	User0			cumesh23@iitk.ac.in	[ ]	[v]	2025-04-0
[ ]	kishores23			kishore200519@gmail.com	[ ]	[v]	2025-04-0

The profile database is also updated

	9392545171	RM Supplies	HR Kadim Diwan	RM, IIT Kanpur	9	36NSUCW4961B2Z5

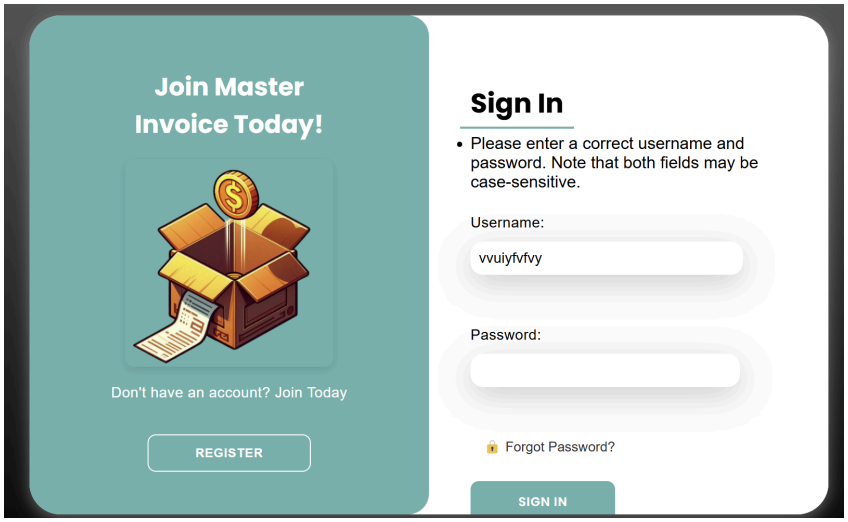
Result: When the user enters it directs to the profile filling page otherwise it is showing an error message about an incorrect password.

Additional Comments: Resend OTP works only after the time limit of 4 minutes is up (during OTP verification).

3.1.2 Logging in a User

Module Details: This module is the integration of all the components of the login page.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

- **Test case-1:** If the user has not signed up and attempts to log in with an unregistered username and password.
  Result: It displays the message "Please enter a correct username and password. Note that both fields may be case-sensitive."



Additional Comments:

3.1.3 Forgot Password

Module Details: This module integrates all the components necessary for the forgot password password page.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

- **Test case-1:** Checking whether the forgot password feature is working or not.
  Result: The OTP is sent to the email, and the password can be reset, and the user can re-log in.

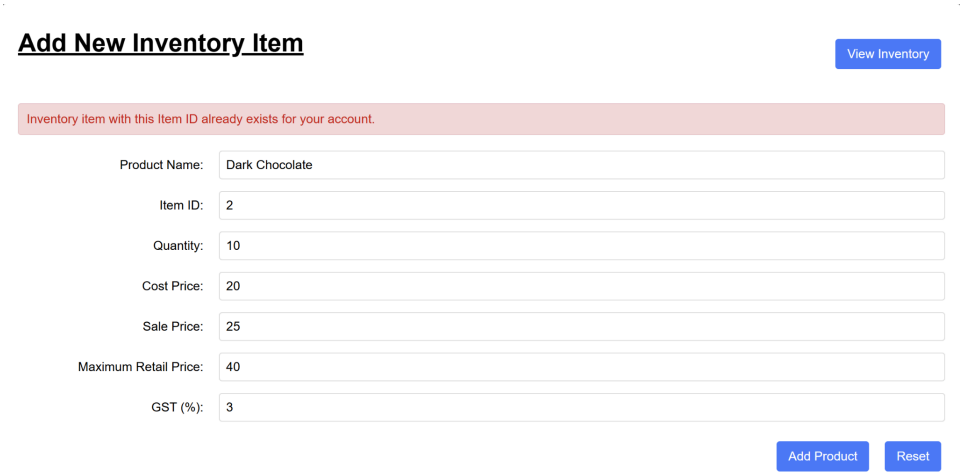
Additional Comments: The CSS of this page is not done completely.

3.2 Inventory

3.2.1 Add Inventory

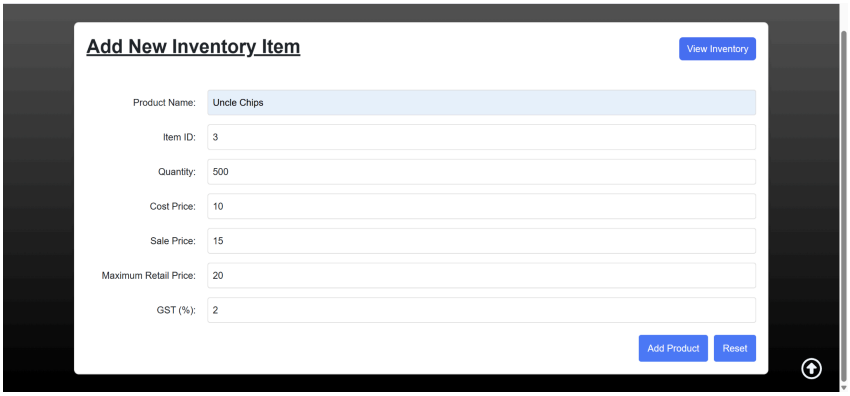
Module Details: This module integrates all the components necessary for adding a product.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

- **Test case-1:** Adding item with the same item ID.
  Result: It displays the message “Inventory item with this Item ID already exists for your account”.

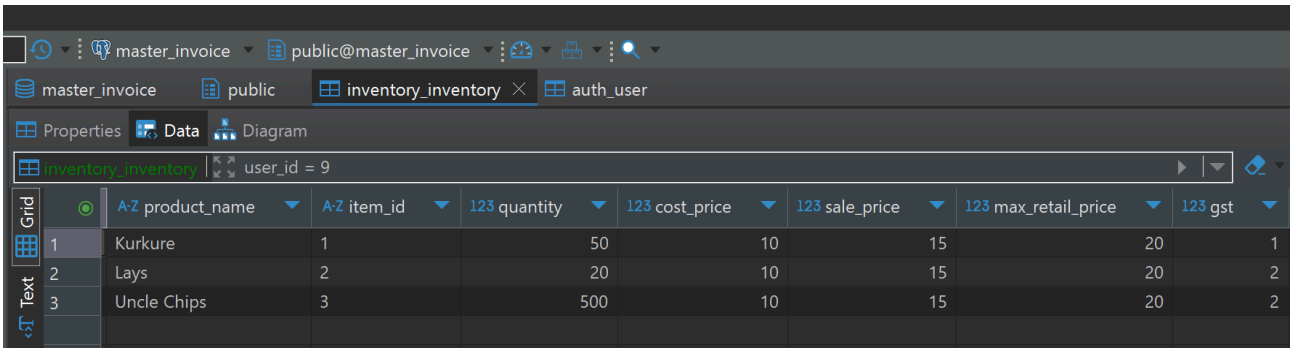


(we are adding again Item id '2')

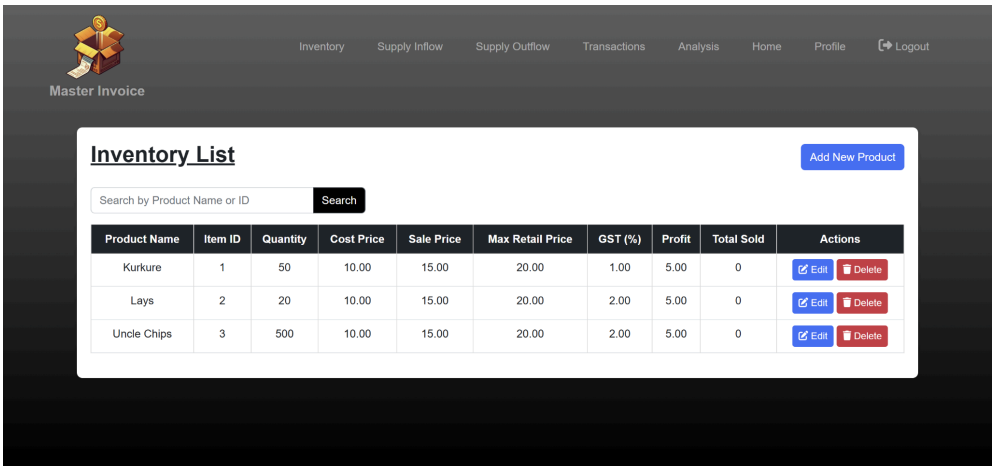
- **Test case-2:** Filling valid data and adding an item.
  Result: It displays the message "Inventory item added successfully!"



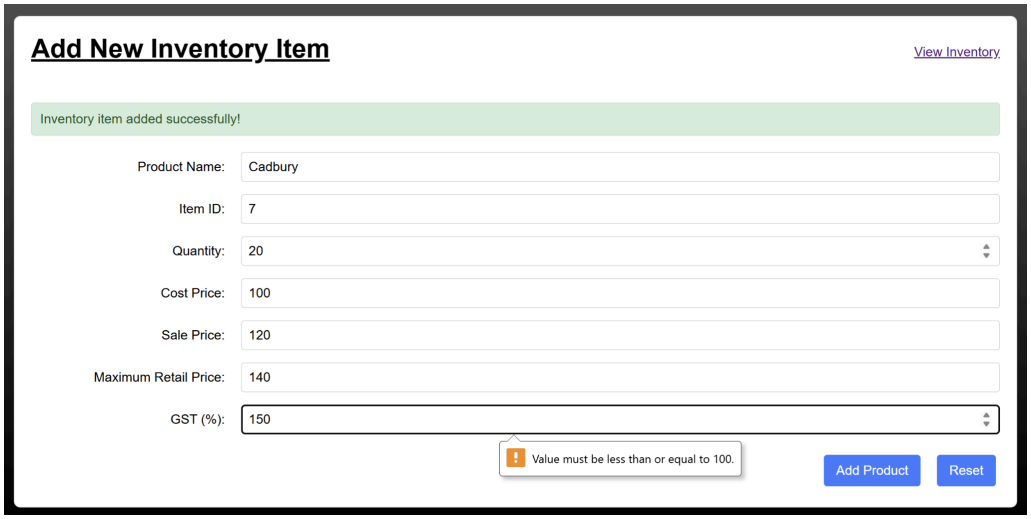
The item is getting updated in the database also



Even the inventory(front end) is getting updated



- **Test case-3:** Adding items with GST greater than 100 percent.
  Result: It displays the message "Value must be less than or equal to 100".

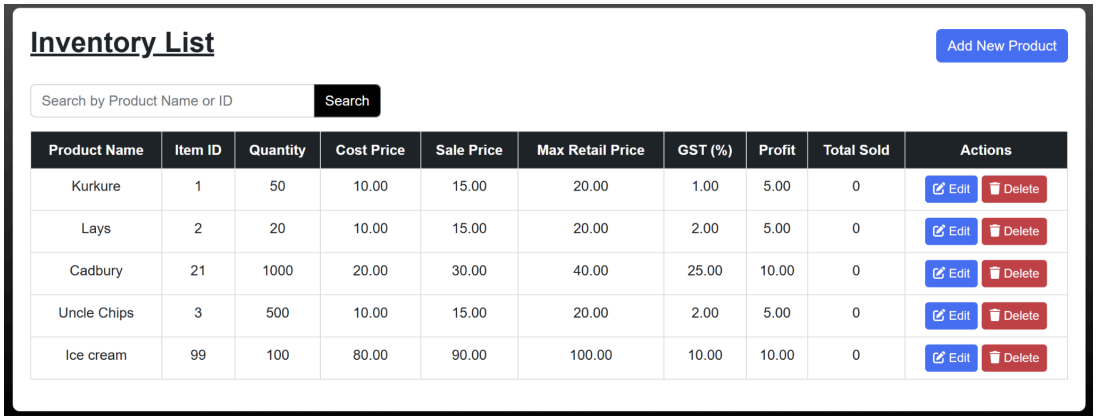


Additional Comments: N/A

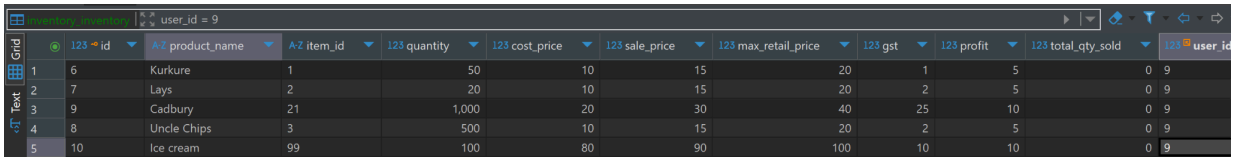
3.2.2 View Inventory

Module Details: This module integrates all the components necessary for viewing inventory.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

- **Test case-1:** To verify whether all the products in the inventory of a particular user are fetched correctly or not.

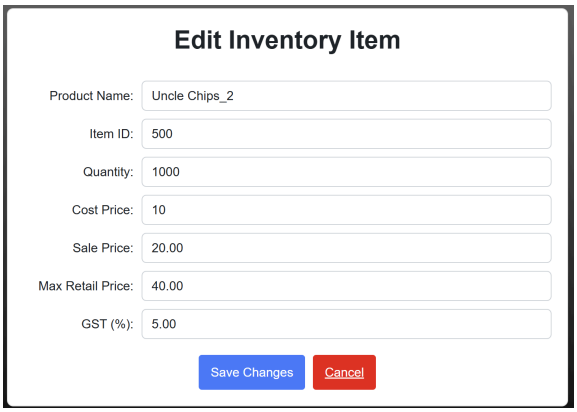


Now verifying with the database

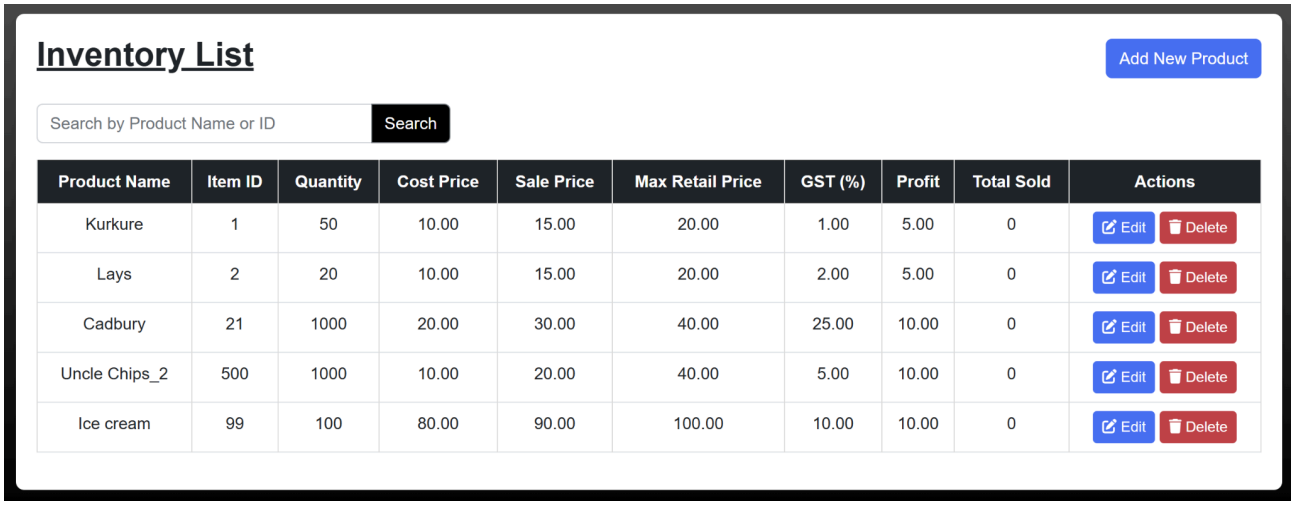


Result: All the products are fetched correctly

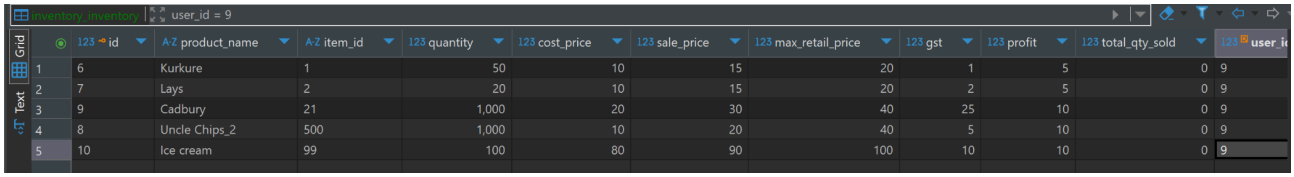
- **Test case-2:** To verify the feature of updating the details of an item in the inventory.



Now verifying with the view inventory(front end) and the backend.

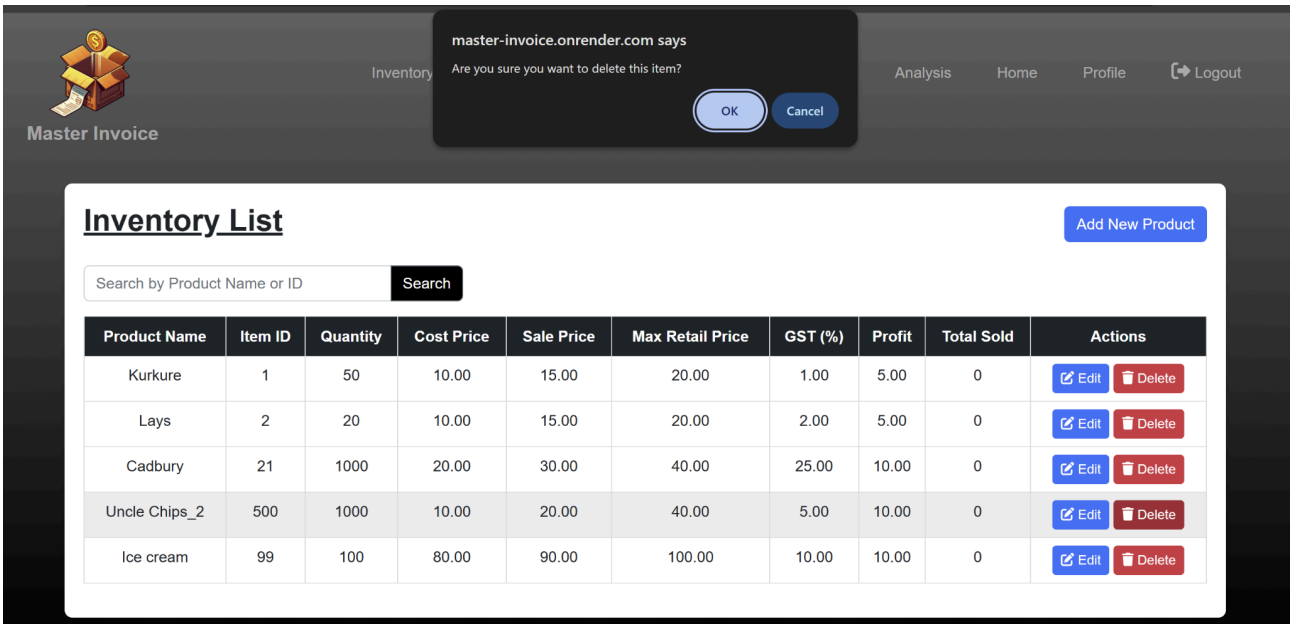


The database:

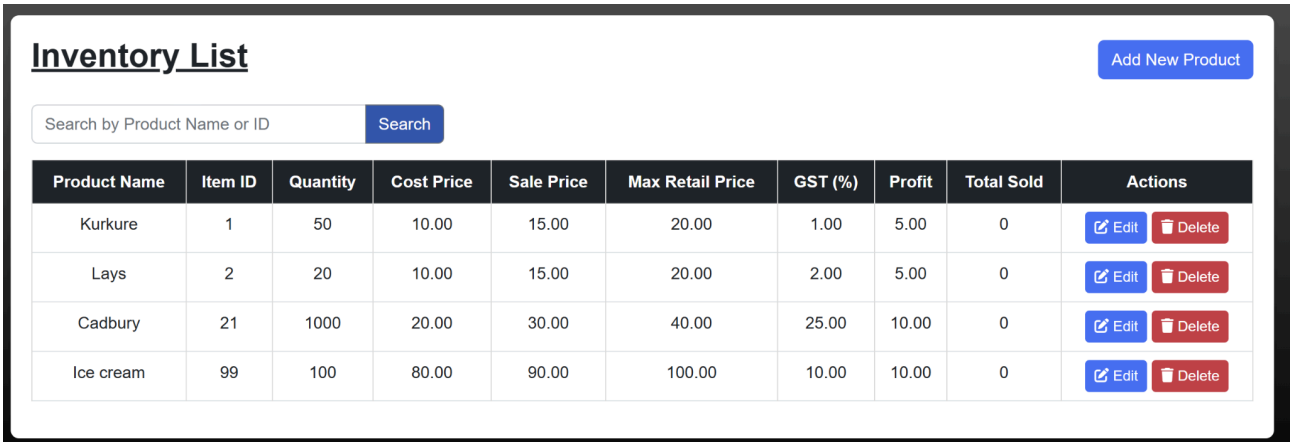


Result: The product details are updated successfully.

