DAY 2

PLANNING THE TECHNICAL FOUNDATION

E-COMMERCE CLOTHING WEBSITE

The technical requirements to build an e-commerce clothes website can be broadly divided into three categories: system requirements, non-functional requirements, and functional needs. Here is a thorough explanation:

1. Functional Requirements

These are the features and functionalities the website should provide:

User Management

- User registration, login, and profile management.
- Password reset and secure account recovery.
- Role-based access (admin, customer, vendor, etc.).

Product Catalog

- Product listing with categories, subcategories, and filters (e.g., size, color, price range, brand).
- Detailed product pages with images, descriptions, sizes, reviews, and availability.

Search and Navigation

- Advanced search functionality with predictive suggestions.
- Navigation menus and breadcrumbs for easy browsing.
- Sorting and filtering options.

Shopping Cart

- Add, update, or remove items from the cart.
- Persistent cart across sessions for logged-in users.

Checkout and Payment

- Secure checkout process with guest and registered user options.
- Payment gateway integration (e.g., credit/debit cards).
- Support for multiple currencies and tax calculations.
- Provide shipping address and select a delivery.

Order Management

- Order tracking and history for users.
- Email/SMS notifications for order confirmation, shipping, and delivery.

Inventory Management

Real-time inventory tracking.

Automatic stock updates when orders are placed.

Shipping and Delivery

- Integration with shipping carriers.
- Delivery options (standard, express, etc.) with cost calculation.

Customer Support

- Chatbot or live chat for assistance.
- FAQs and help center.
- Return/refund request functionality.

Admin Panel

- Dashboard for managing users, products, orders, and reports.
- Analytics for sales, customer behavior, and website performance.

2. Non-Functional Requirements

These define the quality attributes and performance standards:

Performance

- Fast page load times (<3 seconds).
- Scalable to handle high traffic during sales or events.

Security

- Secure user data with SSL/TLS encryption.
- Role-based access control and secure admin logins.

Scalability

- Scalable backend architecture to handle growth (users, products, and transactions).

Usability

- Intuitive and mobile-friendly UI/UX.
- Accessible to all users

Reliability

- 99.9% uptime with a robust hosting solution.
- Backup and disaster recovery mechanisms.

SEO and Marketing

- SEO-friendly URLs, metadata, and schema markup.
- Social media integration for easy sharing.
- Support for discounts, coupons, and promotional campaigns.
-

3. System Requirements

These involve the technology stack and infrastructure:

Frontend

- Frameworks/Libraries: React.js
- Responsive design using CSS frameworks like Bootstrap or Tailwind CSS.

Backend

- Programming Language: Python , JavaScript (Node.js)
- API development for communication between frontend and backend.

Payment Gateway

- Integration with providers like Stripe, Easypaisa , jazzcash, (HBL,UBL payment digital gateway)

Third-Party Integrations

- Analytics: Google Analytics, meta (facebook) ads manager.
- Email service: Mailchimp or SendGrid.
- Shipping services: FedEx, TCS, Leapord , DHL etc.

