

FINAL PROJECT REPORT

PROJECT TITLE : E-Commerce Grocery Store

Team ID : 06

Name: Agan Peter Pidaparthi, UTA ID: 1002074432

Name: Ganji Jaya Sravani, UTA ID: 1002112714

Name: Lalitha Nagasai Nagaraju, UTA ID: 1002114097

PROJECT DESCRIPTION :

Our objective is to create an e-commerce website for a grocery store that will give consumers an easy way to browse, choose, and buy food online. The website will provide a large selection of goods, such as fresh vegetables, household goods, beverages, pantry necessities, etc. Our goal is to develop an online grocery shopping experience that is as seamless as possible while maintaining customer loyalty.

PROJECT PROPOSAL :

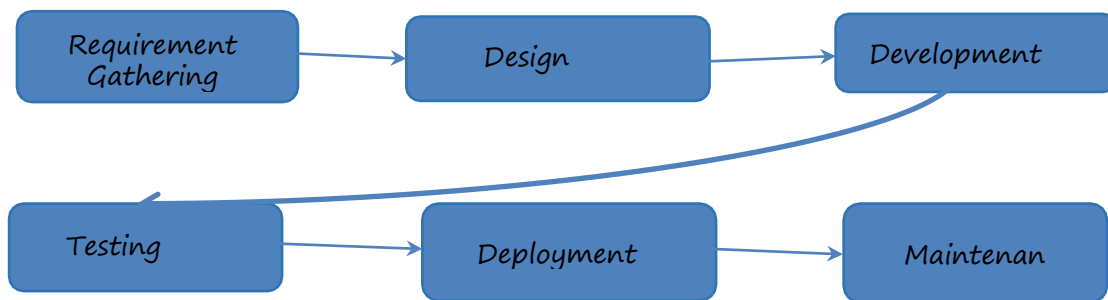
The Objective of the E-commerce Grocery project is to improve accessibility, efficiency, and client satisfaction by creating a user-friendly e-commerce platform that allows customers to purchase goods online conveniently. via features like search functionality and an efficient checkout procedure, the platform will offer a large selection of supermarket products, speed up the shopping experience, and encourage repeat business via rewards and promotional offers. The project seeks to provide scalability to support future expansion while meeting the changing needs of modern consumers by utilizing technology to make grocery shopping more efficient and convenient.

Desired Requirements/Constraints:

1. User registration and login system for both customers and administrators.
2. Product catalog with categories, descriptions, images, and pricing.
3. Search functionality allowing users to find products easily.
4. Shopping cart feature to add, remove, and modify items before checkout.
5. Secure payment gateway integration for online transactions.
6. Order tracking system to monitor the status of placed orders.
7. User review and rating system for products.
8. Scalability to handle increasing traffic and product inventory.
9. Compliance with data protection regulations to ensure customer privacy

SDLC Model :

For the Grocery Shopping website project, the most suitable SDLC model would be the Agile Model. Agile methodologies are highly suitable. Agile provides flexibility, adaptability, and the ability to respond to changes quickly, which are crucial in the fast-paced and evolving environment of app development. Among Agile methodologies, Scrum is commonly chosen for its structured approach to iterative development and emphasis on collaboration and communication.



PROJECT TIMELINE

The screenshot shows the Monday.com interface for a project named 'Grocery Store'. The timeline is organized into a table with columns for Project, Person, Status, Estimated Completion, Timeline, Date Completed, Test, Planned Effort, and Effort Spent. The project is broken down into sub-items, including Planning, Design, Development, Deployment, Testing, and Maintenance. The Development phase is further detailed with sub-items like Generating Data, working on data, Designing UI pages, Query processing, and Integration. The timeline shows the project is currently in the Development phase, with a planned completion date of March 30th. The Effort Spent column shows the total effort spent on the project is 107 hours.

Project	Person	Status	Estimated Comp...	Timeline	Date Completed	Test	Planned Effort	Effort Spent	+
Planning	JD	Done	Feb 16	Feb 11 - 16	Feb 16	Completed	9 hours	8 hours	
Design	JD	Done	Feb 25	Feb 17 - 25	Feb 25	Completed	17 hours	19 hours	
Development	JD	Done	Mar 30	Feb 26 - Mar 27	Mar 28	UI screens are developed and response is gene...	47 hours	50 hours	
Subitem	Owner	Status	Estimated Comple...	Timeline	Date completed	Test			+
Generating Data	JD	Done	Mar 2	Feb 26 - Mar 2	Mar 2				
working on data	JD	Done	Mar 8	Feb 27 - Mar 7	Mar 7				
Designing UI pages	JD	Done	Mar 14	Feb 28 - Mar 14	Mar 14				
Query processing	JD	Done	Mar 29	Mar 11 - 29	Mar 28				
Integration	JD	Done	Mar 29	Mar 7 - 29	Mar 28				
Deployment	JD	Done	Apr 5	Mar 17 - Apr 5	Apr 4	Deployed locally	9 hours	7 hours	
Testing	JD	Done	Apr 15	Apr 11 - 15	Apr 15	Manual & Selenium Automation testing done	15 hours	17 hours	
Maintenance	JD	Done	Apr 25	Apr 8 - 25	Apr 24	Check for any changes and deployment	10 hours	8 hours	
				Feb 11 - Apr 25			107 hours turn	109 hours turn	

COST AND TIME ESTIMATION

To calculate the cost using COCOMO, we'll use the provided information:

Effort: 3.0 Person-months

Total Equivalent Size (SLOC): 1015 SLOC

Effort Adjustment Factor (EAF): 1.00

Average Monthly Cost per Person: (we'll assume \$6000 as an example)

Schedule: (we'll use the given Effort and assume a schedule of 3 months for simplicity)

First, we need to calculate the cost per month:

Average Monthly Cost per Person = Cost / (Effort * Schedule)

Average Monthly Cost per Person = \$134 / (3.0 * 3)

Average Monthly Cost per Person = \$134 / 9

Average Monthly Cost per Person \approx \$14.89 (rounded to two decimal places)

Now, we can calculate the total cost:

Cost = Effort * Average Monthly Cost per Person * Schedule

Cost = 3.0 * \$14.89 * 3

Cost \approx \$134.01

So, the estimated cost of the project using COCOMO is approximately \$134.01.