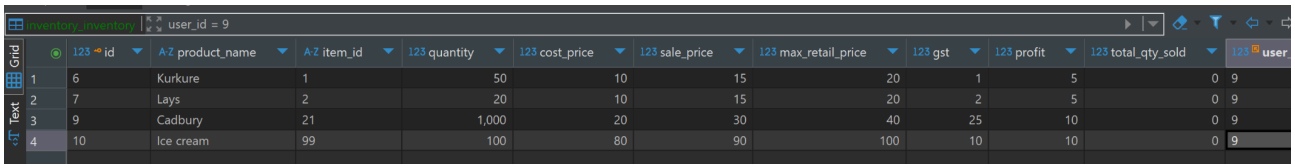
- **Test case-3:** Deleting the item from the inventory
  (Here we are now deleting the Uncle_chips_2 just by pressing the button delete)



Inventory after deletion(frontend):



Database(backend) after deletion:



Result: First, we get an alert message confirming the deletion, and the product is deleted successfully.

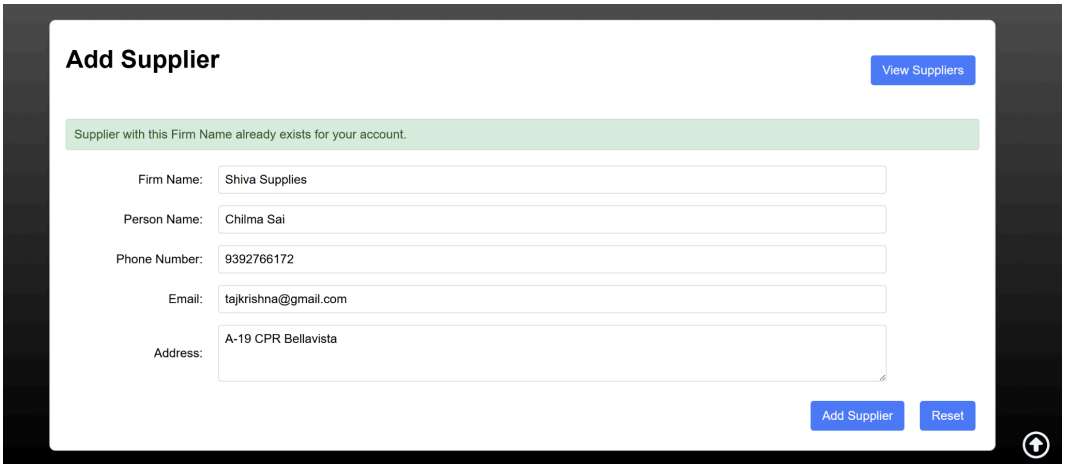
Additional Comments: N/A

3.3 Inward Supply

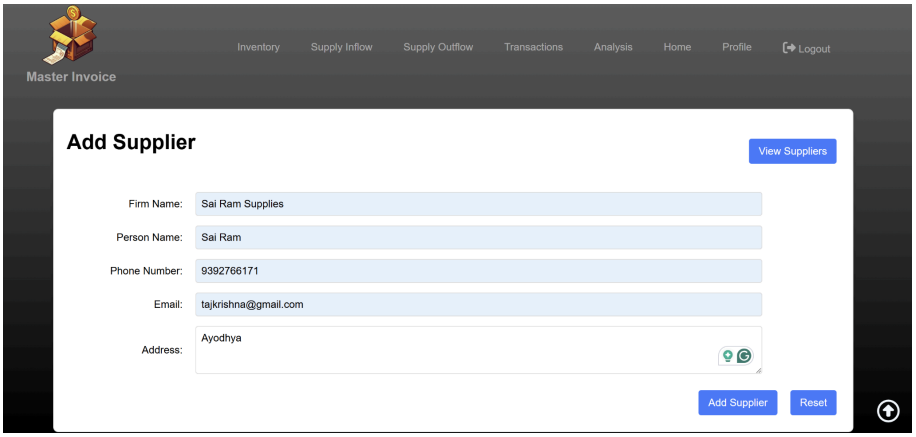
3.3.1 Add Supplier

Module Details: This module integrates all the components of the add supplier page.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

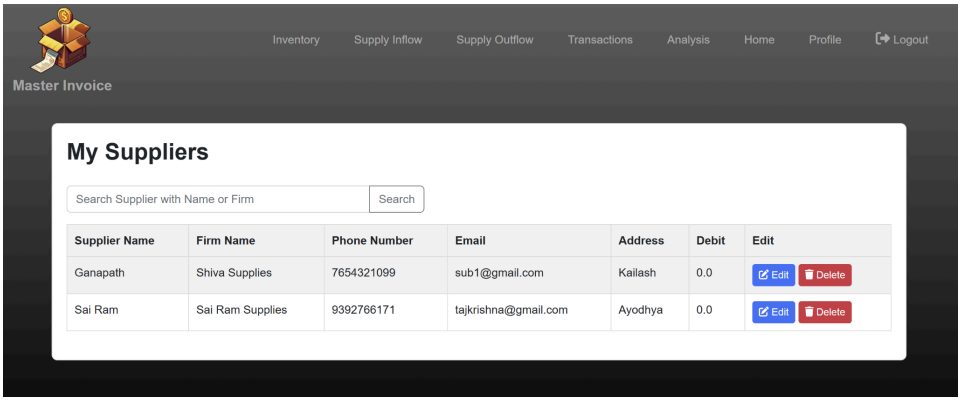
- **Test case-1:** Adding a supplier with the same Firm Name.
  Result: It displays the message "Supplier with same Firm Name already exists for your account."



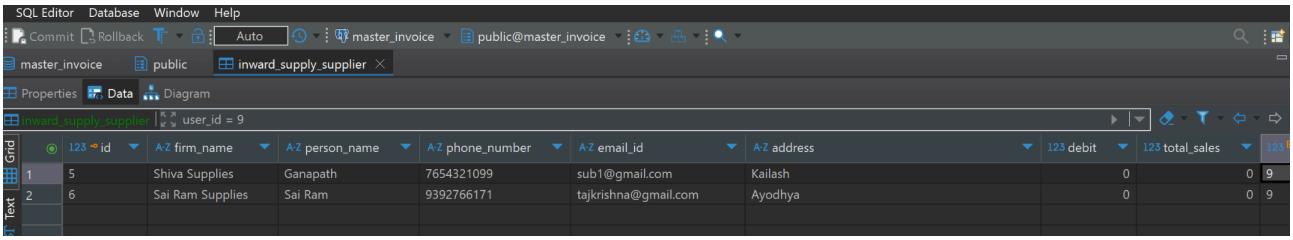
- **Test case-2:** Adding with valid details.



The suppliers page(front page) is updated



The database is getting updated



Result: After adding a supplier a message “Supplier added successfully!” is displayed.

Additional Comments: N/A

3.3.2 View Suppliers

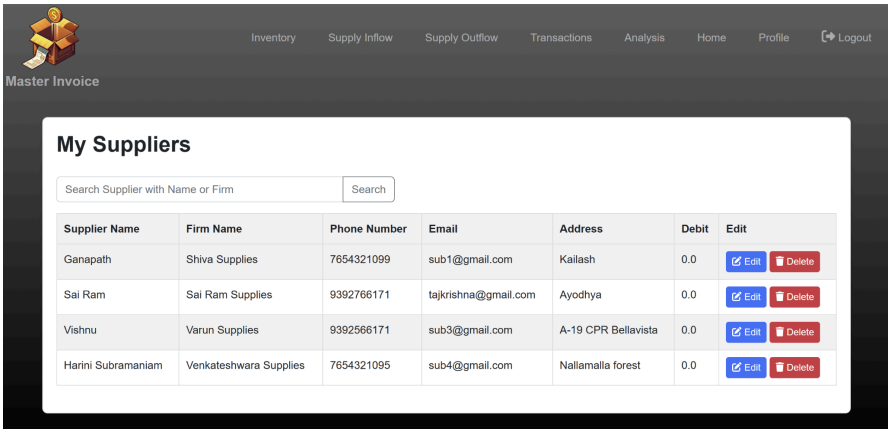
Module Details: This module integrates all the components of the view suppliers page.

Test Owner: While(1)

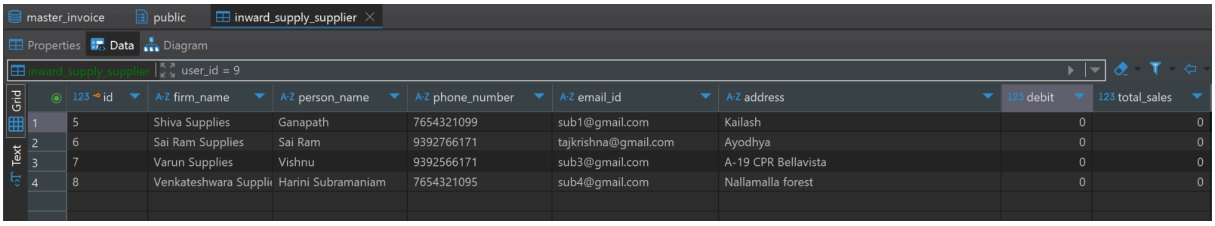
Test Date: 01-04-2025

Test Results:

- **Test case-1:** To verify whether all the suppliers of a particular user are fetched correctly or not.

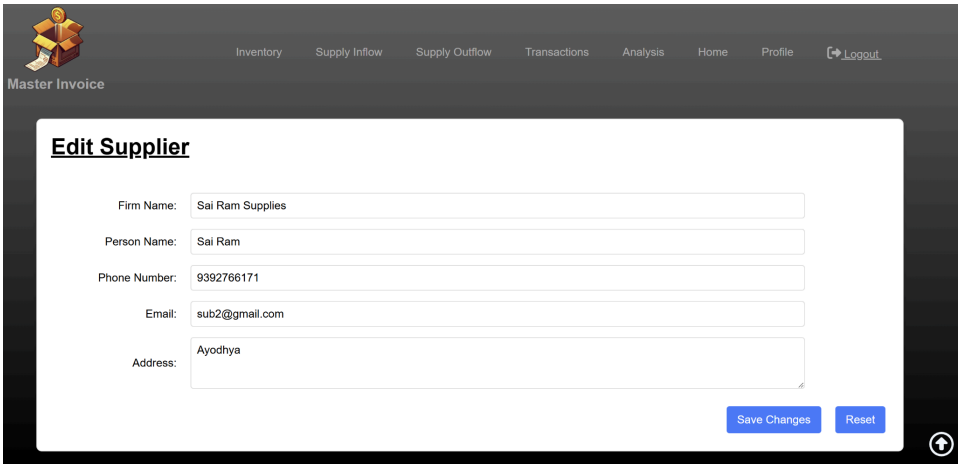


Now verifying with the backend

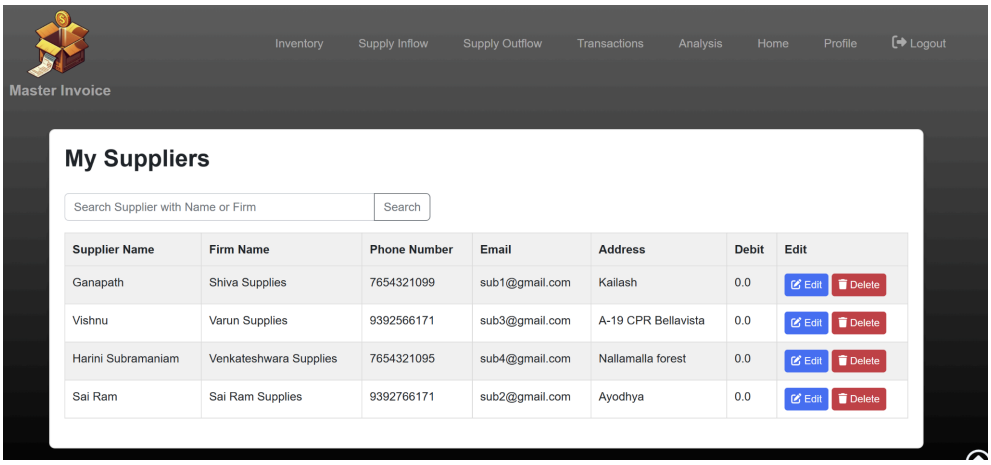


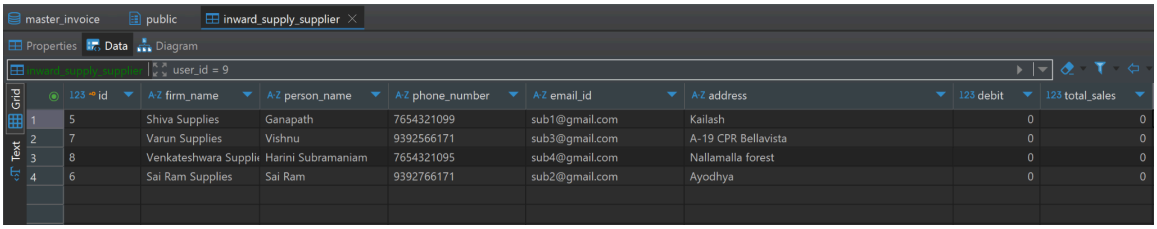
Result: All the suppliers are fetched correctly

- **Test case-2:** To verify the feature of updating the details of a supplier.



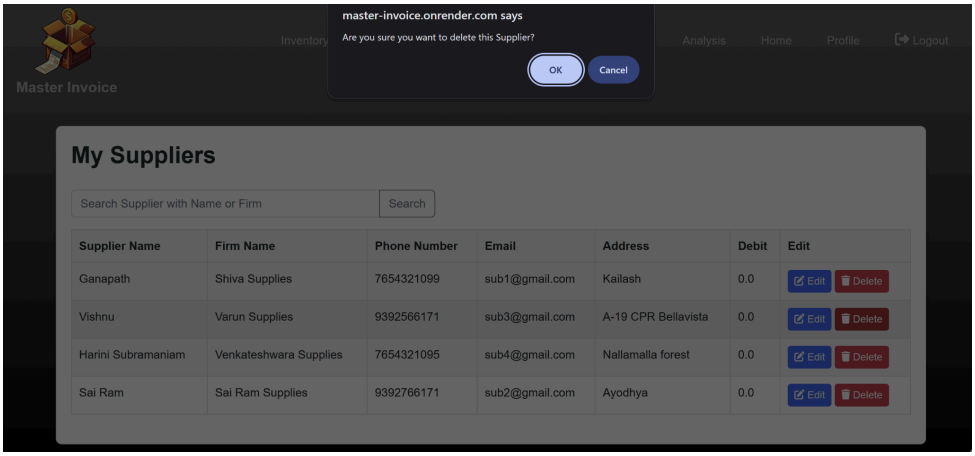
(Here we are changing the email address to sub2@gmail.com from tajkrishna@gmail.com)
Now checking the frontend and backend



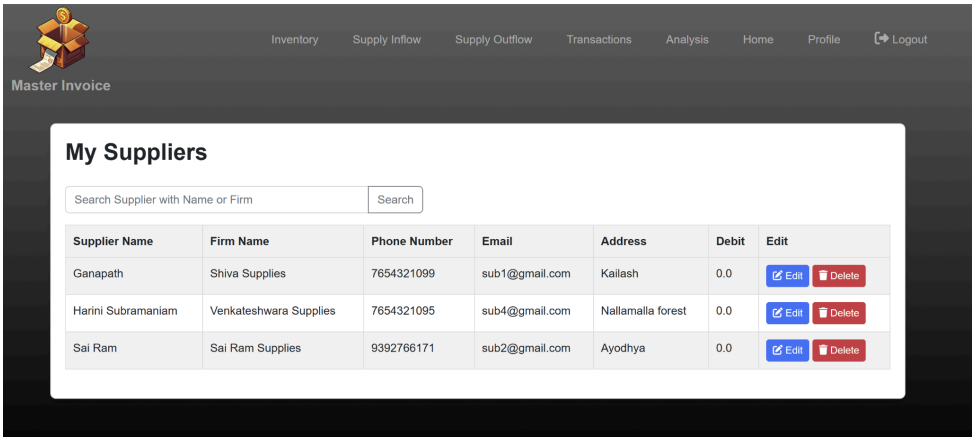


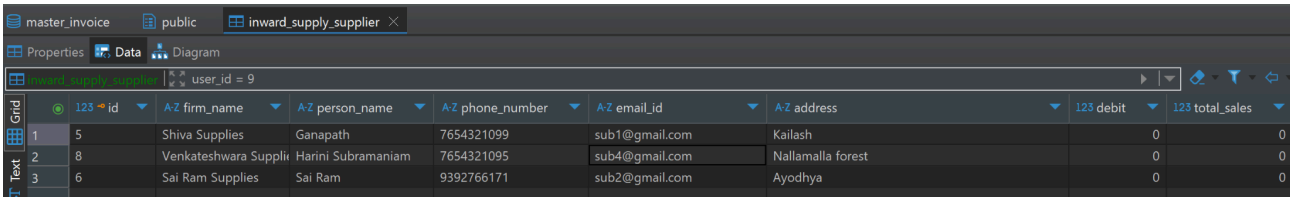
Result: The details of the supplier are updated successfully.