Development Tools

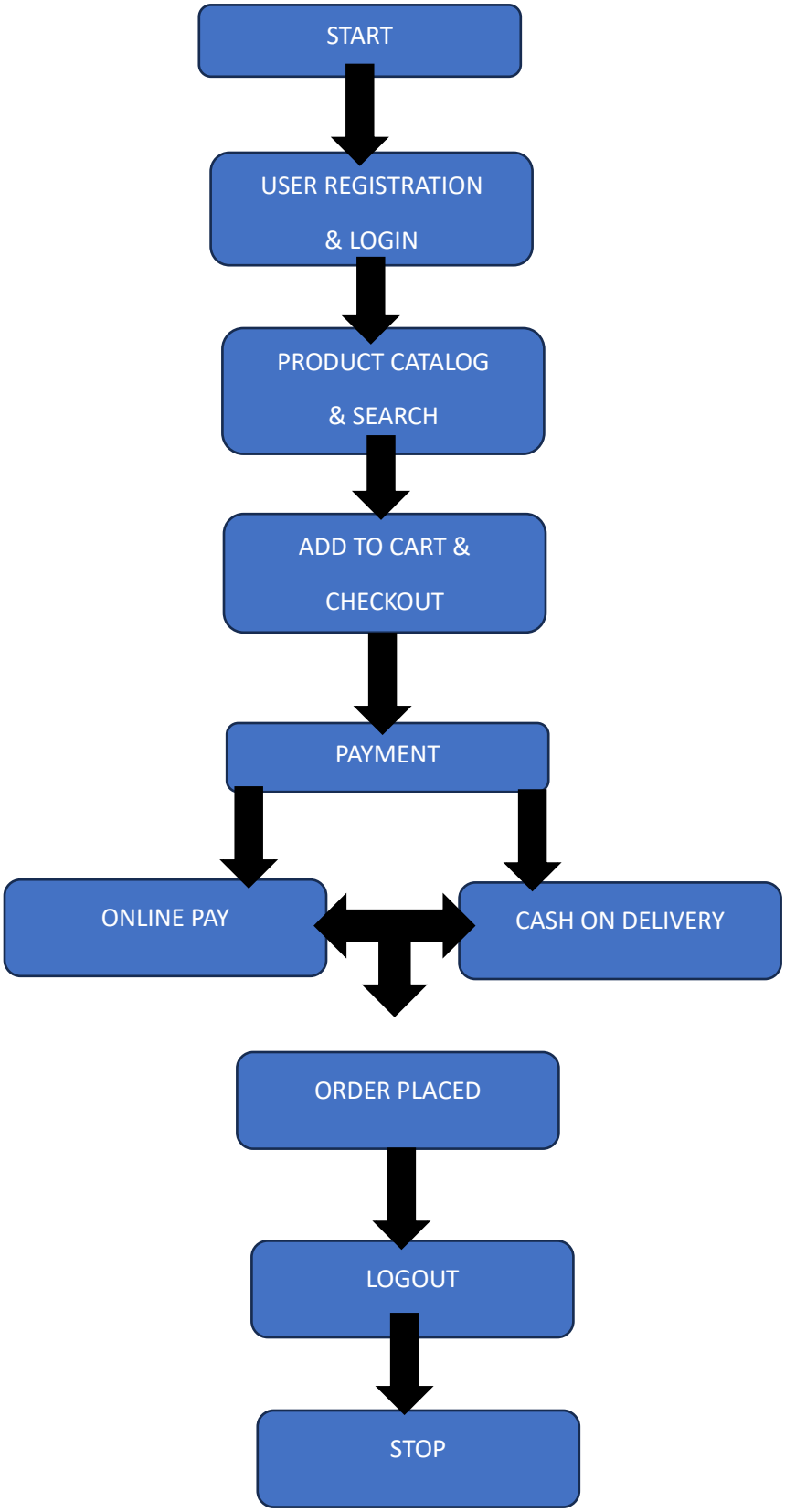
- Version control: Git (GitHub)
- vercel for smooth deployments.

4. Additional Features (Optional)

- Personalized recommendations
- virtual try-ons.
- Loyalty programs and reward points.
- Multi-language and multi-currency support for global reach.

By outlining these technical requirements, you can ensure the development process is smooth, and the final product meets user expectations.

FLOW CHART


START
USER REGISTRATION & LOGIN
PRODUCT CATALOG & SEARCH
ADD TO CART & CHECKOUT
PAYMENT
ONLINE PAY
CASH ON DELIVERY
ORDER PLACED
LOGOUT
STOP