COCOMO II - Constructive Cost Model

Software Size: Sizing Method: **Source Lines of Code**

Software Size Probability Distribution

Distribution Type: **Normal**
Iterations: **1000**

Software Size	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	700					
Reused	650	0	0	20	3	
Modified	715	10	20	70	5	96

Software Scale Drivers

Precedentedness: **Nominal**
Development Flexibility: **Nominal**
Architecture / Risk Resolution: **Nominal**
Team Cohesion: **Nominal**
Process Maturity: **Nominal**

Software Cost Drivers

Product
Required Software Reliability: **Nominal**
Data Base Size: **Nominal**
Product Complexity: **Nominal**
Developed for Reusability: **Nominal**
Documentation Match to Lifecycle Needs: **Nominal**

Personnel
Analyst Capability: **Nominal**
Programmer Capability: **Nominal**
Personnel Continuity: **Nominal**
Application Experience: **Nominal**
Platform Experience: **Nominal**
Language and Toolset Experience: **Nominal**

Platform
Time Constraint: **Nominal**
Storage Constraint: **Nominal**
Platform Volatility: **Nominal**

Project
Use of Software Tools: **Nominal**
Multisite Development: **Nominal**
Required Development Schedule: **Nominal**

Maintenance: **Off**

Software Labor Rates
Cost per Person-Month (Dollars): **45**
Calculate

Software Labor Rates

Cost per Person-Month (Dollars)

45

Calculate

Results

Software Development (Elaboration and Construction)

Staffing Profile

Effort = 3.0 Person-months
Schedule = 5.2 Months
Cost = \$134

Your project is too small to display a staffing profile due to truncation.

Total Equivalent Size = 1015 SLOC
Effort Adjustment Factor (EAF) = 1.00

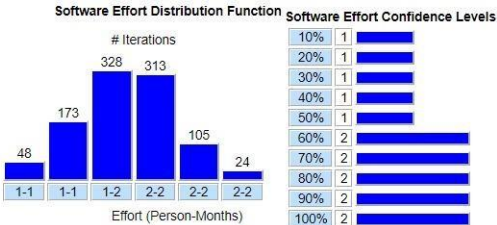
Acquisition Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.2	0.6	0.3	\$8
Elaboration	0.7	1.9	0.4	\$32
Construction	2.3	3.2	0.7	\$102
Transition	0.4	0.6	0.6	\$16

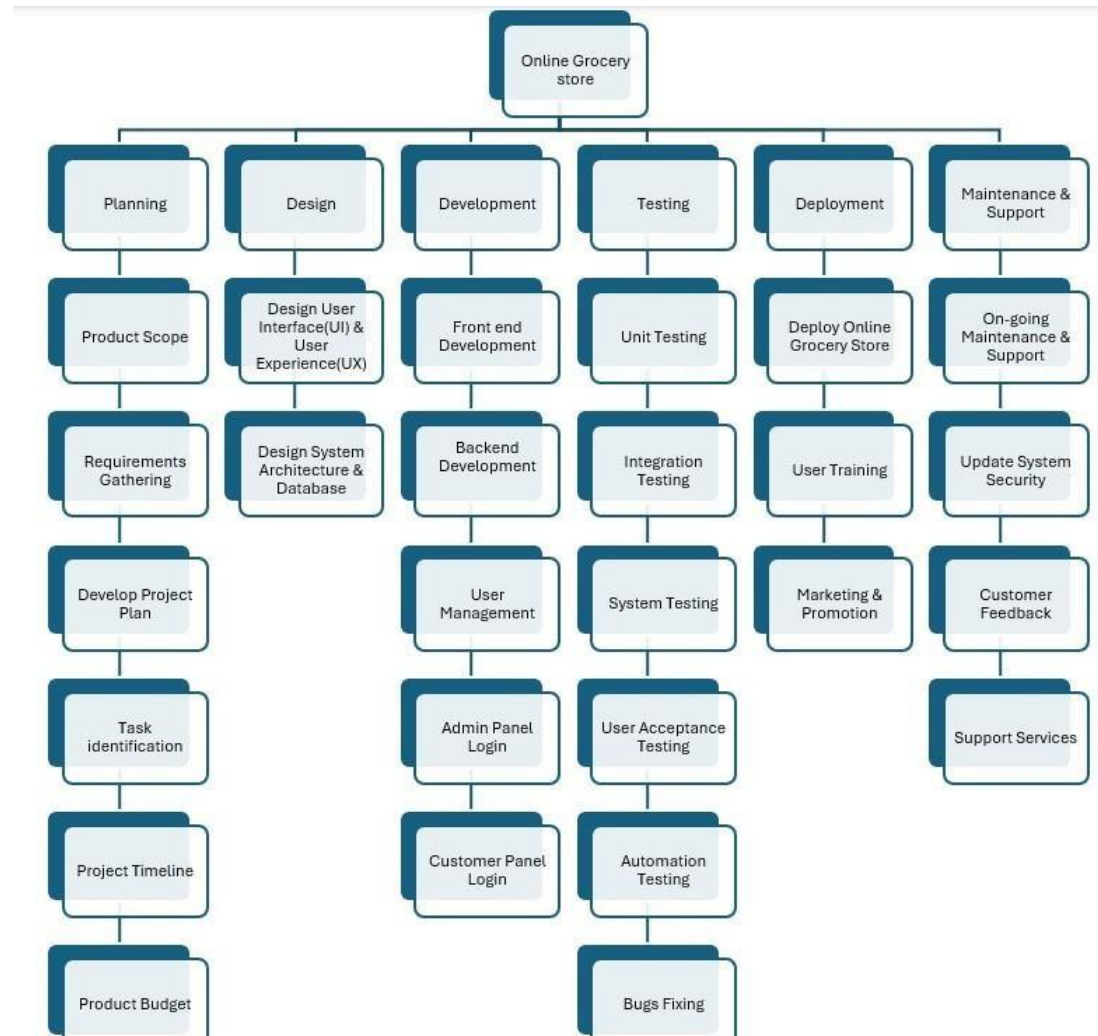
Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.0	0.1	0.2	0.1
Environment/CM	0.0	0.1	0.1	0.0
Requirements	0.1	0.1	0.2	0.0
Design	0.0	0.3	0.4	0.0
Implementation	0.0	0.1	0.8	0.1
Assessment	0.0	0.1	0.5	0.1
Deployment	0.0	0.0	0.1	0.1