- **Test case-3:** To verify the feature of deleting suppliers.



(Here I am deleting the the Supplier with firm name Varun Supplies)
Now verifying with the frontend and backend





Result: First, we get an alert message confirming the deletion, after which the supplier is deleted successfully.

Additional Comments:

3.3.3 Supplier Invoice Bill

Module Details: This module integrates all the components of the suppliers' invoice page.

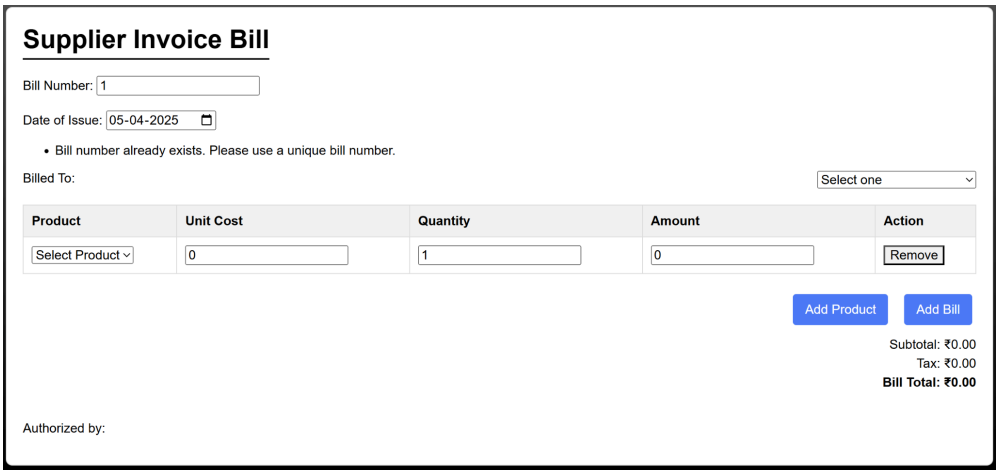
Test Owner: While(1)

Test Date: 02-04-2025

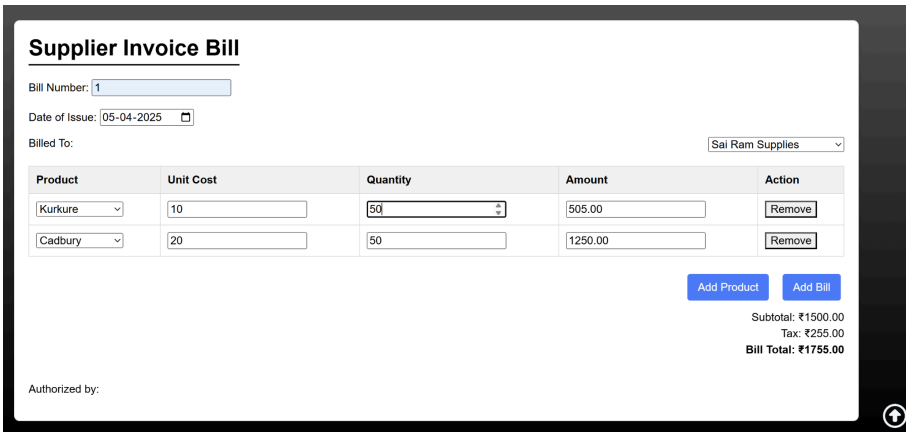
Test Results:

- **Test case-1:** To verify that the bills have a unique bill number.

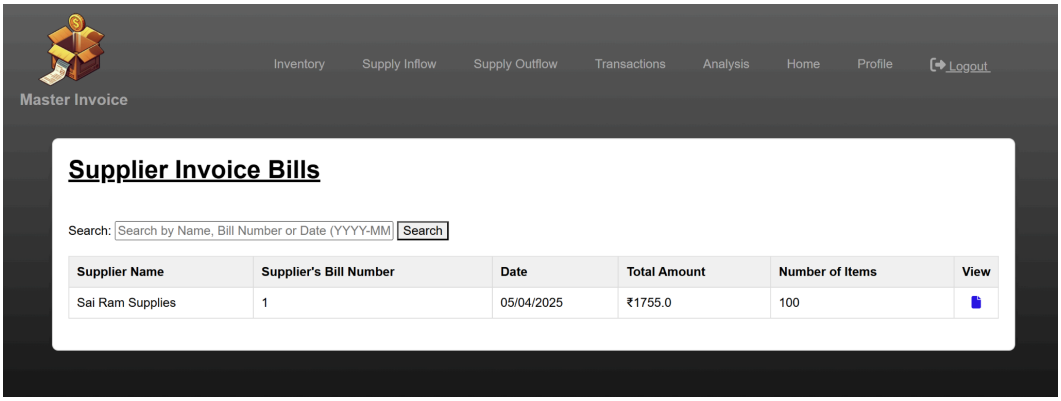
  Result: Adding a bill with the same bill number displays the error message "Bill number already exists. Please use a unique bill number."



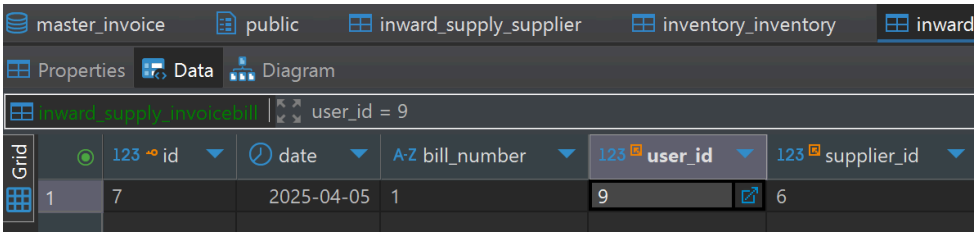
- **Test case-2:** The Supplier Invoice Bills show all history of past bills for suppliers.



Now verifying the invoice list



Now, checking the back end:

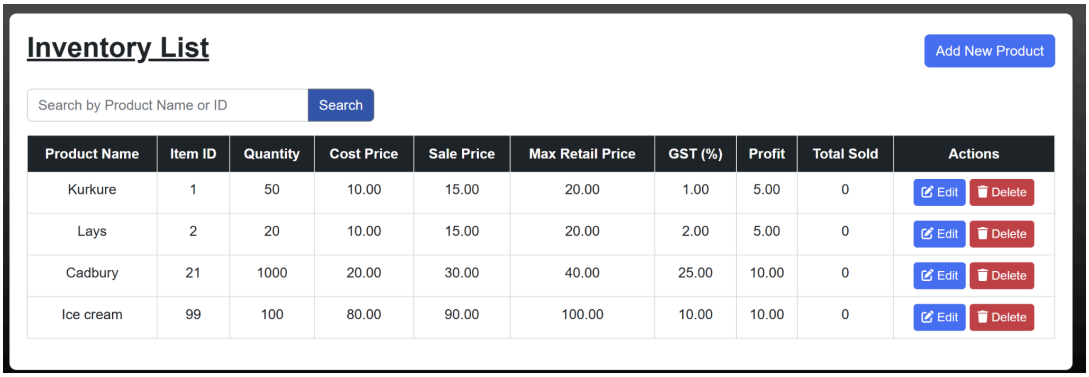


Result: The history and database are simultaneously changing.

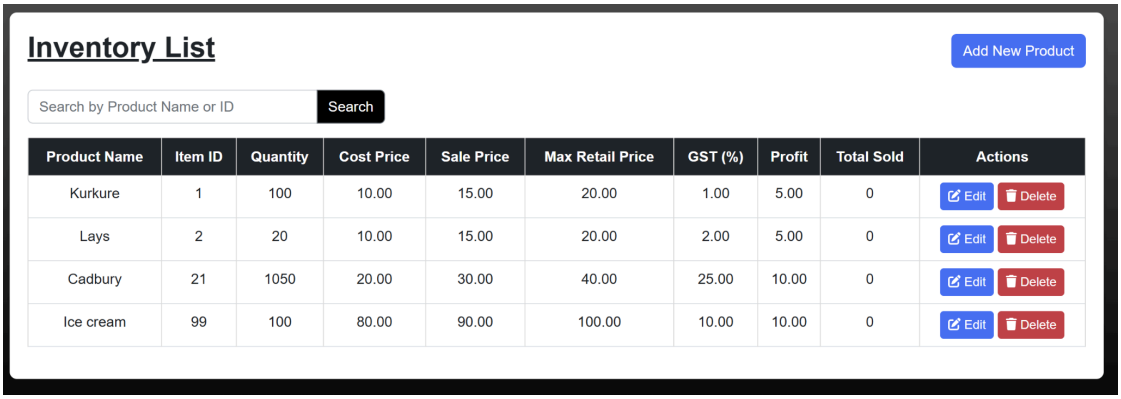
- **Test case-3:** The new bill changes the quantity in the inventory for the product.

Checking the inventory:

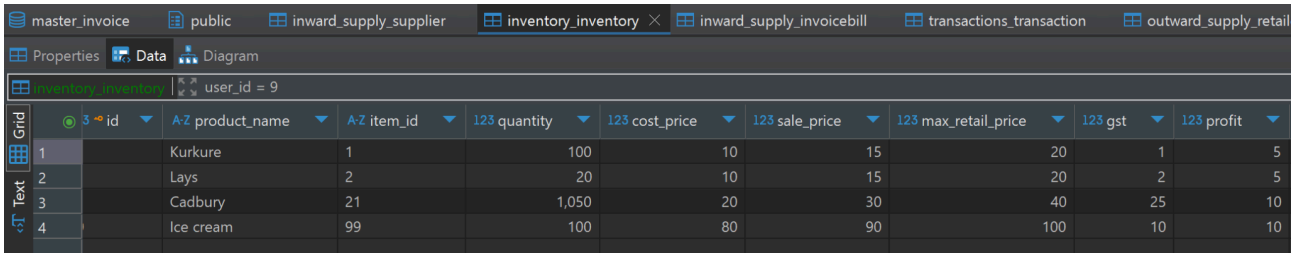
Before adding:



After adding:



Now checking the backend:



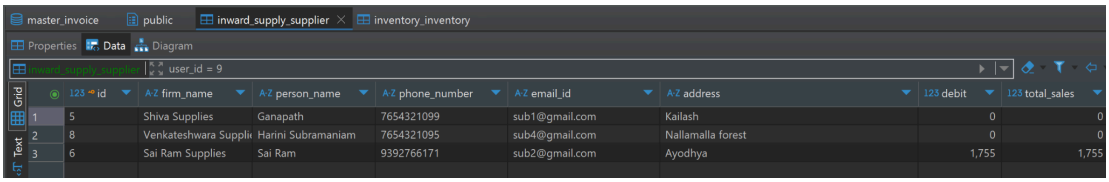
Result: The inventory in the front end and the database in the back end are simultaneously changing after making the bills.

The quantity of both Kurkure and Cadbury is increased by 50 (as we have added a bill with 50 each)

- **Test case-4:** The new bill changes the quantity in debit for the supplier.
  Now checking the Credit of the Supplier:

Sai Ram	Sai Ram Supplies	9392766171	sub2@gmail.com	Ayodhya	1755.0	Edit Delete

Before, it was 0 now, it is updated.
Now checking the backend:



Result: The supplier's debit in the front end and database in the backend are simultaneously changed by adding the debit amount.

Additional Comments:

3.3.4 INVOICE LIST

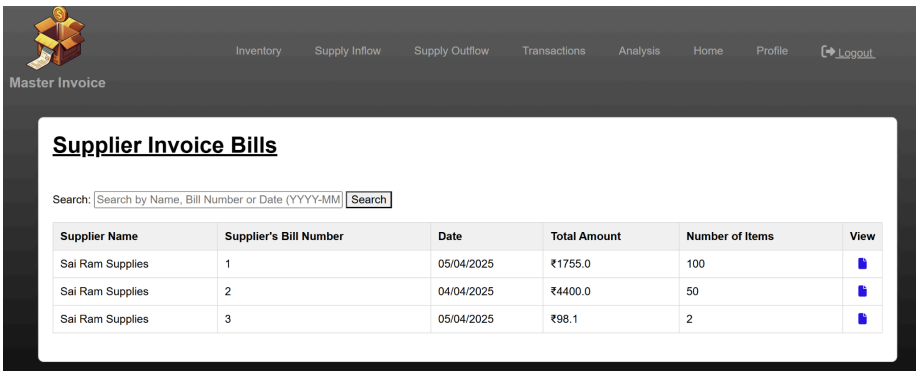
Module Details: This module integrates all the components of the suppliers' past transactions for all entered suppliers' bills.

Test Owner: While(1)

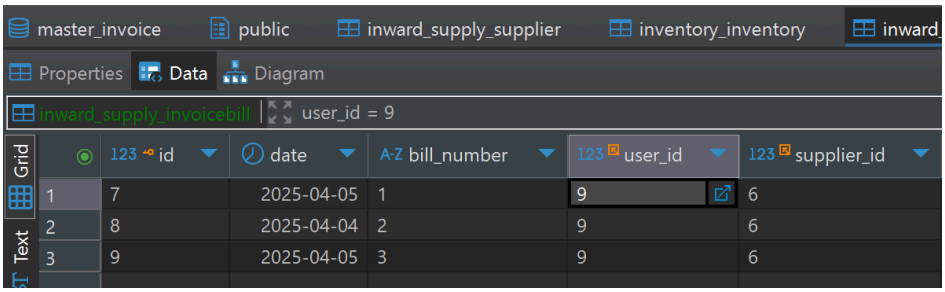
Test Date: 02-04-2025

Test Results:

- **Test case-1:** To verify that all the bills are fetched properly.



Now checking the backend:



Result: The front-end and backend databases are automatically updating a new bill into transaction inflow history.

Additional Comments:

3.4 Outward Supply

3.4.1 Add Retailer

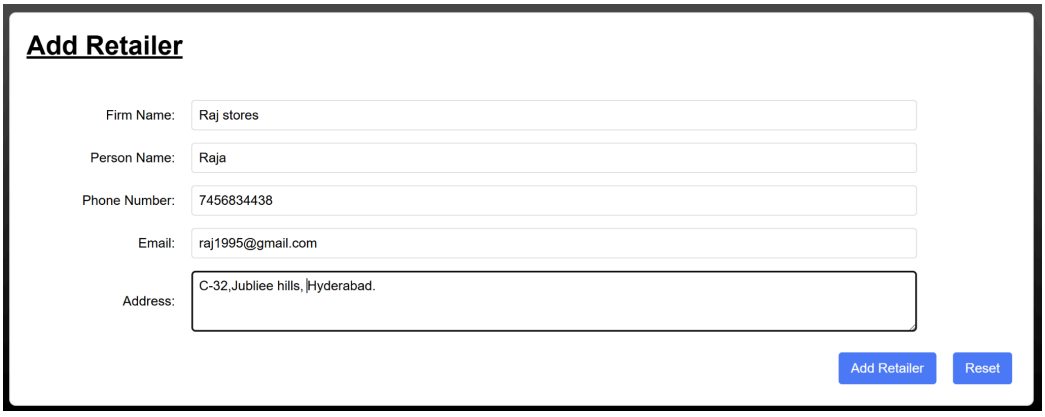
Module Details: This module integrates all the components of the add retailer page.

Test Owner: While(1)

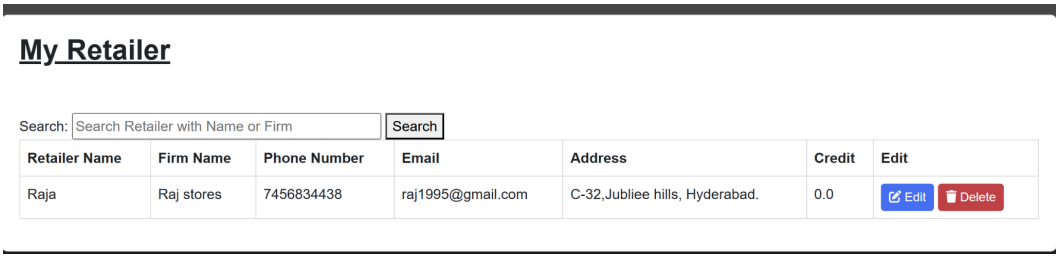
Test Date: 01-04-2025

Test Results:

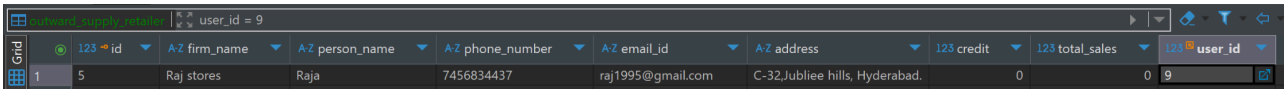
- **Test case-1:** Adding a retailer with the same Firm Name.
  Result: It displays the message "Retailer with same Firm Name already exists for your account."
- **Test case-2:** Adding with valid details.



The retailer's page(front page) is updated.



The database is getting updated



Result: The retailer is getting added successfully.

Additional Comments: N/A

3.4.2 View Retailers

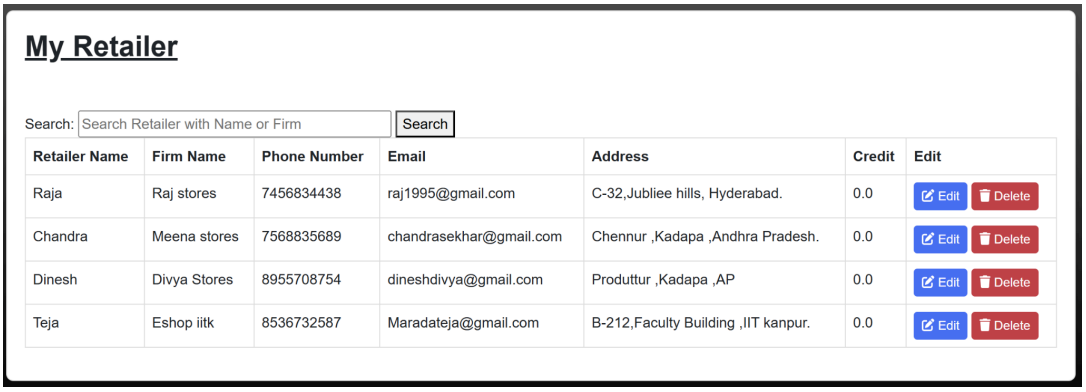
Module Details: This module integrates all the components of the view retailers page.

Test Owner: While(1)

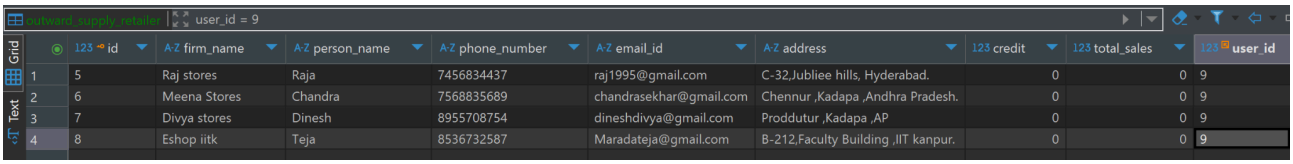
Test Date: 01-04-2025

Test Results:

- **Test case-1:** To verify whether all the retailers of a particular user are fetched correctly or not.