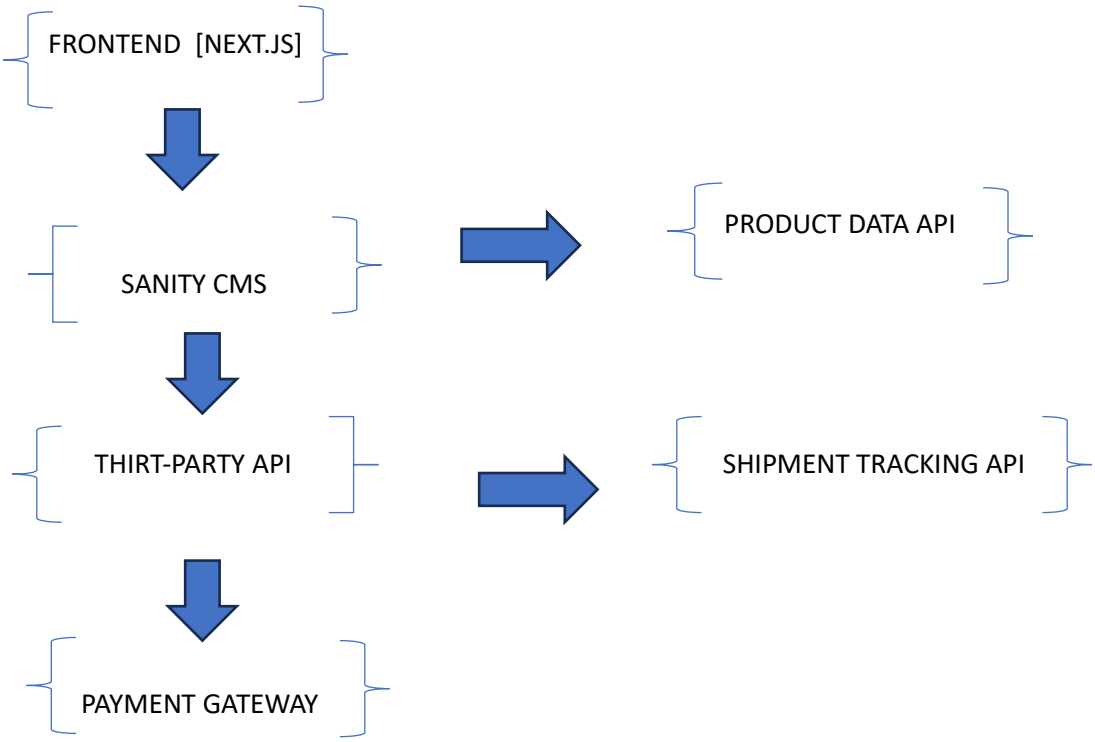
FRONTEND REQUIRMENTS:

USER RECIVES A NOTIFOCTION AND CAN LEAVE A REVIEW .USER-FRIENDLY INTERFACE FOR BROWSING PRODUCT.

RESPONSIVE DESIGN FOR MOBILE AND DESKTOP USERS.ESSENTIAL PAGES: PRODUCT LISTING, PRODUCT DETAILS, CART, CHEACKOUT, ORDER CONFIRMATION .

SYSTEM ARCHITECTURE OVERVIEW

DIAGRAM:



FRONTEND	BACKEND	APIS	TOOLS
• **NEXT.JS** FORBUILDING MODERN, DYNAMICS AND SERVER. RENDERD UIS • **TAILWIND** FOR RESPONSIVE AND VISUALLY APPESLING DESINGS • **SHADCN UI** FOR PRE-DESIGNED CUSTOMIZABLE COMPONENTS.	• **SANITY CMS** TO MANAGE AND STRUCTURE EFFECTIVELY • **CLERK** FOR SEEMLESS USERS AUTHENTICATION AND MANAGEMENT	• **SHIPENGINE** SIMPLIFIES SHIPMENT TRACKING AND DELIVERY • **STRIPE** HANDLESS SECURE AND EFFICIENT ONLINE PAYMENTS	• **GITHUB** VERSION CONTROL AND COLLABORATIVE DEVELOPMENT • **POSTMAN** FOR TESTING AND DOCUMENTING APIS EFFECTIVELY. • **VERCEL** FAST AND RELABLE DEPLOYMENT FOR NEXT.JS APPLICATIONS

API ENDPOINTS:

Data for the table representing API endpoints and their functions

```
api_endpoints_data = {
    "API Endpoint": [
        "POST /auth/login",
        "GET /user/profile",
        "POST /orders/create",
        "GET /products/list",
```

```
        "PUT /user/update",
        "DELETE /cart/remove",
        "POST /payment/checkout",
        "GET /notifications/unread"
    ],
    "Functionality": [
        "Authenticate users and provide an access token.",
        "Retrieve the profile information of the logged-in user.",
        "Create a new order in the system.",
        "Fetch a list of all available products.",
        "Update user profile details.",
        "Remove a specific item from the shopping cart.",
        "Process payments for the user's order.",
        "Get all unread notifications for the user."
    ]
}

# Create a DataFrame for better visualization
api_endpoints_df = pd.DataFrame(api_endpoints_data)
```

```
# Plot the table
fig, ax = plt.subplots(figsize=(10, 4))
ax.axis("off")  # Turn off the axis
table = ax.table(
    cellText=api_endpoints_df.values,
    colLabels=api_endpoints_df.columns,
    cellLoc="left",
    loc="center",
    colColours=["lightblue"] * len(api_endpoints_df.columns),
)

# Format the table
table.auto_set_font_size(False)
table.set_fontsize(10)
table.auto_set_column_width(col=list(range(len(api_endpoints_df.columns))))

# Display the table
plt.show()
```