Acquisition Monte Carlo Results

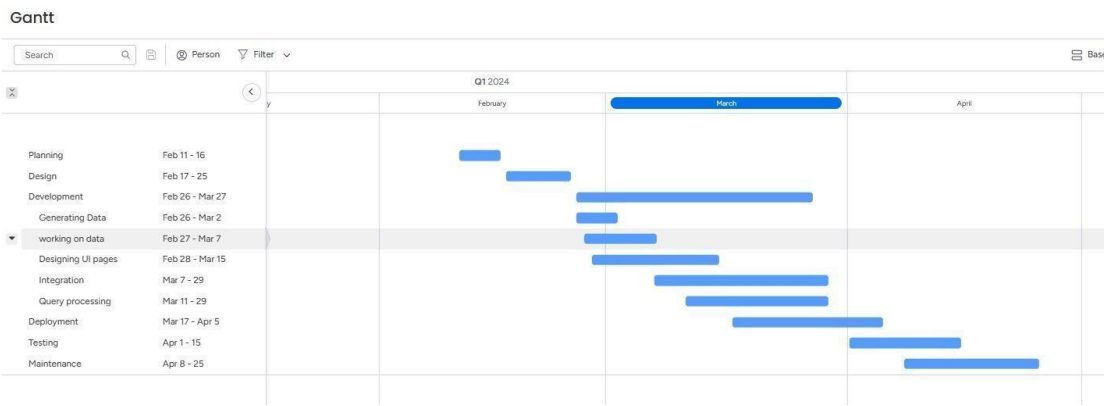
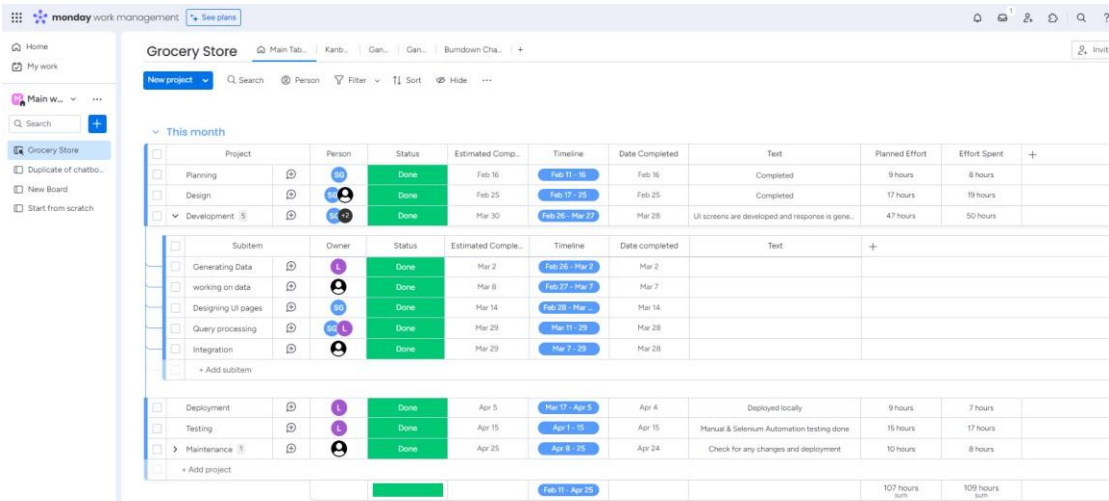


WORK BREAKDOWN STRUCTURE



PROJECT SCHEDULE

We used a Gantt chart to visualize the project schedule. It provides a clear and visual representation of the project timeline, showing when each task starts and ends relative to other tasks. This helped us to manage the project and understand the sequence of activities, dependencies between tasks, and overall timeline of the project. We also relied on Monday.com for organizing tasks and collaborating effectively. Its customizable features helped us stay on track and meet deadlines efficiently.



The above Gantt chart illustrates a project schedule. This particular Gantt chart shows.

the schedule for a project with the following phases:

- Planning (Feb 11-16)
- Development (Feb 26-Mar 27)
- Generating Data (Feb 26-Mar 2)
- Working on Data (Feb 27-Mar 7)
- Designing UI Pages (Feb 28-Mar 15)

- Integration (Mar 7-29)
- Query Processing (Mar 11-29)
- Deployment (Mar 17-Apr 5)
- Testing (Apr 1-15)
- Maintenance (Apr 8-25)

The bars in the chart represent the tasks, and the length of the bars shows the duration of each task. The vertical line at the left of the chart indicates the current date (April 29, 2024).

RISK FACTORS AND RISK ANALYSIS

Risk Identification:

Technology Risk:

- Risk Factor : Dependencies for API integration and mapping services
- Probability : Moderate (25-50%)
- Risk Planning : Implement graceful degradation strategies

Tools Risk :

- Risk Factor : Rely on specific development tools or platforms that might be unsupported
- Probability : Low (10-25%)
- Risk Planning : Evaluating toolsets, flexibility in technology choices and have migration plan

Requirement Risks :

- Risk Factor : Incomplete or ambiguous requirements that may lead to misunderstanding
- Probability : High (50-75%)
- Risk Planning : Comprehensive requirement gathering, implementing a robust process.