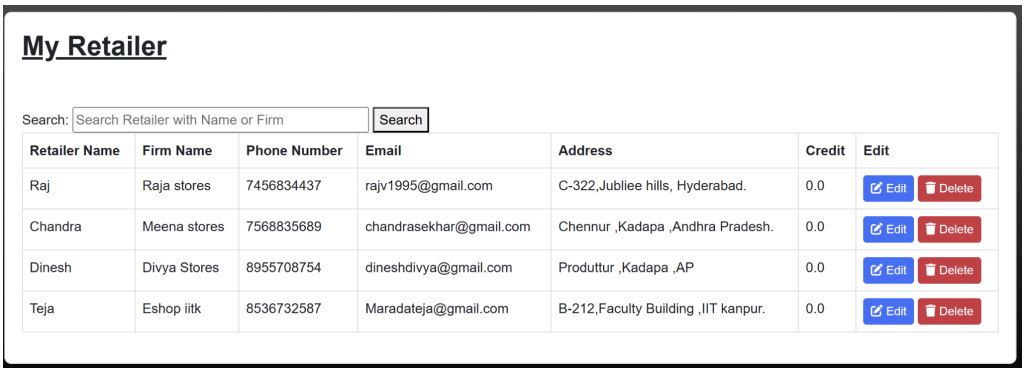
Now verifying with the backend

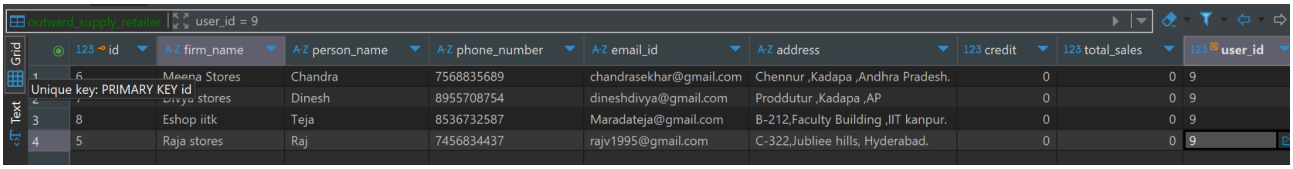


Result: All the retailers are fetched correctly

- **Test case-2:** To verify the feature of updating the details of a retailer.

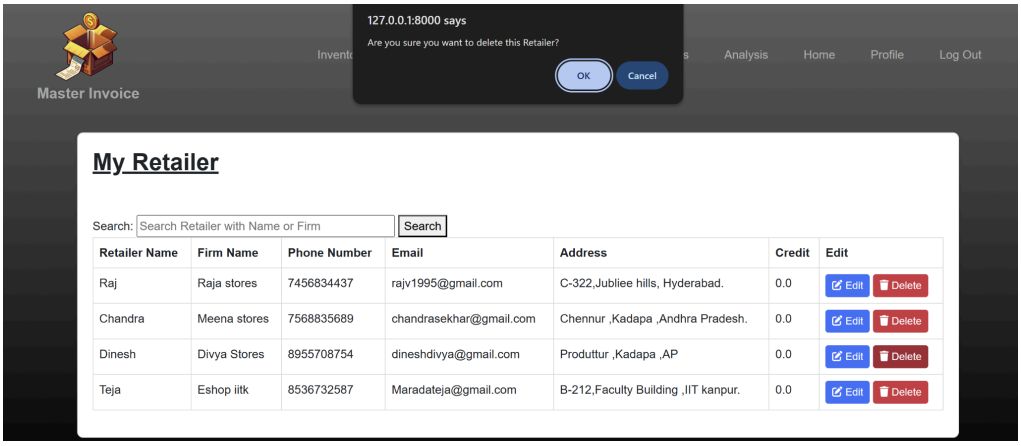
Now checking the frontend and backend



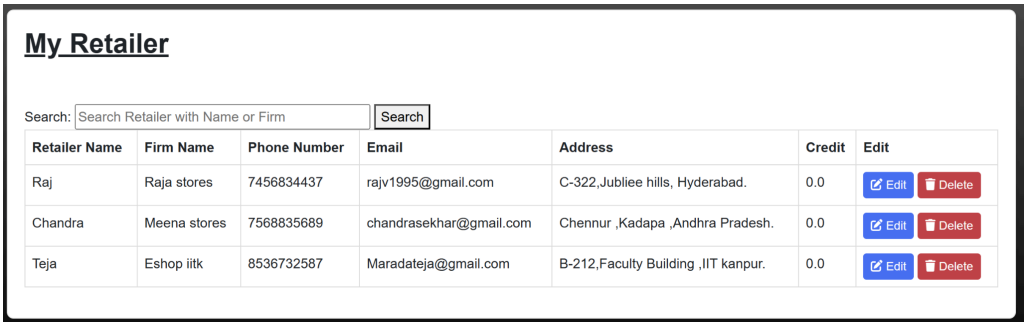


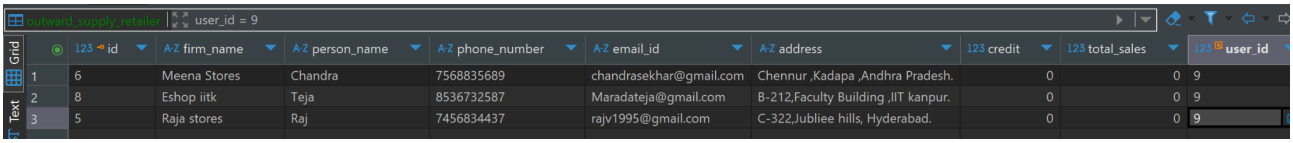
Result: The details of the retailer are updated successfully.

- **Test case-3:** To verify the feature of deleting retailers.



Now verifying with the frontend and backend





Result: First, we get an alert message confirming the deletion; the retailer is deleted successfully.

Additional Comments:

3.4.3 Retailer Invoice Bill

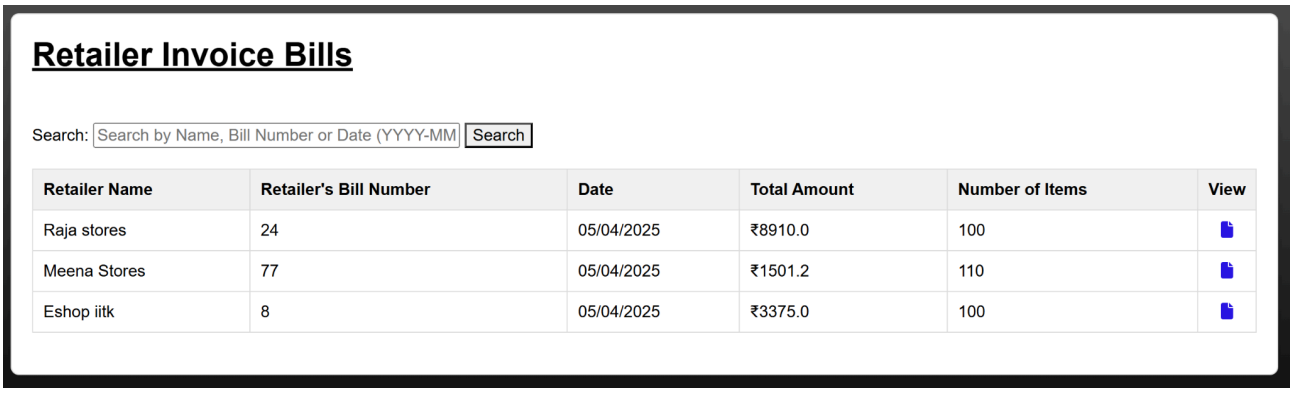
Module Details: This module integrates all the components of the retailers' invoice page.

Test Owner: Vundela Obula Reddy

Test Date: 02-04-2025

Test Results:

- **Test case-1:** To verify that the bills have a unique bill number.
  Result: Adding a bill with the same bill number displays the error message "Bill number already exists. Please use a unique bill number."
- **Test case-2:** The Retailer Invoice Bills show all history of past bills for retailers.
  Result: The history and database are simultaneously changing.



outward_supply_outward_invoice | user_id = 9

	id	date	bill_number	discount	profit	retailer_id	user_id
1	8	2025-04-05	24	10	1,000	5	9
2	9	2025-04-05	77	10	550	6	9
3	7	2025-04-05	8	10	1,000	8	9

- **Test case-3:** The new bill changes the quantity in the inventory for the product.

Result: After making the bills, the inventory in the front end and the database in the back end change simultaneously.

Before making the bill: Cadbury=950.

Inventory List

Add New Product

Search by Product Name or ID | Search

Product Name	Item ID	Quantity	Cost Price	Sale Price	Max Retail Price	GST (%)	Profit	Total Sold	Actions
Kurkure	1	0	10.00	15.00	20.00	1.00	5.00	0	Edit Delete
Lays	2	10	10.00	15.00	20.00	2.00	5.00	0	Edit Delete
Cadbury	21	950	20.00	30.00	40.00	25.00	10.00	0	Edit Delete
Ice cream	99	0	80.00	90.00	100.00	10.00	10.00	0	Edit Delete

inventory_inventory | user_id = 9

	id	product_name	item_id	quantity	cost_price	sale_price	max_retail_price	gst	profit	total_qty_sold	
1	6	Kurkure	1	0	10	15	20	1	5	0	9
2	7	Lays	2	10	10	15	20	2	5	0	9
3	9	Cadbury	21	950	20	30	40	25	10	0	9
4	10	Ice cream	99	0	80	90	100	10	10	0	9

Made a bill with 100 Cadbury.

After making the bill: Cadbury =950-100=850.

Inventory List

Add New Product

Search by Product Name or ID Search

Product Name	Item ID	Quantity	Cost Price	Sale Price	Max Retail Price	GST (%)	Profit	Total Sold	Actions
Kurkure	1	0	10.00	15.00	20.00	1.00	5.00	0	Edit Delete
Lays	2	10	10.00	15.00	20.00	2.00	5.00	0	Edit Delete
Cadbury	21	850	20.00	30.00	40.00	25.00	10.00	0	Edit Delete
Ice cream	99	0	80.00	90.00	100.00	10.00	10.00	0	Edit Delete

inventory_inventory | user_id = 9

	id	product_name	item_id	quantity	cost_price	sale_price	max_retail_price	gst	profit	total_qty_sold	
1	6	Kurkure	1	0	10	15	20	1	5	0	9
2	7	Lays	2	10	10	15	20	2	5	0	9
3	9	Cadbury	21	850	20	30	40	25	10	0	9
4	10	Ice cream	99	0	80	90	100	10	10	0	9

- **Test case-4:** The new bill changes the quantity in credit for the retailer.
  Result: The retailer's debit in front end and database in backend is simultaneously changing by adding the credit amount.

Before making the bill, credit is 8910.0

My Retailers

Add Retailer

Search: Search Retailer with Name or Firm Search

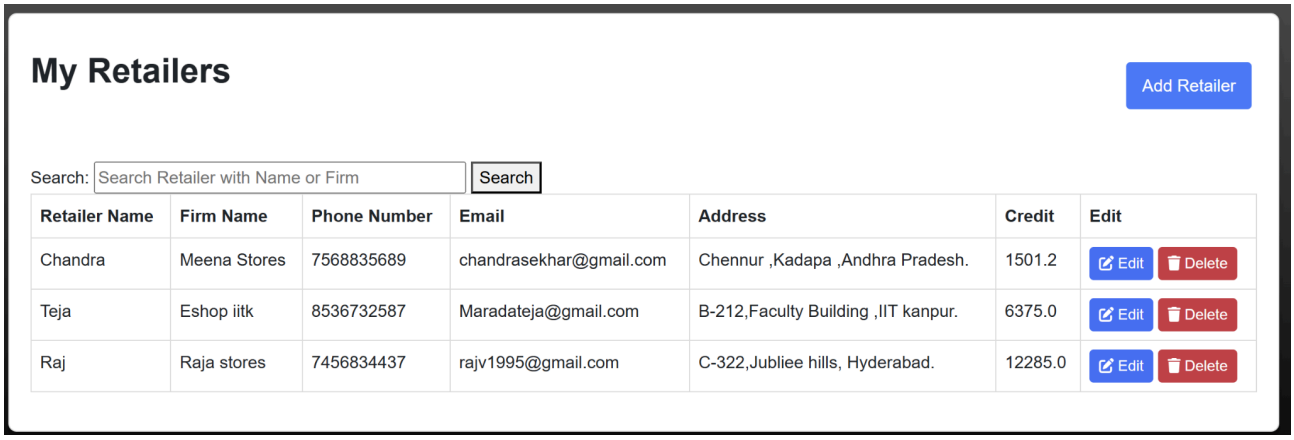
Retailer Name	Firm Name	Phone Number	Email	Address	Credit	Edit
Raj	Raja stores	7456834437	rajv1995@gmail.com	C-322,Jubliee hills, Hyderabad.	8910.0	Edit Delete
Chandra	Meena Stores	7568835689	chandrasekhar@gmail.com	Chennur ,Kadapa ,Andhra Pradesh.	1501.2	Edit Delete
Teja	Eshop iitk	8536732587	Maradateja@gmail.com	B-212,Faculty Building ,IIT kanpur.	6375.0	Edit Delete

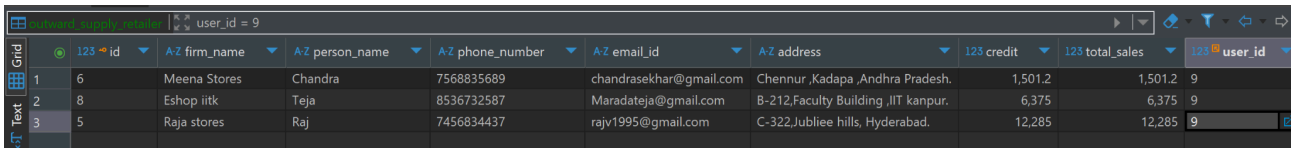
outward_supply_retailer | user_id = 9

	id	firm_name	person_name	phone_number	email_id	address	credit	total_sales	user_id
1	5	Raja stores	Raj	7456834437	rajv1995@gmail.com	C-322,Jubliee hills, Hyderabad.	8,910	8,910	9
2	6	Meena Stores	Chandra	7568835689	chandrasekhar@gmail.com	Chennur ,Kadapa ,Andhra Pradesh.	1,501.2	1,501.2	9
3	8	Eshop iitk	Teja	8536732587	Maradateja@gmail.com	B-212,Faculty Building ,IIT kanpur.	6,375	6,375	9

Made a bill with 100 Cadbury. (total bill: 3375)

After making the bill: 8910+3375 =12285





- **Test case-5:** The bill discount should not be greater than 100.
  Result: The system is showing an error message that the discount should not be greater than 100.

Additional Comments:

3.4.4 OUTWARD INVOICE BILL

Module Details: This module integrates all the components of the retailer's past transactions for all entered retailer bills.

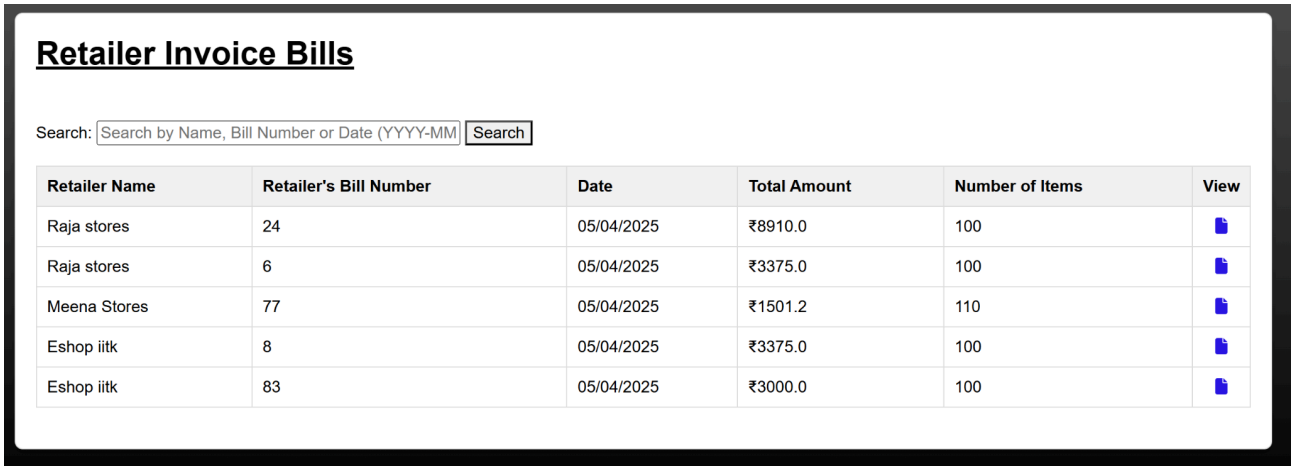
Test Owner: While(1)

Test Date: 02-04-2025

Test Results:

- **Test case-1:** To verify that all the bills are there in history.
  Result: The front-end and backend databases are automatically updating a new bill into transaction outflow history.