Estimation Risk :

- Risk Factor : Underestimation of project timeline leading to schedule delays
- Probability : Moderate (25-50%)
- Risk Planning : Regular review and adjust project plans based on progress

RISK IDENTIFICATION	RISK ASSESSMENT	RISK MITIGATION PLAN
Technology Risk : Dependency on third-party API's	Moderate (25-50%)	Implement graceful degradation strategies
Tool Risk : Compatibility issues with chosen development tools/platforms	Low (10-25%)	Evaluating toolsets, flexibility in technology choices and have migration plan
Requirements Risk : Ambiguous or incomplete project requirements	High(50-75%)	Comprehensive requirement gathering, implementing a robust process
Estimation Risk : Underestimation of project timeline	Moderate(25-50%)	Regular review and adjust project plans based on progress

PROJECT QUALITY ASSURANCE

1. Requirements Gathering and Analysis:

- Thorough analysis to understand the scope, objectives, and expected outcomes.
- Verify that requirements are documented accurately and are aligned with the expectations

2. Quality Planning :

- Develop a detailed quality management plan outlining quality objectives, standards and metrics.
- Define accuracy of product listing, smooth checkout process.
- Establish quality metrics like page load times, error rates during checkout and order accuracy.

3. Design Phase :

- Conduct design reviews to validate UI/UX designs, navigation flows, and visual elements.
- Conduct design reviews and walkthroughs

4. Development and Testing :

- Implement coding standards for front-end and back-end development.
- Conduct Unit Testing for Individual components.
- Perform integration testing to ensure seamless interaction between different modules
- Validate end-to-end scenarios such as user registration, Product search, order placement, and delivery scheduling.

5. Quality Assurance and Control :

- Implemented processes such as code reviews, automated testing and manual testing.
- Used Selenium for regression testing, API testing to ensure consistency and reliability.

By following these quality practices throughout the project lifecycle, we were able to enhance reliability, usability and overall quality of our project.

Entity-Relationship Diagram (ER Diagram) for our grocery shopping website:

Entities:

- **User:** This class represents a user of the grocery store system. It has attributes such as userID (unique identifier for the user), username, email, password, and address. It also has methods like addUser() to add a new user, deleteUser() to delete a user, and updateUser() to update a user's information.
- **Admin:** This class likely represents the system administrator and might have functionalities to manage users and products (add, delete, and update). However, the specific functionalities are not shown in the provided class diagram.
- **Product:** This class represents a product sold in the grocery store. It has attributes such as productID (unique identifier for the product), name, description, price, category (e.g., produce, meat, dairy), imageURL (link to the product image), and dietary_restrictions (e.g., vegan, gluten-free). It also has methods like addProduct() to add a new product, deleteProduct() to delete a product, and updatePrice() to update the price of a product.

- **Order:** This class represents an order placed by a user. It has attributes such as orderID (unique identifier for the order), userID (foreign key referencing the User class), orderStatus (e.g., placed, processing, shipped, delivered), and a list of products in the order. It also has methods like trackOrder() to track the status of an order, cancelOrder() to cancel an order, and likely updateOrder() to update the order details (not shown in the class diagram).

- **Payment:** This class represents a payment made for an order. It has attributes such as paymentID (unique identifier for the payment), userID (foreign key referencing the User class), orderID (foreign key referencing the Order class), payment_amount, and payment_date. It also has methods like addPayment() to add a new payment, deletePayment() to delete a payment, and updatePayment() to update the payment details.

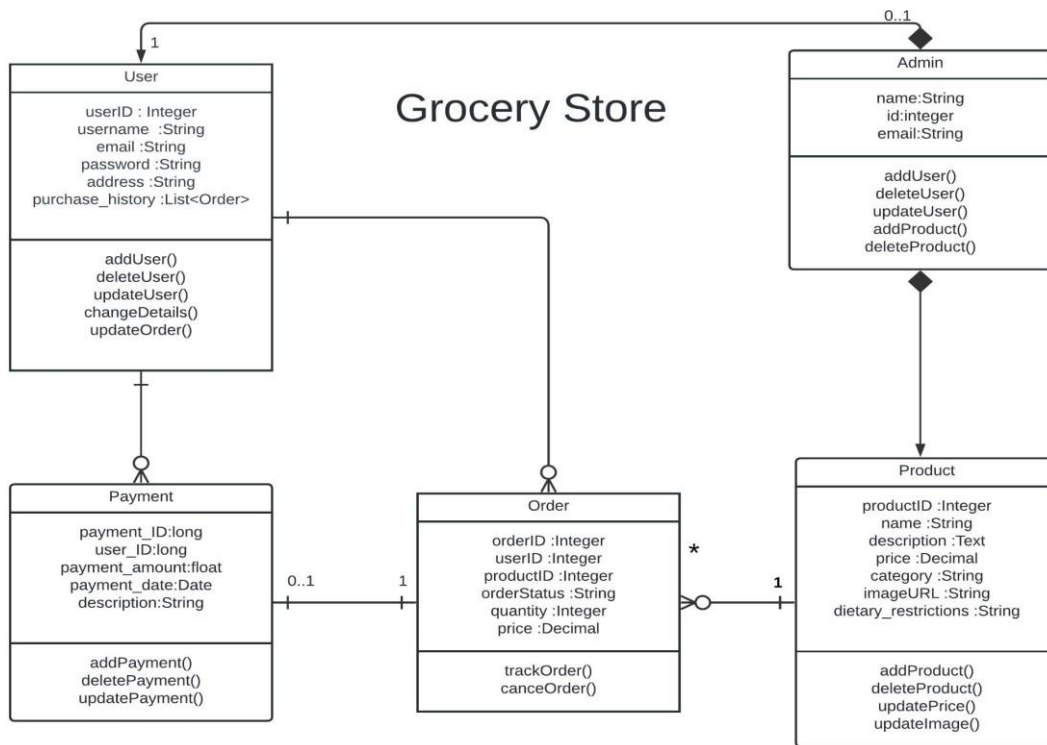
Relationship Connection:

The relationships between the classes are as follows:

- **User:** - A user can place many orders (indicated by the asterisk (*) next to userID in the Order class).
- **Order:** - An order is placed by one user (indicated by the foreign key userID referencing the User class). - An order can contain many products (indicated by the many-to-many relationship between Order and Product). - An order has one payment (indicated by the foreign key orderID referencing the Payment class).
- **Payment:** - A payment is made by one user (indicated by the foreign key userID referencing the User class). - A payment is made for one order (indicated by the foreign key orderID referencing the Order class).

Below UML Diagram:

- Entities are represented by rectangles.
- Attributes are listed within each entity.
- Relationships between entities are depicted by lines.
- Cardinality (relationship type) is indicated near each relationship line.
- The lines indicate the direction of the relationship and the cardinality (one-to-many, many-to-one, etc.).



IMPLEMENTATION:

Planning and Scheduling for Grocery Shopping Website (Scrum Approach):

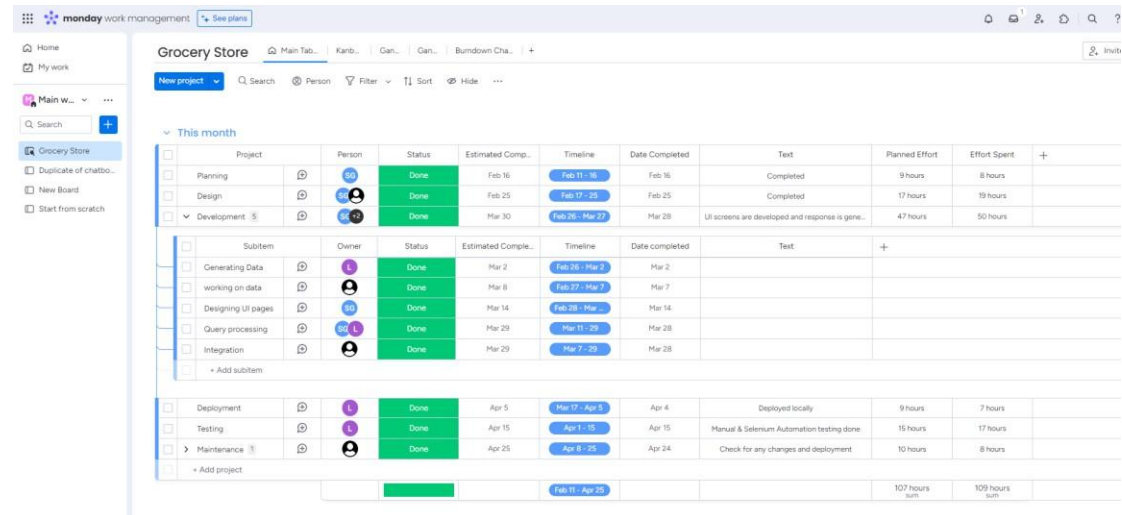
Planning Tools:

Product Backlog: This is a prioritized list of all features, requirements, and bug fixes for the project. User stories are typically added here first, and the development team selects them for each sprint based on priority and feasibility.

Product Roadmap: This is a visual representation of the project timeline, outlining major milestones and feature releases.

Burn Down Chart: This helps visualize the remaining work in a sprint by tracking completed tasks against the total estimated effort.

Task Management Tools: Online tools like Jira, Monday.com or Asana can help manage tasks, track progress, and collaborate within the team.



The screenshot shows the Monday.com interface for a project named 'Grocery Store'. The table below represents the data visible in the 'This month' view.

Project	Person	Status	Estimated Comp.	Timeline	Date Completed	Text	Planned Effort	Effort Spent	+
Planning	[Avatar]	Done	Feb 16	Feb 11 - 16	Feb 16	Completed	9 hours	8 hours	
Design	[Avatar]	Done	Feb 25	Feb 17 - 25	Feb 25	Completed	17 hours	19 hours	
Development	[Avatar]	Done	Mar 30	Feb 26 - Mar 27	Mar 28	UI screens are developed and response is gene...	47 hours	50 hours	
Subitem	Owner	Status	Estimated Comple.	Timeline	Date completed	Text			+
Generating Data	[Avatar]	Done	Mar 2	Feb 26 - Mar 2	Mar 2				
working on data	[Avatar]	Done	Mar 8	Feb 27 - Mar 7	Mar 7				
Designing UI pages	[Avatar]	Done	Mar 14	Feb 28 - Mar 14	Mar 14				
Query processing	[Avatar]	Done	Mar 29	Mar 11 - 29	Mar 28				
Integration	[Avatar]	Done	Mar 29	Mar 7 - 29	Mar 28				
Deployment	[Avatar]	Done	Apr 5	Mar 17 - Apr 5	Apr 4	Deployed locally	9 hours	7 hours	
Testing	[Avatar]	Done	Apr 15	Apr 11 - 15	Apr 15	Manual & Selenium Automation testing done	15 hours	17 hours	
Maintenance	[Avatar]	Done	Apr 25	Apr 8 - 25	Apr 24	Check for any changes and deployment	10 hours	8 hours	
							107 hours sum	109 hours sum	

Each column contains cards that represent tasks. Each card contains information about the task, such as:

- **Task Name:** A brief description of the task.
- **People Assigned:** The people who are assigned to complete the task.
- **Status:** The current status of the task (e.g., Done, Working on it, To Do).
- **Estimated Completion:** The estimated date for completing the task.
- **Timeline:** The actual dates the task was worked on.
- **Date Completed:** The date the task was completed.
- **Text:** Any additional details about the task.
- **Planned Effort:** The estimated amount of time (in hours) required to complete the task.
- **Effort Spent:** The amount of time (in hours) actually spent working on the task.