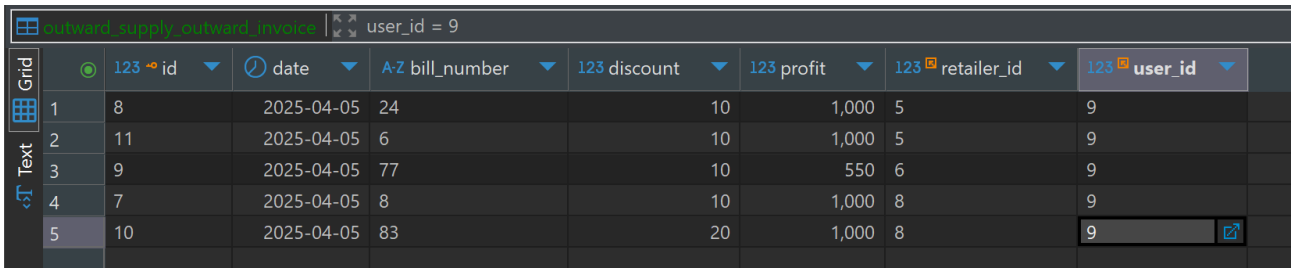
Before making the bill:(5 transactions)



Retailer Invoice Bills

Search: Search by Name, Bill Number or Date (YYYY-MM Search

Retailer Name	Retailer's Bill Number	Date	Total Amount	Number of Items	View
Raja stores	24	05/04/2025	₹8910.0	100	
Raja stores	6	05/04/2025	₹3375.0	100	
Meena Stores	77	05/04/2025	₹1501.2	110	
Eshop iitk	8	05/04/2025	₹3375.0	100	
Eshop iitk	83	05/04/2025	₹3000.0	100	

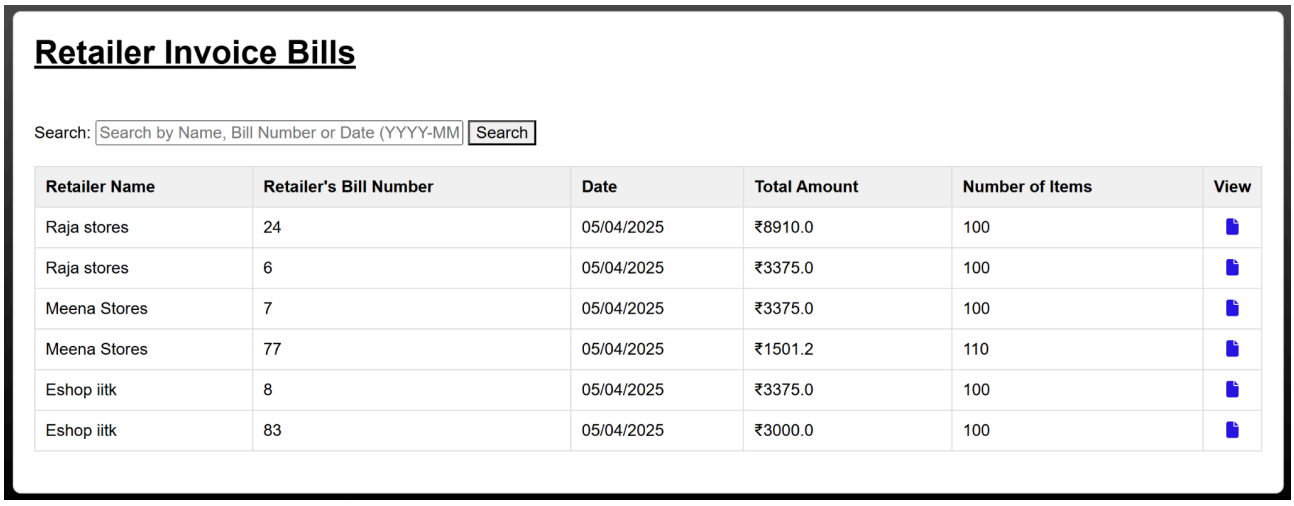


outward_supply_outward_invoice | user_id = 9

Grid Text

	id	date	bill_number	discount	profit	retailer_id	user_id
1	8	2025-04-05	24	10	1,000	5	9
2	11	2025-04-05	6	10	1,000	5	9
3	9	2025-04-05	77	10	550	6	9
4	7	2025-04-05	8	10	1,000	8	9
5	10	2025-04-05	83	20	1,000	8	9

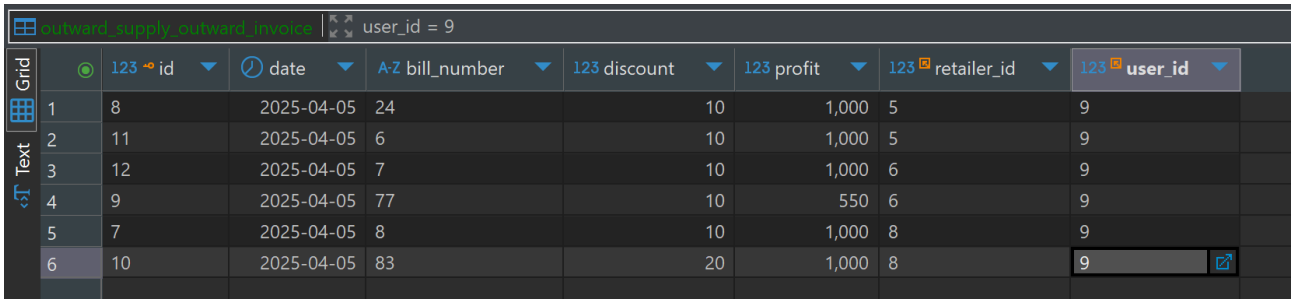
After making a transaction with retailer ID 1: (6 transactions)



Retailer Invoice Bills

Search: Search by Name, Bill Number or Date (YYYY-MM Search

Retailer Name	Retailer's Bill Number	Date	Total Amount	Number of Items	View
Raja stores	24	05/04/2025	₹8910.0	100	
Raja stores	6	05/04/2025	₹3375.0	100	
Meena Stores	7	05/04/2025	₹3375.0	100	
Meena Stores	77	05/04/2025	₹1501.2	110	
Eshop iitk	8	05/04/2025	₹3375.0	100	
Eshop iitk	83	05/04/2025	₹3000.0	100	



outward_supply_outward_invoice | user_id = 9

Grid Text

	id	date	bill_number	discount	profit	retailer_id	user_id
1	8	2025-04-05	24	10	1,000	5	9
2	11	2025-04-05	6	10	1,000	5	9
3	12	2025-04-05	7	10	1,000	6	9
4	9	2025-04-05	77	10	550	6	9
5	7	2025-04-05	8	10	1,000	8	9
6	10	2025-04-05	83	20	1,000	8	9

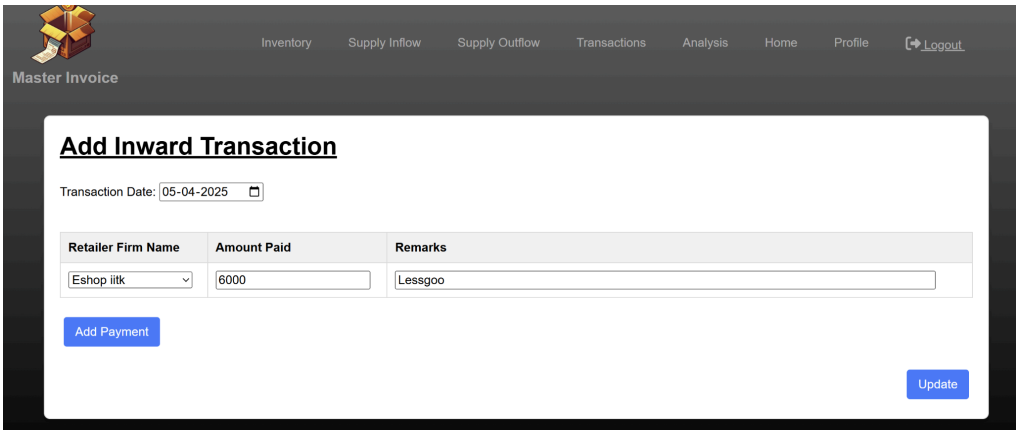
Additional Comments:

3.5 Transactions

3.5.1 Add Inward Transactions :

Module Details: This module integrates all the components of the transactions into the user's account.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

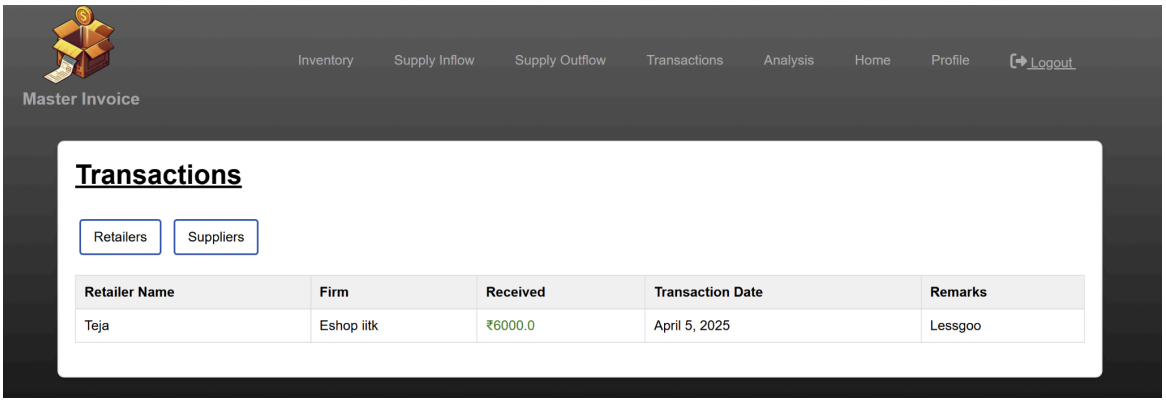
- **Test case-1:** Verifying the functionality of the add payment option(from the retailer).
  Updating payment:



Before:

Teja	Eshop iitk	8536732587	Maradateja@gmail.com	B-212,Faculty Building ,IIT kanpur.	6375.0	Edit Delete

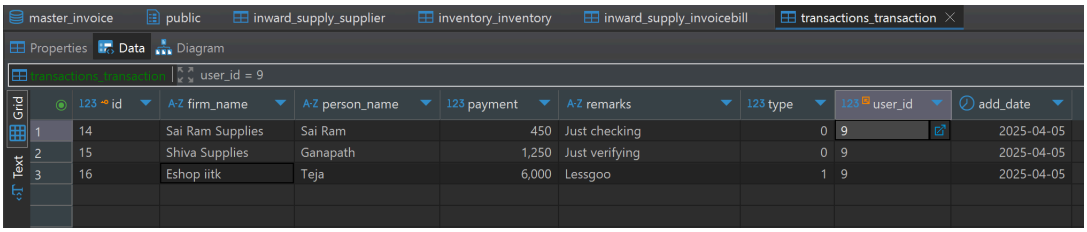
After:



The history is updated
Now checking the debit of that retailer

Teja	Eshop iitk	8536732587	Maradateja@gmail.com	B-212,Faculty Building ,IIT kanpur.	375.0	Edit Delete

Now checking the backend:



Here the **type 1** indicates **"Payment Inflow"**

8	Eshop iitk	Teja	8536732587	Maradateja@gmail.com	B-212,Faculty Building ,IIT kanpur.	375	6,375

Result: The debit of the Retailer is decreased by the chosen amount, and the transaction history is updated along with the pending transactions.

3.5.2 Add Outward Transactions :

Module Details: This module integrates all the components of the add outward transactions page.

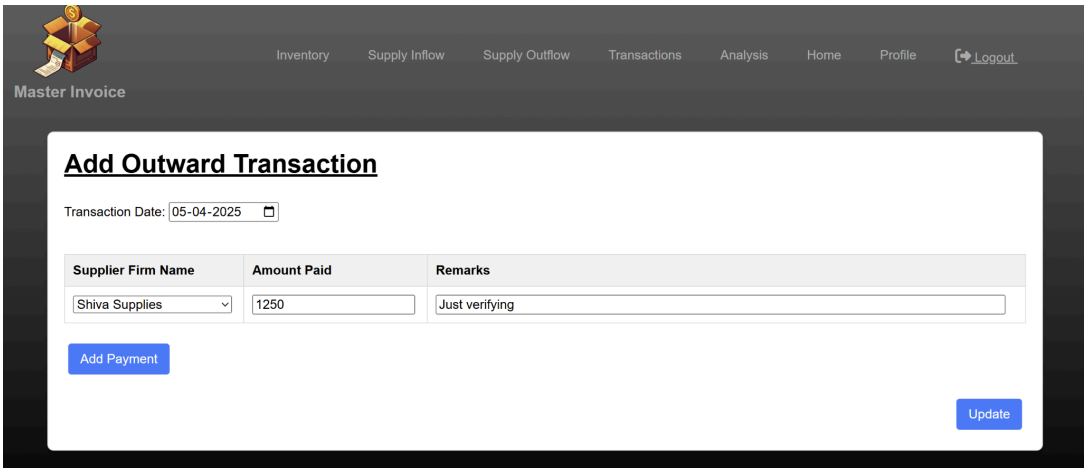
Test Owner: While(1)

Test Date: 01-04-2025

Test Results:

- **Test case-1:** Verifying the functionality of the update payment option(to the supplier).

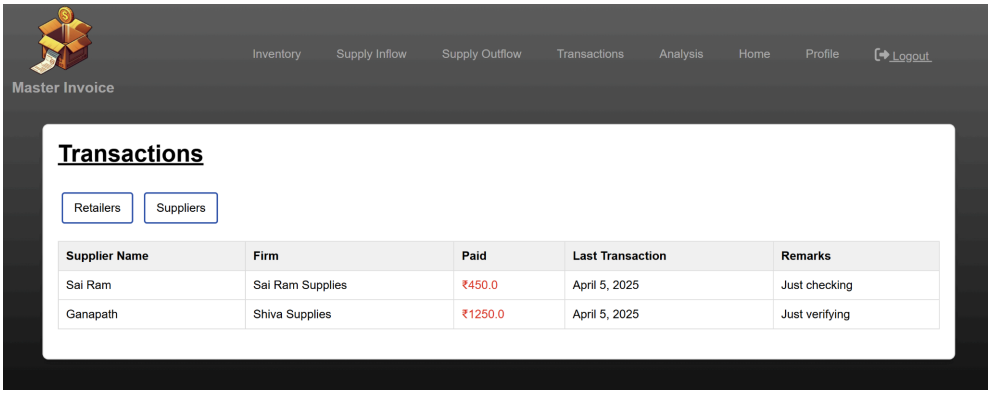
  Adding a payment now:



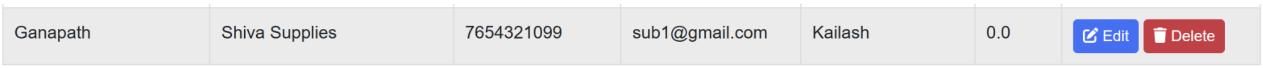
Before payment:

Ganapath	Shiva Supplies	7654321099	sub1@gmail.com	Kailash	1250.0	Edit Delete

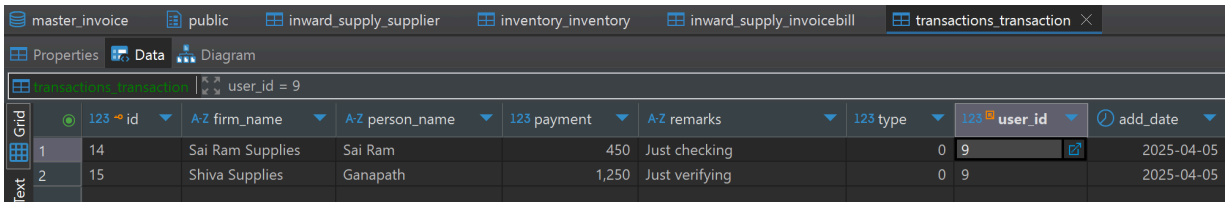
After Payment:



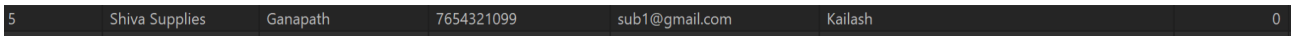
The transaction history is updated
Now checking the debit of the Supplier



Now checking the backend:
Transaction history:



Here, **type 0** indicates **"Payment outward"**

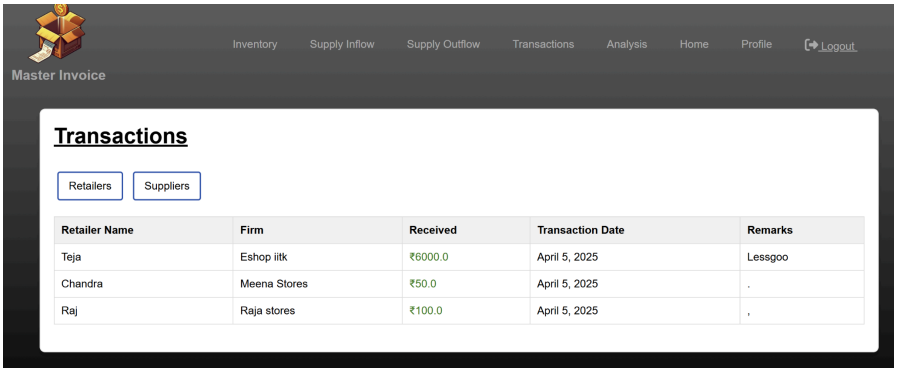


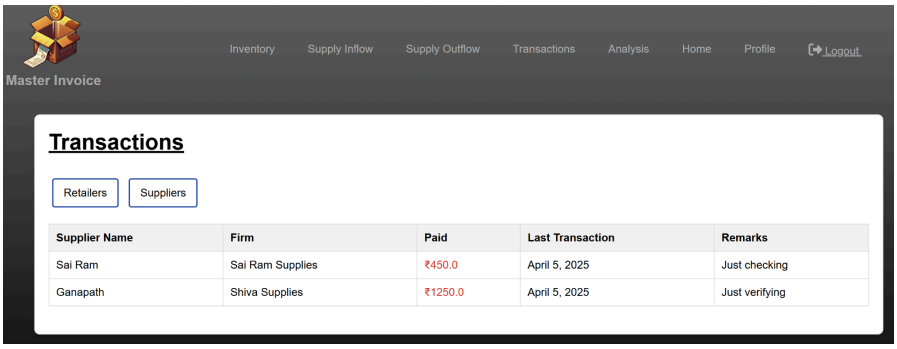
Result: The credit of the Supplier is decreased by the chosen amount, and the transaction history is updated.

3.5.3 View Transaction History :

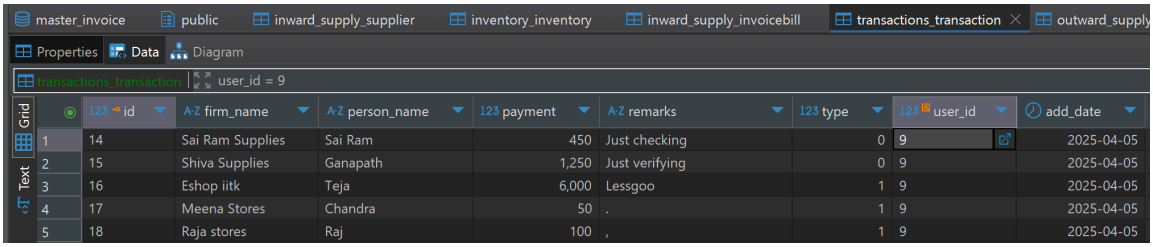
Module Details: This module integrates all the components of the view transaction history.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

- **Test case-1:** Verifying that all the transactions are fetched from the database properly.





Now checking the database:



Result: The transaction frontend is updating without delay, and the database is updating.

3.5.4 Pending Transactions :

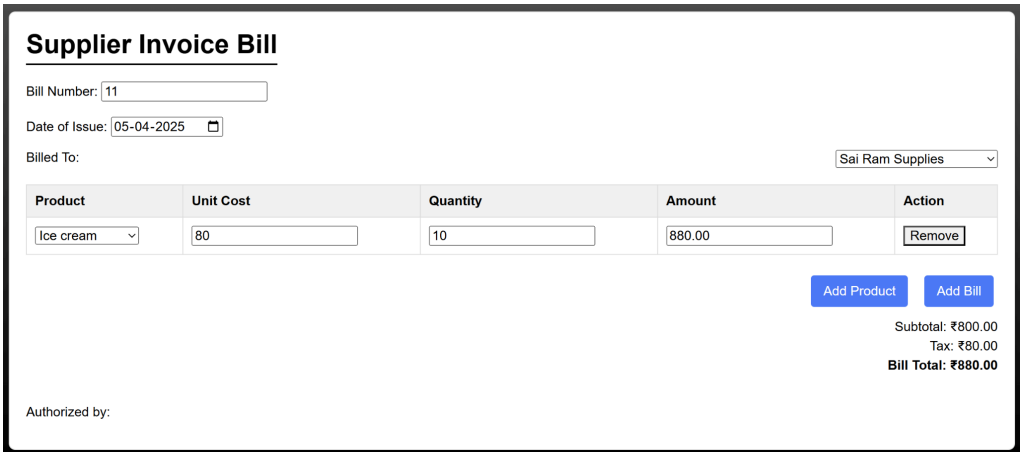
Module Details: This module integrates all the components of the pending transactions.