Task Name: Planning (This task is likely referring to the planning for the development of the grocery store planning tool, not the planning for running a grocery store)

Status: Done

Completed: February 16, 2024

Planned Effort: 9 hours

Effort Spent: 8 hours

Task Name: Design (This task likely refers to the design of the user interface for the grocery store planning tool)

Status: Done

Completed: February 25, 2024

Planned Effort: 17 hours

Effort Spent: 19 hours

Task Name: Development (This task likely refers to the fifth iteration of development on the grocery store planning tool)

Status: Done

Completed: March 28, 2024

Text: UI screens are developed and response is generated

Planned Effort: 47 hours

Effort Spent: 50 hours

Subtasks:

Generating Data (Done, Feb 26-Mar 2)

Working on data (Done, Mar 27-Mar 7)

Designing UI pages (Done, Feb 28-Mar 15)

Query processing (Working on it, Mar 29-Mar 28)

Integration (Working on it, Mar 7-29)

Task Name: Deployment (This task likely refers to deploying the grocery store planning tool to a production environment)

Status: Completed

Estimated Completion: April 5, 2024

Planned Effort: 10 hours

Effort Spent: 8 hours

Task Name: Testing (This task likely refers to testing the grocery store planning tool)

Status: Completed

Estimated Completion: April 15, 2024

Planned Effort: 15 hours

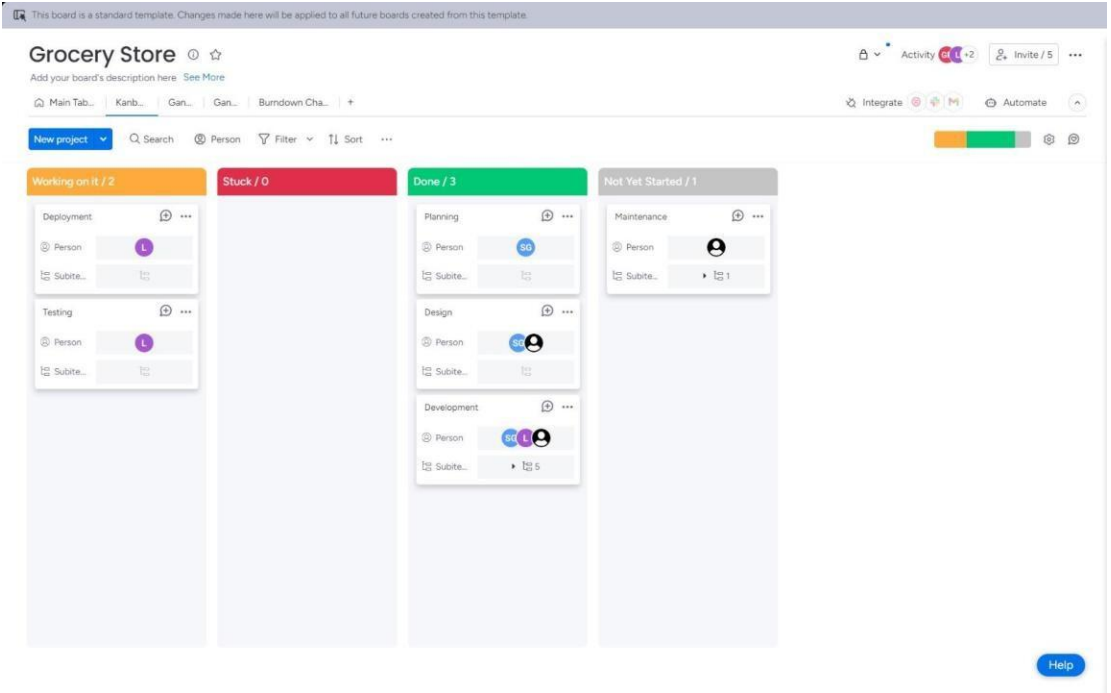
Task Name: Maintenance (This task likely refers to fixing bugs and making improvements to the grocery store planning tool after it is deployed)

Status: Completed

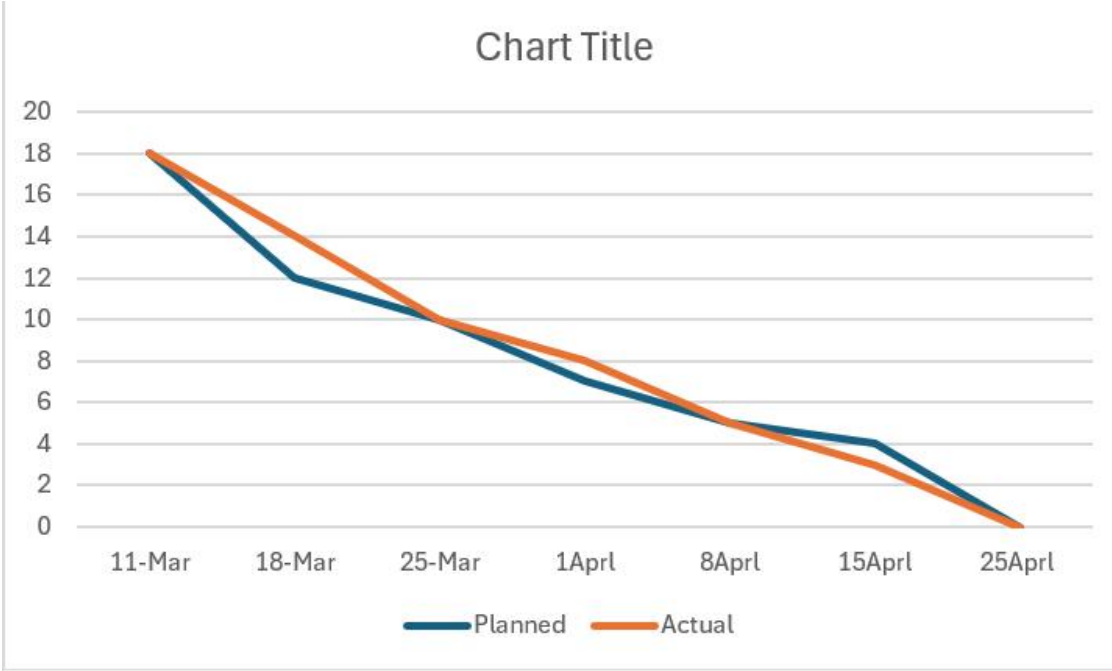
Estimated Completion: April 25, 2024

Planned Effort: 10 hours

KANBAN :



BURNDOWN CHART :



This image represents a line graph that shows the number of events planned and actualized over a four-week period, likely March. The blue line represents the number of events that were planned, and the orange line represents the number of events that were actually carried out.

The x-axis shows the dates, with four evenly spaced tick marks labeled from 11-Mar to 25-Mar, and then one labeled 1Aprl. The y-axis shows the number of events, with tick marks from 0 to 20 which represents the number of tasks.

Design Methods and Implementation for Grocery Shopping Website:

User Interface (UI) and User Experience (UX) Design:

User Research: Conduct user research to understand target audience needs and preferences. This can involve surveys, interviews, or usability testing.

Information Architecture: Define the website's structure and organization to make navigation and information retrieval easy. Utilize sitemaps and user flows to visualize navigation paths.

Wireframing and Prototyping: Create low-fidelity wireframes to establish the basic layout and functionality. Then, progress to high-fidelity prototypes for user testing and feedback refinement.

Visual Design: Design the website's visual elements like color scheme, typography, and imagery, ensuring a clean, user-friendly, and aesthetically pleasing interface.

Design/Implementation:

Hardware Requirements

- I3 Processor with 4GB RAM

Software Requirements

- HTML
- CSS
- JS
- PHP
- SQL
- VSCode
- Apache
- PhpMyAdmin

TESTING

Manual Testing

We have performed various Test Cases, exploring software functionalities, and identifying defects or issues with the application. After Initial testing we have performed regression testing to ensure that the software's functionalities are working as expected. We have performed different testing types and they are Unit Testing, Integration Testing, System Testing and User Acceptance testing. Below are the test cases with description, Expected Result and the status of the test cases.

Test Case	Description	Expected Result	Status
TID01	Verify that a user can create an account with all mandatory fields correctly filled.	The system should successfully create the user account and display a confirmation message.	PASS
TID02	Check if user can login using valid credentials after creating an account	The system should authenticate the user and redirect them to the dashboard or homepage.	PASS
TID03	Test adding items to the shopping cart	Items should be added to the card with accurate quantity and pricing information	PASS
TID04	Verify users can search for products using the search bar	Relevant products matching the search query should be displayed in the search results	PASS
TID05	Test the checkout process by adding items to the cart and completing the purchase	The checkout process should proceed smoothly without errors, and the user should receive an order confirmation	PASS
TID06	Test the "Add to favorites" functionality for users	Users should be able to add items to their favorites list and view them later	PASS
TID07	Check the functionality of product categories	Users should be able to browse products under different categories.	PASS

TID08	Test the navigation links in header and footer of the website	Users should be able to navigate to different sections of the website using the header and footer links	PASS
TID09	Check the functionality of shopping cart page and checkout page	The shopping cart page should display all items added by the user, along with the options to update quantities, remove items and proceed to checkout.	PASS
TID010	Verify the functionality of the “My Orders”, where users can view their past orders.	Order History section should display a list of previous orders with order details such as date, items, total amount and order status.	PASS
TID011	Verify the behavior when adding a product to the cart with the quantity of zero or negative	The system should not allow adding a product with zero or negative quantity	PASS
TID012	Verify that admin can add new products to the application	Newly added products should appear in the product catalog and available for the user	PASS
TID013	Verify that a user can update their profile information(eg: Name, Email, Address)	Changes should be reflected in the user’s profile without errors	PASS
TID014	Verify the behavior of the “Logout” option in the user account menu	Clicking on “Logout” should successfully log the user out of their account and redirect them to the login page	PASS
TID015	Check if admin can view and manage user orders and statuses	Admin should have access to order details, be able to change order statuses and manage orders	PASS
TID016	Test the functionality of updating product details	Changes made by the admin should reflect accurately on the product pages	PASS
TID017	Check if an admin can view and the	Once the changes are made on the	PASS

	product details in the application after updating them	backend, they should reflect on the application page	
TID018	Test the responsiveness of the web application on different devices	The application should display properly on different screen sizes	PASS

Automated Testing

We have performed an Automated Testing process using Selenium, which allowed us to simulate the user actions on the web, Such as logging in, navigating, adding items to the cart and checking out. It helped in automating repetitive test scenarios, ensuring consistent testing across different runs, and identifying issues or bugs in the application's functionality. Below is the log representation of automated test scenario performed using Selenium:

Running 'Add Products to cart' 16:54:11

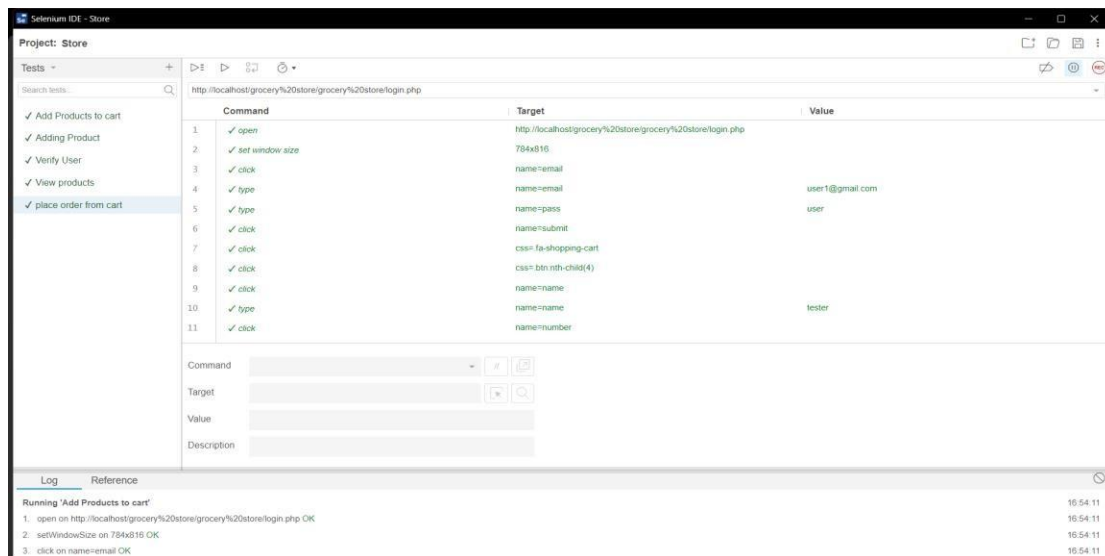
1. open on `http://localhost/grocery%20store/grocery%20store/login.php` OK16:54:11
2. `setWindowSize` on 784x816 OK16:54:11
3. click on `name=email` OK16:54:11
4. type on `name=email` with value `user1@gmail.com` OK16:54:12
5. type on `name=pass` with value `user` OK16:54:12
6. click on `name=submit` OK16:54:12
7. click on `css=.box:nth-child(3) > .btn:nth-child(11)` OK16:54:12
8. click on `css=.box:nth-child(4) > .qty` OK16:54:13
9. type on `css=.box:nth-child(4) > .qty` with value `2` OK16:54:13
10. click on `css=.box:nth-child(4) > .btn:nth-child(11)` OK16:54:13
11. click on `css=.fa-shopping-cart` OK16:54:13
12. pause on 3000 OK16:54:13
13. close OK16:54:16

'Add Products to cart' completed successfully16:54:16

Running 'Adding Product'16:54:23

1. open on `http://localhost/grocery%20store/grocery%20store/login.php` **OK**16:54:24
2. `setWindowSize` on 784x816 **OK**16:54:24
3. click on `name=email` **OK**16:54:24
4. type on `name=email` with value `admin1@gmail.com` **OK**16:54:24
5. type on `name=pass` with value `admin` **OK**16:54:24
6. click on `name=submit` **OK**16:54:24
7. click on `css=.box:nth-child(4) > .btn` **OK**16:54:24
8. click on `name=name` **OK**16:54:25
9. type on `name=name` with value `apple` **OK**16:54:25
10. click on `name=price` **OK**16:54:25
11. type on `name=price` with value `2` **OK**16:54:25
12. click on `name=category` **OK**16:54:25
13. select on `name=category` with value `label=fruits` **OK**16:54:26
14. click on `name=details` **OK**16:54:26
15. type on `name=details` with value `indian apple` **OK**16:54:26
16. click on `name=add_product` **OK**16:54:26
17. click on `name=add_product` **OK**16:54:26
18. click on `id=user-btn` **OK**16:54:27
19. click on `css=.delete-btn:nth-child(4)` **OK**16:54:27
20. close **OK**16:54:27

'Adding Product' completed successfully16:54:27



MAINTENANCE

Here are the four types of maintenance that is applicable to the E-Commerce Grocery Store project:

1. Corrective Maintenance: This type of maintenance involves fixing issues or bugs that arise after the system has been deployed. In this grocery shopping website, corrective maintenance we identified and resolved errors, malfunctions, or unexpected behavior in the system. For example, if users encounter issues during the checkout process, corrective maintenance would be performed to address and rectify these issues promptly.

2. Adaptive Maintenance: Adaptive maintenance focuses on making modifications to the system to ensure its compatibility with changes in the external environment, such as updates to operating systems, web browsers, or third-party integrations. For the grocery shopping website, we updated the platform to work seamlessly with new versions of web browsers or payment gateways, ensuring that customers can continue to access and use the website without any disruptions.

3. Perfective Maintenance: Perfective maintenance involves making enhancements or improvements to the system to enhance its functionality, performance, or user experience based on feedback from users or

stakeholders. For this grocery shopping website, we added new features or refining existing ones to better meet the needs and preferences of customers. For example, integrating a recommendation engine to suggest relevant products based on user preferences could be considered perfective maintenance.

4. Preventive Maintenance: Preventive maintenance aims to proactively identify and address potential issues or risks before they escalate into larger problems. This could involve regular monitoring, performance tuning, and security updates to ensure the stability, security, and scalability of the system. In the case of the grocery shopping website, we did regular security audits to identify and patch vulnerabilities, as well as performance optimization to ensure optimal loading times and responsiveness for users.

Roles and Responsibilities :

Lalitha (Front-end Developer):

- Developed user interface components using HTML, CSS, and JavaScript.
- Integrated front-end components with back-end services.
- Implemented user authentication and authorization mechanisms.
- Developed features such as product catalog browsing, search functionality, and shopping cart management.
- Conducted code reviews and optimizations to improve performance and maintainability of the front-end codebase.

Sravani (Back-end Developer):

- Implemented RESTful APIs for communication between the front-end and backend components of the application.
- Developed features such as secure payment processing, order tracking, and user authentication using back-end frameworks.
- Ensured data security by implementing encryption, access control, and other security measures.
- Optimized server-side code for performance and scalability to handle increasing traffic and data volumes.
- Written comprehensive unit tests and conduct integration testing to ensure the reliability and correctness of server-side code.

Agan Peter (Full-stack Developer):

- Collaborated with front-end developers to integrate front-end components with back-end APIs and ensure smooth functionality.

- Assisted in developing user interface components using HTML, CSS, and JavaScript.
- Assisted in implementing RESTful APIs for communication between the front-end and backend components.
- Collaborated with the back-end developer to develop features such as secure payment processing, order tracking, and user authentication.
- Assisted in writing comprehensive unit tests and conducting integration testing to ensure the reliability and correctness of both front-end and back-end code.