Test Owner: While(1)

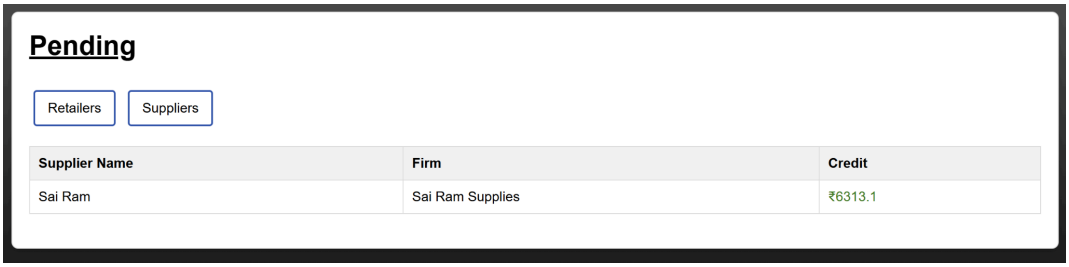
Test Date: 01-04-2025

Test Results:

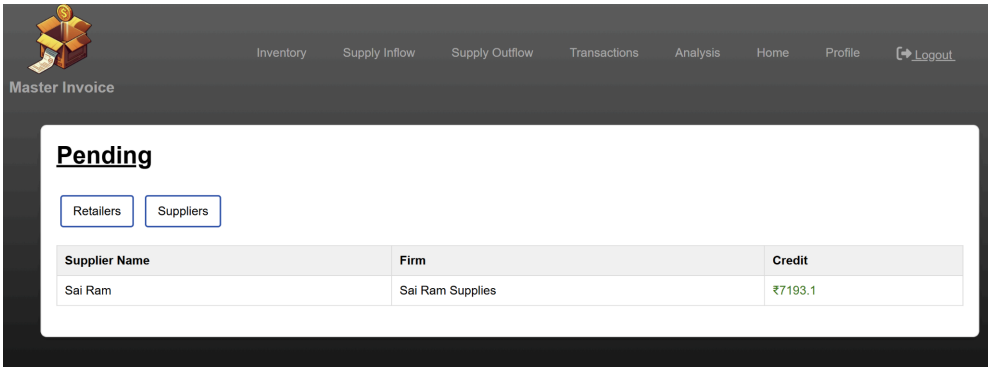
- **Test case-1:**When a user is adding a bill to the supplier,the amount is increasing in his credit of the supplier.
  Adding a bill



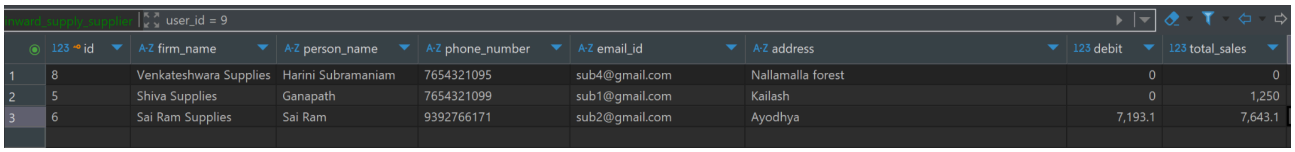
Before payment:



After payment:



The amount is updated.

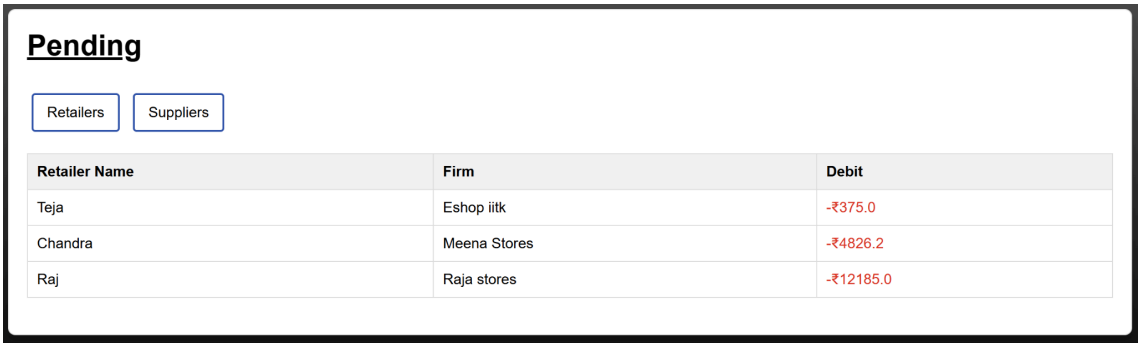


	id	firm_name	person_name	phone_number	email_id	address	debit	total_sales
1	8	Venkateshwara Supplies	Harini Subramaniam	7654321095	sub4@gmail.com	Nallamalla forest	0	0
2	5	Shiva Supplies	Ganapath	7654321099	sub1@gmail.com	Kailash	0	1,250
3	6	Sai Ram Supplies	Sai Ram	9392766171	sub2@gmail.com	Ayodhya	7,193.1	7,643.1

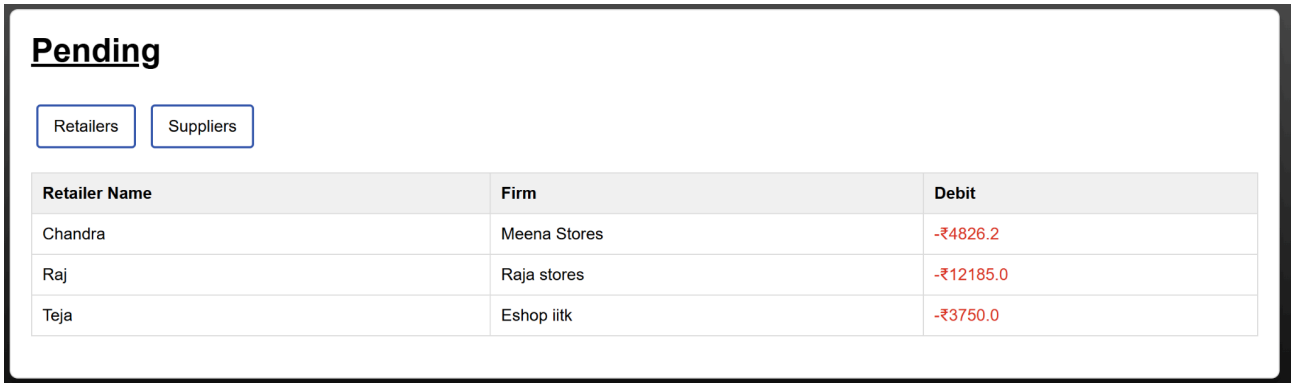
Result: The database is updating and the front end is showing all pending transactions.

- **Test case-2:** When a user is adding a bill in a retailer, the amount is increasing in his debit of the retailer.
  Adding a bill:

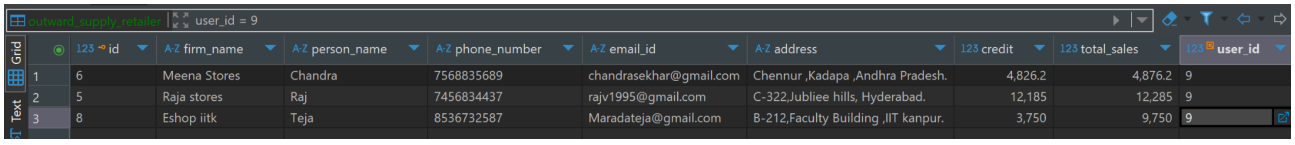
Before payment:



After payment:

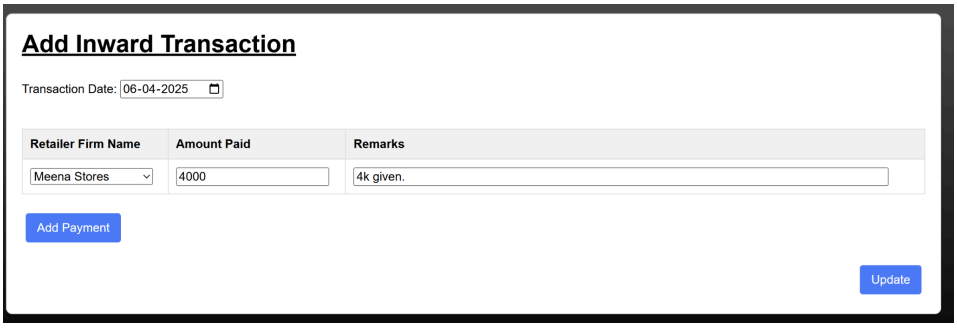


The amount is updated.

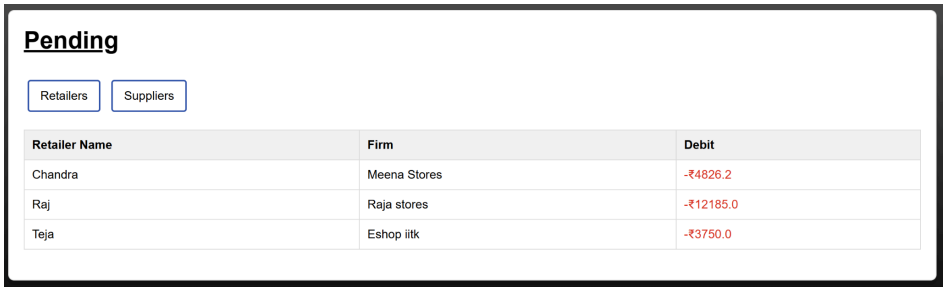


Result: The database is updating and the front end is showing all pending transactions.

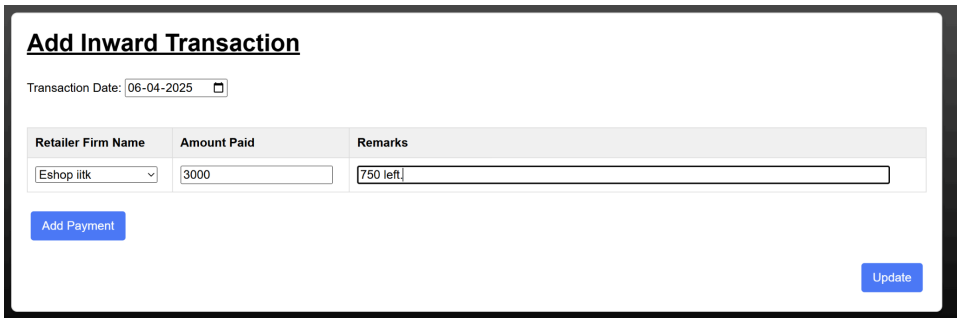
- **Test case-3:** When a retailer has made a transaction.

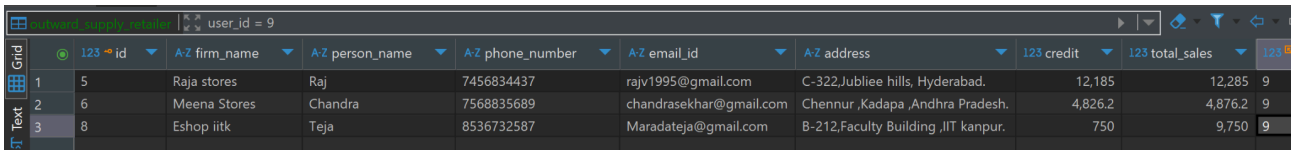


Before transaction :



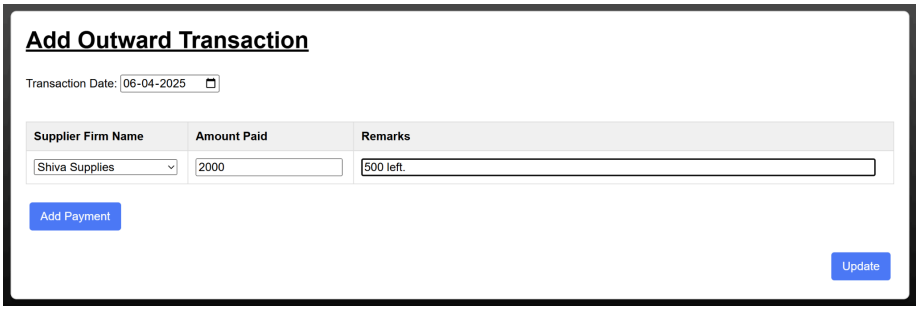
After transaction :



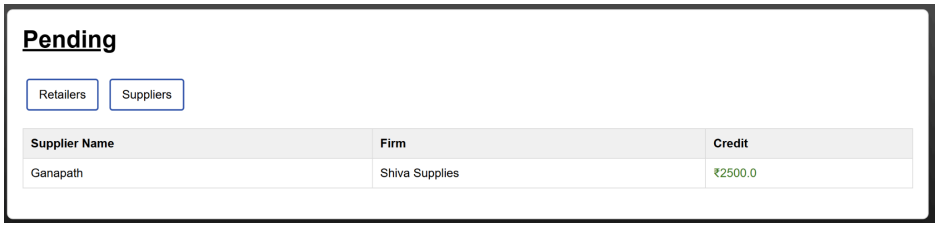


Result: The database is updating in the backend and the front end is showing all pending transactions.

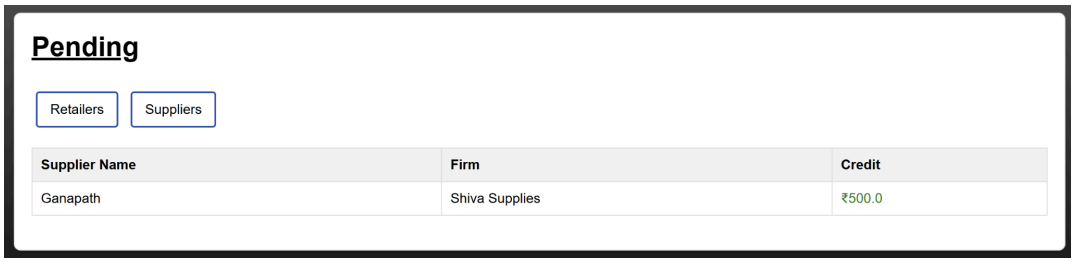
- **Test case-4:** When a user has made a transaction to a supplier.

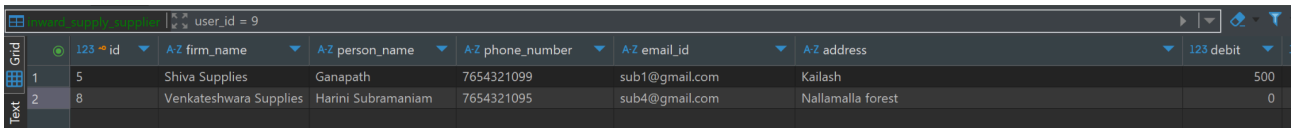


Before transaction :



After transaction :



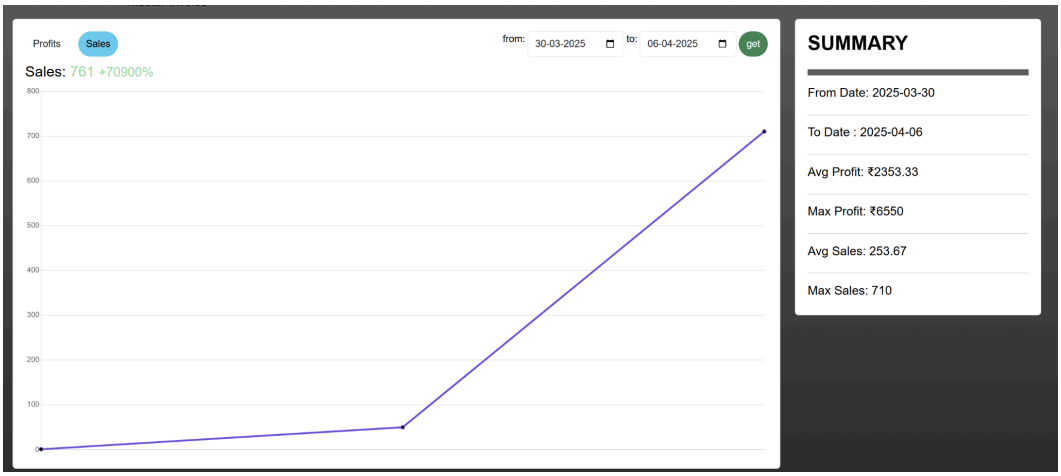


Result: The database is updated in the backend and the front end is showing all pending transactions with updated credits.

3.6 Analysis

Module Details: This model ensures smooth functionality of the analysis generation system as soon as the user chooses the desired range of dates.
Test Owner: While(1)
Test Date: 01-04-2025
Test Results:

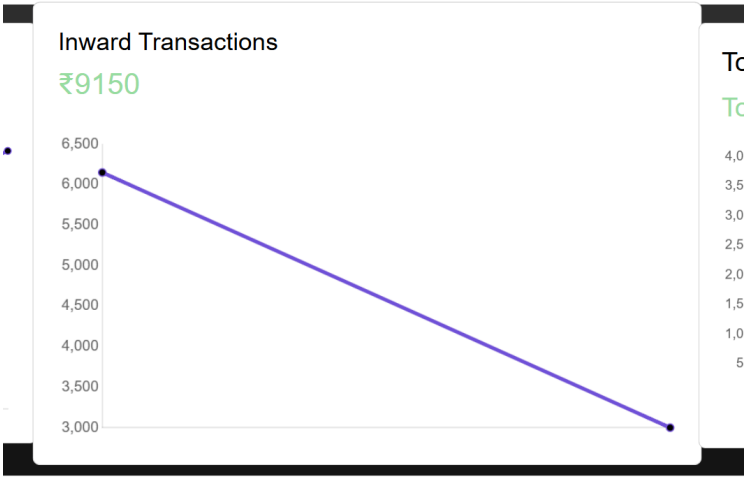
- **Test case-1:** Checking whether profits are fetched correctly and the graph is displayed according to the selected date range.



Result: The profits(SP-CP) are calculated correctly, and the graph is displayed correctly.(It is fetching the data from the outward invoice bills adding up the profit attribute for each item and displaying the whole profit). The sales(Total quantity sold in bills) are directly fetched from the bills and are displayed correctly.

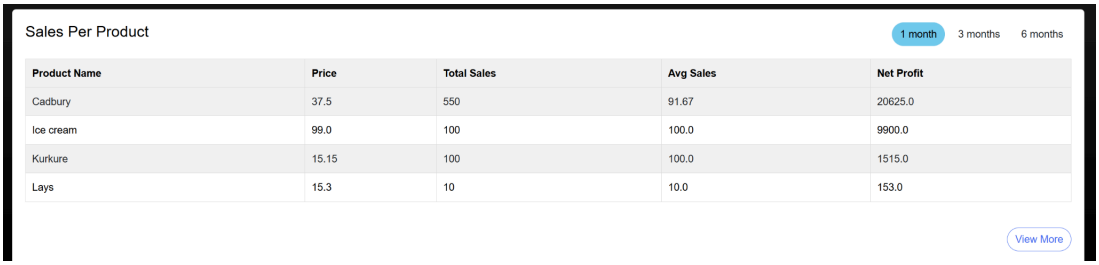
- **Test case-2:** Checking whether the table “Bills” is fetching data correctly.
  Result: It is fetching data correctly.(it simply calculates the total number of outward bills genrated)

- **Test case-3:** Checking whether table “Inward transactions” is fetching data correctly correctly.



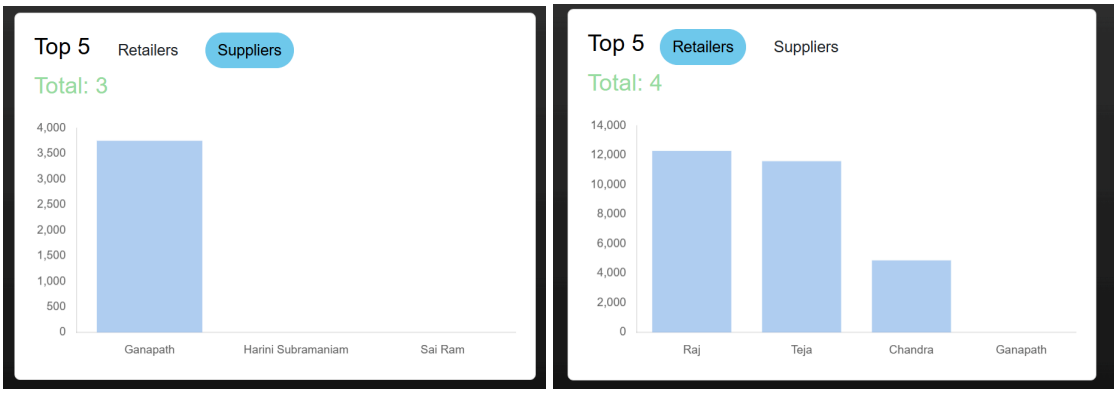
Result: It is fetching data from the inward transaction section(i.e, it adds up all the transaction models of type 1)successfully.

- **Test case-4:** Checking whether the top products sold are updated correctly.



Result: The top products are estimated using the outward bills, where from each bill the quantity is checked and is added for that product.

- **Test case-5:** Checking whether top 5 retailers and suppliers are showing correctly when more than 5 suppliers are added.



Result: The bar graph is showing the top 5 suppliers and retailers and updating after making bills.

3.7 Profile

3.7.1: View profile

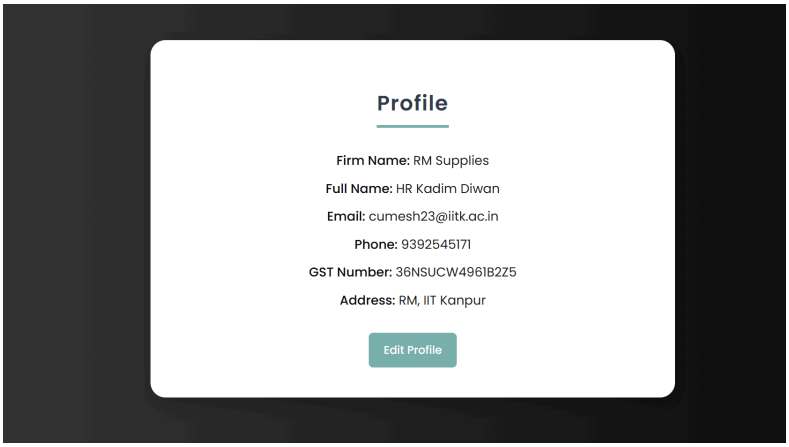
Module Details: This module integrates all the components of the view profile page.

Test Owner: While(1)

Test Date: 01-04-2025

Test Results:

- **Test case-1:** Verifying whether all the details are fetched and shown correctly.



now checking the backend.

id	phone	firm_name	full_name	address	user_id	GST_number
8	9392545171	RM Supplies	HR Kadim Diwan	RM, IIT Kanpur	9	36NSUCW4961B2Z5

Result: All the details are shown correctly.

3.7.2: Edit profile

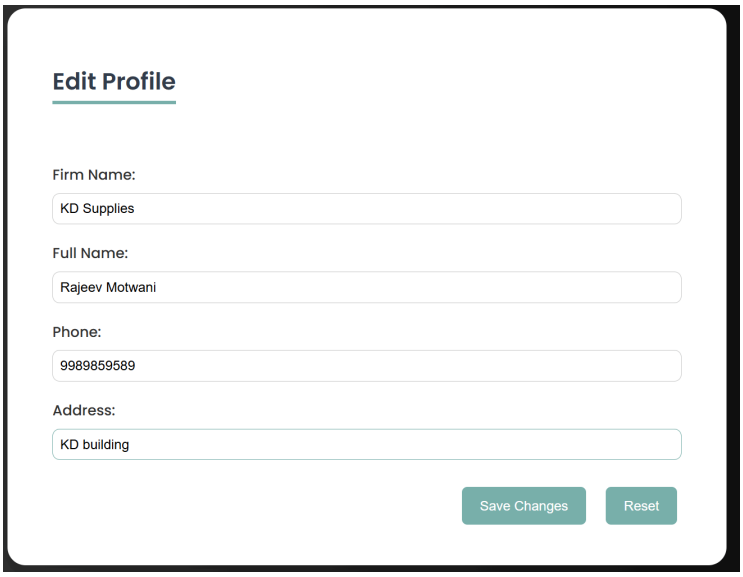
Module Details: This module integrates all the components of the edit profile page.

Test Owner: While(1)

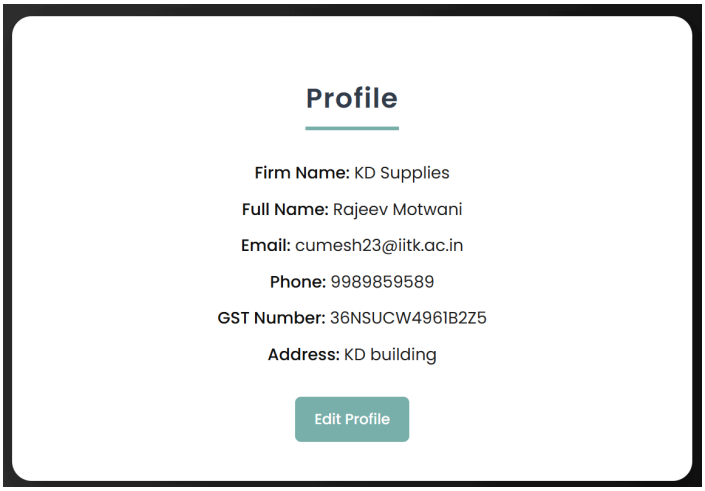
Test Date: 01-04-2025

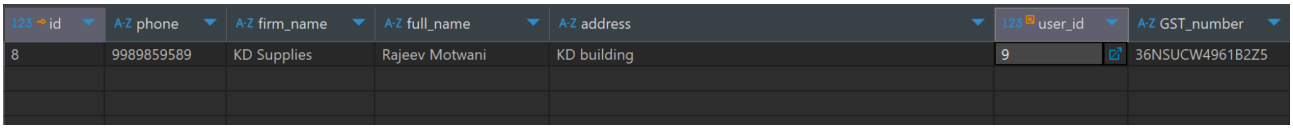
Test Results:

- **Test case-1:** Verifying the functionality of the edit profile.



Now checking front end and the database





id	phone	firm_name	full_name	address	user_id	GST_number
8	9989859589	KD Supplies	Rajeev Motwani	KD building	9	36NSUCW4961B2Z5

Result: All the details are updated successfully.

3.8 Logout

3.8.1:

Module Details: This module integrates all the components of the logout.

Test Owner: While(1)

Test Date: 01-04-2025

Test Results:

- **Test case-1:** Checking the functionality of the logout button.

Result: The user is logged out successfully and the user is redirected to the login-page and the database of every change made by the user is saved.

4 System Testing

FUNCTIONAL REQUIREMENTS

1. **Requirement:** The User should be able to sign in using valid credentials(username and password)

The sign-in functionality was rigorously tested to ensure seamless and secure user authentication. Test cases were designed to validate various scenarios, including successful logins, incorrect credentials, and system responses. During testing, we manually accessed the login page, entered registered usernames and valid passwords, and confirmed that the system correctly authenticated users and redirected them to the dashboard.

We also verified error-handling by inputting invalid credentials, such as unregistered usernames and incorrect passwords, ensuring the system displayed appropriate error messages (e.g., "Invalid credentials") and prevented unauthorized access. Additionally, session persistence was validated by checking that users remained logged in until manual logout or session expiration.

Test Owner: While(1)
Test Date: 02/04/2025
Test Results: The Sign-in page worked as expected. We had tested it on our deployed website. No bug was reported.
Additional Comments: In case of wrong credentials, the login page toggles

2. **Requirement:** The user should be able to sign up using valid credentials (include full name, firm name, phone number, email, and GST number) with a strong enough password.

The sign-up functionality underwent extensive testing to ensure a secure and user-friendly registration process. The system properly validated all user inputs, requiring a unique username, valid email format, and a strong enough valid password, which had to be re-verified through the confirm password field. The OTP verification mechanism worked as intended, sending a time-sensitive code to the registered email that automatically expired after the set duration. Crucially, the resend OTP feature was correctly disabled until the countdown timer elapsed, preventing abuse while maintaining usability. After successful OTP verification (whether initial or resent), users could proceed to complete their profile details. Negative test cases confirmed that the system blocked weak passwords, mismatched password confirmations, invalid OTPs, and premature resend attempts, displaying appropriate error messages in each scenario. The end-to-end process maintained strict security protocols while providing clear guidance, and all successfully registered accounts were immediately accessible for login without any issues.

Test Owner: While(1)
Test Date: 02/04/2025
Test Results: The Register page worked as expected.
Additional Comments: NA

3. Requirement: The dashboard shall provide the user with various options like inventory, inward supply, outward supply, analysis, transactions, profile, about us.

To check this requirement , after logging in we manually navigated to each of the options making sure each of the options is easily accessible. Everything worked fine , no bugs were detected.

Test Owner: While(1)

Test Date: 26/03/24

Test Results: The Dashboard provides options to navigate to different pages effectively. everything works fine and no bugs were detected.

Additional Comments: NA

4. Requirement: Add Product – The user will be required to fill in data related to the product they are adding. and updated when the user adds an inward invoice bill.

Test Description: Add Product functionality was extensively validated to ensure new entries could be seamlessly created and stored. Testers began by navigating to the Add Product form and filling in various required fields, including the product name, item ID, Quantity, cost price, selling price, MRP and GST(%). Special focus was given to validating the default behavior of the quantity field, which was expected to initialize with a value of zero.

All error messages, redirections, and field hints were reviewed.Products were subsequently tracked to verify their presence in the inventory list and validate that no unexpected duplication occurred. We also simulated subsequent invoice entries to ensure the quantity field updated correctly post product creation. The user interface and form responsiveness were found to be intuitive and stable throughout the tests.Uniqueness of Product ID is verified.

Test Owner: While(1)

Test Date: 01/04/2025

Test Results: The Product addition worked as expected with correct defaults and dynamic dropdowns. No bugs were found.

Additional Comments: NA

5. Requirement: The user may view all the information related to the products that currently exist in their inventory. They may search using Product name and ID. Each product has attributes such as Name of Product, Item ID,Quantity, Cost Price, Selling Price, MRP, GST(%).

Test Description: The inventory functionality was thoroughly tested to ensure that users could seamlessly view all product details currently present in their account. The test process began with the login and navigation to the inventory section, followed by the verification of product listings. Each product's displayed attributes were compared against the expected fields, including Product Name, Item ID, Company Name, Quantity, Cost Price, Selling Price, MRP and GST(%) to ensure completeness and accuracy.

Both exact and partial search queries based on product name and ID were tested for robustness using multiple Products.

Edge cases, such as empty inventory and extremely large product lists, were also simulated to test scalability and UI responsiveness. Throughout the testing cycle, the user interface remained intuitive, and no functional bugs were encountered. Overall, the inventory was found to be reliable, accurate, and user-friendly in helping users manage product visibility.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: The inventory page displayed all products correctly. Sorting worked as intended. No bugs were reported.
Additional Comments: NA

6. Requirement: Modify Product – The user may change product attributes when required.

Test Description: To assess the Modify Product functionality, testers edited a diverse set of product entries in the inventory. Each product's editable fields, such as quantity, cost price, selling price, GST(%) and MRP were modified one at a time to validate the system’s ability to persist updates correctly. Following each update, the changes were saved and checked immediately in the main inventory view to confirm that the data was reflected without delay.

Scenarios tested included both valid and invalid inputs (e.g., negative pricing or empty fields) to examine system validation responses. Edge cases like editing high-volume products and frequent successive edits were used to stress-test the update mechanism.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: The Modifications were saved correctly and displayed accurately in the inventory.
Additional Comments: NA

7. Requirement: Add Supplier – The user can add a new supplier by providing relevant details such as supplier name, contact information, and firm details.

Test Description: Testing for the Add Supplier functionality involved adding several new supplier records with varying details to verify input handling, validation, and proper data persistence. The form required fields such as supplier name, phone number, email, and firm name, all of which were tested using both typical and edge-case inputs to ensure robust form validation. Special care was taken to test input constraints such as invalid email formats, overly long phone numbers, and duplicate entries.

Upon submission, the system response and the post-save behavior were monitored. Successful entries were verified in the supplier list to ensure real-time reflection. Dropdowns were tested to ensure existing suppliers were not duplicated erroneously. Field reset and cancel functionality were tested to validate proper user experience.

Testers also ensured that any system-generated IDs or tags linked to suppliers were unique and correctly associated with future invoice entries. The UI experience remained stable, with no performance lags.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Suppliers were added without issue, and validation errors were triggered appropriately.
Additional Comments: error message display will be non time varied.

8. Requirement: View Supplier – The user can view a list of all suppliers with detailed attributes and search/filter them as needed.

Test Description: To ensure the reliability of the View Supplier feature, tests involved verifying the presentation and accuracy of the supplier list. Each listed supplier was confirmed to display attributes such as name, contact info, and firm details. Search options were tested with different combinations of fields to evaluate UI responsiveness and backend accuracy.

Testers searched for suppliers using exact names, partial strings, for firm name and supplier name. Pagination and scrolling were verified on long lists to ensure smooth usability. Edge scenarios, like empty supplier databases and high supplier volumes, were used to assess scalability and stability.

The system's ability to update the list dynamically after edits or additions was verified, ensuring data consistency. The interface remained clean and navigable even with expanded datasets.

Test Owner: While(1)
Test Date: 02/04/2025
Test Results: Supplier listings displayed all expected details. Filters and searches are performed as intended.No bugs detected.
Additional Comments: NA

9. Requirement: Add Inward Invoice Bill – The user may add an invoice bill for inward supply by selecting supplier and product(s) involved.

Test Description: The Add Supplier Invoice Bill module was tested by simulating real-world billing scenarios. Testers selected various suppliers and multiple products per bill, then input quantities, unit prices, billing date . Each entry was saved and checked for correct association with the corresponding supplier and product IDs.

Validation tests included missing fields, invalid numerical entries, and duplicate bill numbers. Calculations were independently verified to ensure the system computed totals correctly. Edge cases, such as extremely high quantities were included.

Post-submission behavior, including inventory updates and linkage to supplier history, was closely monitored. The ability to handle invoices with many line items was assessed for performance. The form design supported ease of use and gave real-time feedback on calculations.

Test Owner: While(1)
Test Date: 02/04/2025
Test Results: Invoices were added with correct totals, linked to respective suppliers, and inventory updated. no bugs detected.
Additional Comments: NA

10. Requirement: Invoice Lists– The user can view the invoice history associated with each supplier.

Test Description: Testing for the invoice history feature involved validating that all previous invoice bills associated with each supplier were accurately retrievable and displayed. Each invoice's attributes, including bill number, date, products, quantities, amounts, and taxes, were checked against input records for consistency.

Sorting and filtering the invoices by date, supplier, or total value were tested for accuracy. Testers used high-volume datasets to simulate realistic invoice histories and validated that pagination and loading speeds remained acceptable. Filters like date range and product filters were evaluated for performance.

Random invoice entries were selected and cross-checked with backend data to ensure no mismatches. UI clarity and data grouping were validated for readability. Any recently added invoices also appeared in the history immediately, proving data integrity.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Invoice history displayed all relevant data and performed well under load.
Additional Comments: NA

11. Requirement: Add Retailer – The user can add new retailers by filling in their and their firm's information and contact details. Each retailer is stored for future invoicing and tracking purposes.

Test Description: Testing for the Add Retailer functionality involved adding several new supplier records with varying details to verify input handling, validation, and proper data persistence. The form required fields such as supplier name, phone number, email, and firm name, all of which were tested using both typical and edge-case inputs to ensure robust form validation. Special care was taken to test input constraints such as invalid email formats, overly long phone numbers, and duplicate entries.

Upon submission, the system response and the post-save behavior were monitored. Successful entries were verified in the supplier list to ensure real-time reflection. Dropdowns were tested to

ensure existing suppliers were not duplicated erroneously. Field reset and cancel functionality were tested to validate proper user experience.

Testers also ensured that any system-generated IDs or tags linked to suppliers were unique and correctly associated with future invoice entries. The UI experience remained stable, with no performance lags.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Retailer records were added correctly with validations working as expected.
Additional Comments: NA

12. Requirement: View Retailers – The user can view a list of all retailers with detailed attributes and search/filter them as needed.

Test Description: The View Retailer functionality was tested to ensure all saved retailer records were correctly retrieved and displayed. The retailer listing interface was analyzed for clarity, completeness of information, and ease of navigation. Testers ensured each entry showed details such as retailer name, phone number, email, and transaction history.

Filtering was tested using names and firm's name and alphabetical order. Searches were executed using both exact matches and partial strings to ensure a robust search experience. The pagination behavior and responsiveness on long retailer lists were examined thoroughly.

Edge cases such as no retailer data, newly added retailers, and high volumes were tested to validate system scalability. Additionally, testers cross-checked live data updates after editing retailer details to ensure real-time accuracy.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Retailers were displayed accurately, and all sorting/filtering/searching functionalities worked.
Additional Comments: NA

13. Requirement: Add Outward Invoice Bill – The user can generate a new invoice by selecting a retailer and adding product(s) to the invoice.

Test Description: To verify the Add Retailer Invoice Bill functionality, testers created invoices by selecting retailers and adding multiple products to simulate realistic sales scenarios. Each invoice included inputs for quantities, selling prices, discounts, and applicable taxes. The correctness of all calculations, including subtotal, total with taxes, and discount application, was manually verified.

Form validation was tested for missing fields, invalid numbers, and duplicate invoice IDs. Testers evaluated system responses to extremely large product lists and high transaction values. Interface

performance remained stable, and the invoice preview feature showed accurate summaries before final submission.

After submission, each invoice was checked for proper integration with inventory reduction and retailer history logs. The format of invoice records was reviewed for clarity and correctness. We also verified whether changes in product prices or availability were accurately reflected during invoice creation.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Invoices were generated correctly, and inventory was adjusted accordingly.
Additional Comments: NA

14. Requirement: Out Invoice List – The user can access a history of all invoice bills raised for each retailer.

Test Description: The View Retailer Invoice Bills History module was tested by reviewing previously created invoices associated with various retailers. Testers validated invoice details such as invoice number, date, items sold, amounts, discounts, taxes, and final billed totals. Historical invoices were cross-referenced with actual transactions for accuracy.

The sorting and filtering features were tested using retailer name, invoice date, and invoice value. Date-range filters were used to validate UI behavior and system performance under varied query loads. The invoice list dynamically updated after new invoices were added, confirming real-time sync with backend systems.

High-volume data sets were used to simulate real business use cases. Edge cases like retailers with no invoices and retailers with extensive histories were both tested. In all cases, the user interface remained consistent, data was retrieved efficiently, and invoice records remained unaltered.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Invoices were displayed correctly with complete data, and the search/filter worked well.
Additional Comments: NA

15. Requirement: Add Inward Transactions – The user can record payments made by retailers to settle their outstanding dues.

Test Description: The Payment Inflow functionality was tested by simulating various payment scenarios where retailers paid their dues. Testers selected retailers with pending balances, choose appropriate payment dates, and entered partial and full payment amounts. The system was validated to ensure accurate deduction of outstanding balances.

The comment field was used to log specific notes, such as payment mode or special agreements. Multiple retailers were selected in test cases to verify batch payment recording. Testers intentionally left fields blank or entered invalid amounts to test validation and error prompts.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Payments from retailers were correctly recorded, and dues were updated accurately.
Additional Comments: NA

16. Requirement: Add Outward Transactions – The user can record payments made to suppliers against the company's outstanding dues.

Test Description: Payment Outflow was tested using various supplier debt-clearing scenarios. Testers selected suppliers with pending balances, specified payment amounts and dates, and included remarks about the payment mode or conditions. The system correctly reduced the pending amounts owed to suppliers upon each recorded transaction.

Batch outflows to multiple suppliers were tested to ensure data integrity and correct UI behavior. Form validations for incorrect amounts, blank supplier selections, and overlapping entries were thoroughly examined. The application responded with appropriate error messages and prevented incorrect submissions.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Payments to suppliers were accurately logged, and dues decreased accordingly.
Additional Comments: NA

17. Requirement: Pending Transactions – The user can view pending dues that retailers need to pay or the company owes to suppliers.

Test Description: Pending Transactions functionality was tested by verifying displayed balances for both retailers and suppliers. Testers ensured the system correctly accumulated and displayed pending amounts after invoice creation or partial payments. Multiple test scenarios covered cases with no dues, full dues, and mixed balances.

Testers created new invoices and recorded partial payments to simulate real-world use. The Pending Transactions list updated immediately and consistently.

Scenarios also included edge cases where dues should be recalculated. The test team validated system notifications or alerts regarding overdue payments. Data accuracy between the pending screen and individual invoice or transaction histories was carefully cross-verified.

Test Owner: While(1)
Test Date: 05/04/2025

Test Results: Pending dues are displayed accurately and updated in real time with transactions.
Additional Comments: NA

18. Requirement: View Transaction History – The user can view historical payment transactions either received from retailers or paid to suppliers.

Test Description: The View Transaction History module was thoroughly tested for accurate representation of all past financial transactions. Users could toggle between payments received from retailers and payments made to suppliers. Each transaction record displayed the Firm name, person name, date, amount, and comments.

The system's filters for date range, type (inflow/outflow), and party were tested with a variety of inputs. Testers created transactions, then confirmed their presence in the history log. UI responsiveness and clarity were evaluated on both desktop and mobile views.

Scenarios involving bulk transactions, backdated entries, and amended payment records were validated for correctness. Testers also verified that exporting or printing transaction history worked as expected. The edge cases tested included no transaction records and attempts to filter with non-matching queries.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Transaction records were displayed accurately.
Additional Comments: NA

19. Requirement: Profile – The user can view and edit their profile information.

Test Description: The Profile module was tested for both viewing and editing functionalities. Initially, testers verified that all user information such as name, contact details and address were correctly fetched and displayed. All fields were checked for accuracy and visibility across multiple screen sizes.

Editing capabilities were then assessed by modifying each field, saving the changes, and ensuring they were correctly stored and reflected upon page refresh. Various inputs were tested, including valid updates, incomplete fields, invalid email/phone number formats, and blank submissions. The system displayed appropriate warnings or confirmations accordingly.

The responsiveness of the form elements, their validation messages, and feedback prompts were checked thoroughly.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Profile details were viewed and edited successfully; validations functioned correctly.
Additional Comments: NA

20. Requirement: Contact Us – The user can view the contact details of the application's creators, including various communication channels.

Test Description: The Contact Us module was tested for its ability to correctly display the application's official communication details. Testers verified that email addresses, phone numbers, Instagram handles, LinkedIn profiles, and any other listed contact methods were correctly shown and hyperlinked where appropriate.

Scenarios included validating the formatting of all contact information, checking for broken links, and clicking through each contact method to ensure they open in the correct application (e.g., mail client, browser). Any typos, incorrect numbers, or outdated handles were flagged.

The design was checked for responsiveness and clarity across screen sizes. Contact details remained visible and accessible regardless of layout. The user experience was reviewed to ensure that no interaction or contact required login or authentication.

Accessibility features such as screen reader support and alt text for icons, were also reviewed. Any unnecessary or misleading labels were removed. Suggestions for improvements, like direct inquiry forms or automated chat options, were documented for future enhancements.

Test Owner: While(1)
Test Date: 05/04/2025
Test Results: Contact information was displayed clearly, and all links and formats functioned properly.
Additional Comments: NA

21. Requirement: Profit/Sales Analysis Graph - The system should display accurate profit timeline graphs based on user-selected date ranges.

Test Description: The user navigates to the "Analysis" section where the system displays a profit graph showing timeline data between default dates (today vs. one week ago). The user can select new date ranges using the date pickers and click "Get" to update the graph. The system immediately redraws the graph to show profit data for the selected period. The adjacent statistics panel correctly displays the average profit, maximum profit, average sales and maximum sales for the selected period. The user verifies that graph data matches manual calculations from transaction records. The graph should be responsive and maintain readability when date ranges are changed. All axes should be properly labeled and the graph should render without visual glitches.

Test Owner: While(1)

Test Date: 2025-04-03

Test Results: Profits and sales graph are shown without any glitch and Max, Avg Metrics are updated correctly by changing the date range

Additional Comments: NA

22. Requirement: The system should provide multiple graphical views of sales data with configurable parameters.

Test Description : In the "Analysis" section below the profit graph, the system displays three additional graphs: bills generated over time, inward transactions timeline, and top 5 retailers/suppliers. The user can toggle the third graph between retailers and suppliers view. The Bills graph and inward transaction should show for a static date range of 2 months (if data is available) . The top 5 graph should correctly rank entities by total bill amount and display clear labels. The user can confirm that switching between

Test Owner: While(1)

Test Date: 2025-04-03

Test Results: Bills count and the transactions were shown correctly at a period of 2 months and also bar graphs for suppliers and retailers are shown correctly.

Additional Comments: NA

23. Requirement: The system should display a detailed table of product sales that can be filtered by time period.

Test Description : Below the analysis graphs, the system shows a table of sales per product filtered to one month by default. The user can select 3-month or 6-month periods using the provided buttons. The table updates immediately to show sales data for the selected period, including product name, total quantity sold, and total revenue. The "View More" button expands the table in a new page to show product sales of all products. The user verifies that the table data matches expectations based on bill records. Sorting by columns should work properly. The period selection buttons should clearly indicate the active selection. The table should maintain proper formatting when expanded or when displaying large datasets.

Test Owner: While(1)

Test Date: 2025-04-03

Test Results: Table Correctly represented the product per sales for each of the time period.

Additional Comments: NA

NON-FUNCTIONAL REQUIREMENTS

1. Performance Requirements

- The system’s response time for loading web pages should not exceed 100 milliseconds to deliver a fast, responsive user experience. Reducing latency is essential to prevent user frustration, maintain engagement, and ensure optimal performance.

- When users view their inventory, which may span multiple pages, the application should efficiently handle pagination or infinite scrolling. Loading additional data must not cause any noticeable slowdown, maintaining a consistent and smooth browsing experience regardless of data volume.

2. Safety and Security Requirements

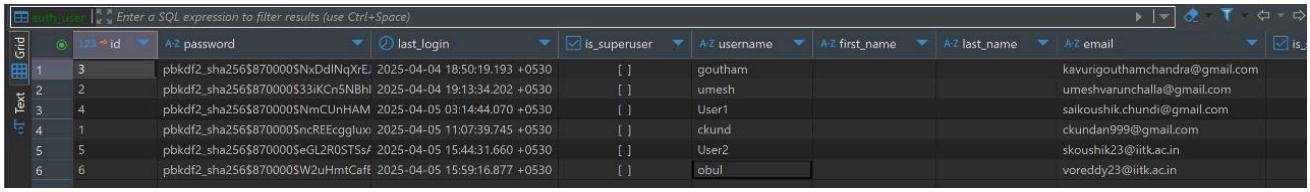
- The software authenticates the user during login using a strong password. a strong password would require at least 8 characters which are not commonly used and not similar to username. In the Sign-up page we put a restriction of at least 8 characters on the password field, without fulfilling this requirement signup is not allowed leading to an error message.

  To validate the password strength, we tested manually, thoroughly using various combinations of passwords. Through testing, we found a few bugs and resolved them by adding error messages accordingly.

- There is a provision to reset password in case the user forgets the password. In such a case, the user is verified via OTP (One Time Password) sent to the user’s registered email.

  To validate the OTP-based password reset functionality as per the specified requirement, we tested it manually. The system's ability to generate unique OTPs and deliver them to users' registered email addresses was confirmed after testing, along with verification of the OTP validity period set at 4 minutes. The forgot password process, from user-initiated requests to OTP-based verification, was tested, covering different edge cases such as a different email being provided that is not registered, wrong OTP entered, and OTP entered after 4 minutes. Each time, our deployed website behaved as expected, maintaining security. Additionally, security measures implemented for OTP generation and transmission were evaluated, like OTP should be very random each time. Throughout testing, no bugs or issues were encountered, affirming the reliability and effectiveness of the OTP-based password reset feature, which provided users with a secure and seamless password recovery experience.

- Passwords and OTPs were stored in a hashed form using SHA-256 encryption. which restricts even administration from viewing the data.

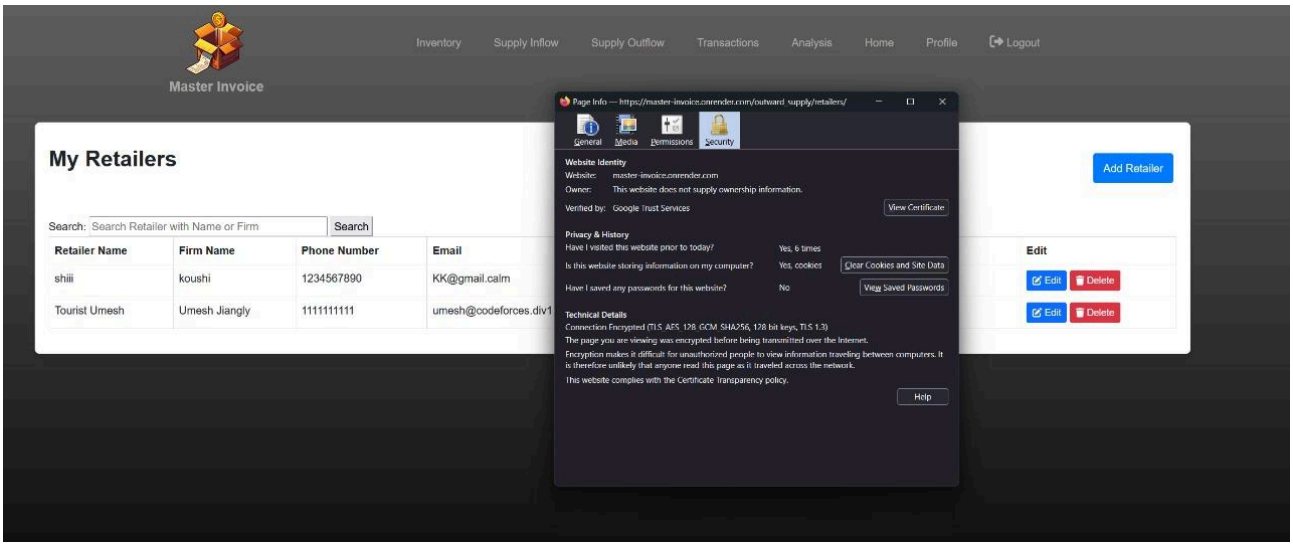


	id	password	last_login	is_superuser	username	first_name	last_name	email
1	3	pbkdf2_sha256$870000$NxDdlNqXrE…	2025-04-04 18:50:19.193 +0530	[ ]	goutham			kavurigouthamchandra@gmail.com
2	2	pbkdf2_sha256$870000$33iKCn5NBhl…	2025-04-04 19:13:34.202 +0530	[ ]	umesh			umeshvarunchalla@gmail.com
3	4	pbkdf2_sha256$870000$NmCUnHAM…	2025-04-05 03:14:44.070 +0530	[ ]	User1			saikoushik.chundi@gmail.com
4	1	pbkdf2_sha256$870000$ncREEcgglux…	2025-04-05 11:07:39.745 +0530	[ ]	ckund			ckundan999@gmail.com
5	5	pbkdf2_sha256$870000$eGL2R0STSs…	2025-04-05 15:44:31.660 +0530	[ ]	User2			skoushik23@iitk.ac.in
6	6	pbkdf2_sha256$870000$W2uHmtCaff…	2025-04-05 15:59:16.877 +0530	[ ]	obul			voreddy23@iitk.ac.in

- To prevent the prediction of OTP, view large sampling, secrets library to generate cryptographically secure random numbers.

```python
# OTP
def generate_secure_otp(length=6):
    return ''.join(str(secrets.randbelow(10)) for _ in range(length))
# Hash the OTP using SHA-256
def hash_otp(otp):
    return hashlib.sha256(otp.encode()).hexdigest()
#OTP storage
```

- All connections to the server should use Transport Layer Security (TLS 1.2/1.3) encryption.All transaction data should be encrypted properly.



This is from our deployed site. We are using TLS1.3 encryption for data.

3. Maintainability:

- The software has been designed in an organized manner so that new features can be added and modifications can be done very easily within two working days. Various components of the software are well organised into folders.

4. Reliability:

- The system demonstrates exceptional robustness and consistent performance under load. During benchmark testing, it exhibited minimal latency and zero request failures, confirming its ability to handle concurrent traffic effectively. The website maintains high availability with significantly reduced downtime, ensuring continuous and reliable access for end users. This level of dependability not only enhances the overall user experience but also strengthens user trust in the platform's stability and responsiveness.

5. Compatibility:

- As a web-based application, the software offers seamless accessibility across any device equipped with a modern web browser. Extensive testing confirmed consistent performance and full functionality on major browsers including Google Chrome, Mozilla Firefox, and Microsoft Edge. This ensures a broad reach and a smooth user experience regardless of the user's preferred platform.

6. Usability:

- The application is designed with user-friendliness at its core. Users can effortlessly update their profiles, manage transaction records with vendors or partners, and establish smooth connections with product distributors, all contributing to a streamlined supply process.

- Additionally, the integrated search functionality enhances navigation and efficiency. It supports partial keyword matches, enabling users to locate desired products quickly and accurately with minimal effort.

7. Scalability:

- We've tested the system with 200 concurrent users using Apache HTTP Server tools
- no errors were found in testing

```
┌──(kali㉿LAPTOP-83DUK7TD)-[~]
└─$ ab -n 200 -c 20 https://master-invoice.onrender.com/
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking master-invoice.onrender.com (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests


Server Software:        gunicorn
Server Hostname:        master-invoice.onrender.com
Server Port:            443
SSL/TLS Protocol:       TLSv1.3,TLS_AES_256_GCM_SHA384,256,256
Server Temp Key:        X25519 253 bits
TLS Server Name:        master-invoice.onrender.com

Document Path:          /
Document Length:        7386 bytes

Concurrency Level:      20
Time taken for tests:   5.159 seconds
Complete requests:      200
Failed requests:        0
Total transferred:      1593800 bytes
HTML transferred:       1477200 bytes
Requests per second:    38.77 [#/sec] (mean)
Time per request:       515.906 [ms] (mean)
Time per request:       25.795 [ms] (mean, across all concurrent requests)
Transfer rate:          301.69 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       32   47  73.0     41    1072
Processing:   179  322 144.5    309    1634
Waiting:      178  321 144.6    308    1633
```

However, to accurately gauge its scalability, it needs to be made available to the general public. Only through widespread usage can we gather sufficient data to assess its performance under higher loads

OTHER REQUIREMENTS

1.Authentication:

- user will be verified with OTP sent to via mail when registering using google app API.

5 Conclusion

5.1 How Effective and exhaustive was the testing?

We were able to achieve complete code and branch coverage. We tested each API to ensure that the database undergoes the required changes. We resolved a number of bugs and took care of all cases. Appropriate alerts have been implemented incase of invalid inputs and edge cases were tested. Examples of such invalid inputs are number of digits in a phone number not being equal to 10 and date input for invoice bills being a future date.

Every component was tested by more than one developer and we ensured that developers tested components which were not developed by them.

5.2 Which components have not been tested adequately?

We were unable to test some non functional requirements mentioned in the SRS document such as the throughput of the system being able to handle 1000 concurrent users and scalability. We were only able to test for 200 concurrent users. These features could benefit from further testing to ensure user satisfaction.

5.3 What difficulties have you faced during testing?

We experienced difficulty using the Django testing framework during the unit tests.

5.4 How could the testing be improved?

The testing could have used more automation for the integration testing. Developers often make implicit assumptions based on their technical understanding of the system—such as expecting users to follow certain workflows or input formats. However, these assumptions may not hold true for end-users who lack the same context, leading to usability issues or unexpected behavior.

Automation testing would have helped bridge this gap by systematically validating how the software behaves under diverse and unpredictable user scenarios.

Appendix A - Group Log

The table only records the meetings among group members. Each member has been consistently working on their assigned tasks, and the actual time and effort invested by the team far exceed what is documented here.

SL. No.	Date	Timings	Venue	Description
1	28/03/2025	17:00 - 19:30	RM Building	Discussed various types of testing.Distributed the tasks of various types of testing among subgroups with in the team.
2	29/03/2025	17:00 - 19:00	RM Building	Discussed various doubts faced by each subgroup in their respective tasks.
3	02/04/2025	16:00 - 18:30	KD Library	All the group members met and every subgroup updated each other with their work..
4	03/04/2025	14:00 - 19:00	KD Library	Discussed doubts among team members.
5	05/04/2025	14:30 - 18:00	KD Library	Reviewed the testing document. Cleared various doubts in the testing document and finalised the testing document.