SOUL OF BANGLADESH

OUR TEAM

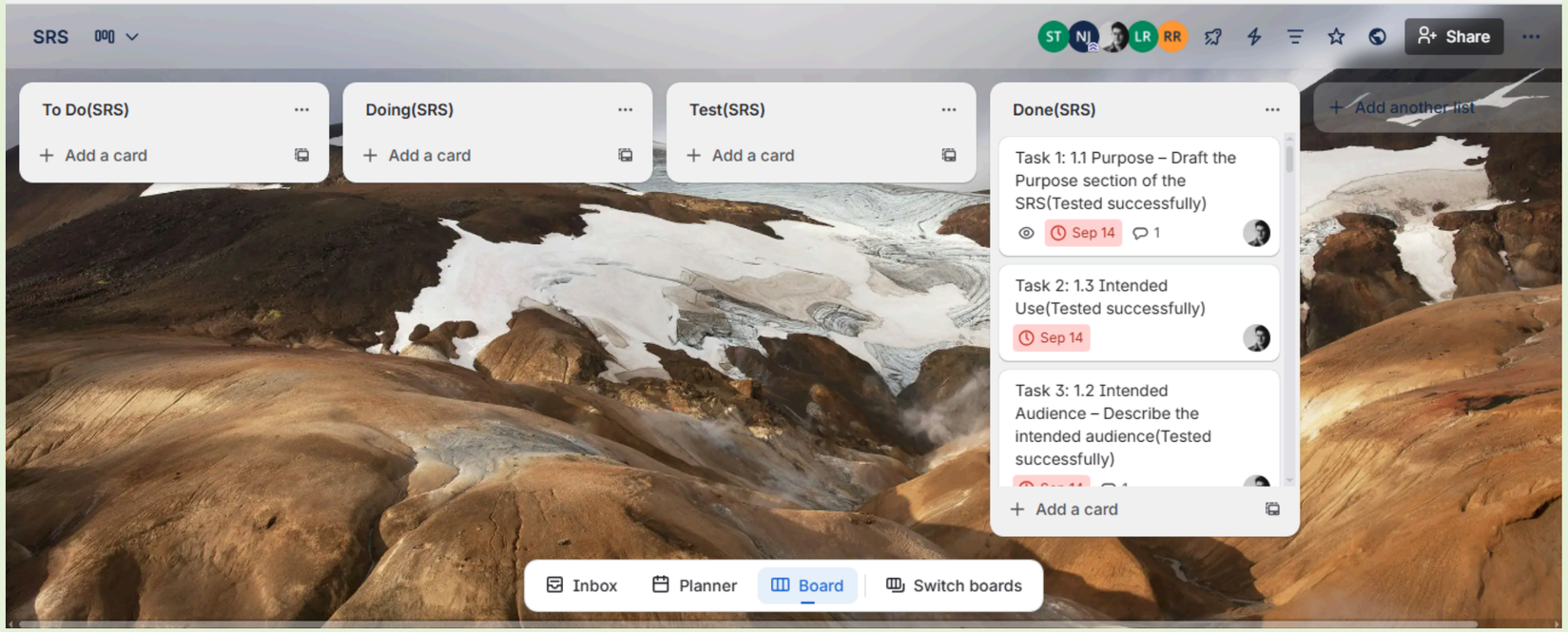
- Labonno Rahan Oishy (2254901107)
- Nasrat Jahan (2254901061)
- Rabeya Islam Rima (2254901055)
- Ameer Sohail(2254901011)
- Sunehra Tabassum (2254901047)

REQUIREMENTS

FEATURES

Labonno Rahman Oishy	Admin Panel, Sundorbon
Nasrat Jahan	Education, Culture
Sunehra Tabassum	Transportation, Hill and Mountain
Ameer Sohail	Sports, Haritage
Rabeya Islam Rima	River and War History

TRELLO ACTIVITY ON SRS



SRS CONTENT IN WIKI

Contents

CODING STANDARDS

CODING STANDARDS

1. HTML STANDARDS:
   - SEMANTIC STRUCTURE: USE SEMANTIC TAGS LIKE <HEADER>, <SECTION>, AND <FOOTER> FOR BETTER READABILITY AND ACCESSIBILITY.
   - CLASS NAMING: USE KEBAB-CASE FOR CSS CLASSES (E.G., MAIN-HEADER).
   - ID NAMING: USE CAMELCASE FOR UNIQUE ELEMENT IDENTIFIERS, TYPICALLY FOR JAVASCRIPT HOOKS (E.G., USERPROFILE).
   - ATTRIBUTES: INCLUDE ALT FOR IMAGES AND DESCRIPTIVE NAMES FOR FORM INPUTS.
   - ACCESSIBILITY: USE ARIA ATTRIBUTES LIKE ROLE AND ARIA-LABEL WHERE NECESSARY.
   - FORMS: USE APPROPRIATE INPUT TYPES (E.G., EMAIL, PASSWORD), REQUIRED AND PATTERN FOR CLIENT-SIDE VALIDATION.
   - META TAGS: INCLUDE PROPER CHARSET AND VIEWPORT IN <HEAD> FOR RESPONSIVE DESIGN.

CODING STANDARDS

2. CSS STANDARDS:
   - CLASSES: USE KEBAB-CASE (E.G., INFO-BOX, READ-MORE-BTN).
   - IDS: USE CAMELCASE FOR UNIQUE ELEMENTS (E.G., MAINNAV).
   - INDENTATION: USE 2 SPACES FOR INDENTATION.
   - INLINE STYLES: AVOID INLINE STYLES; USE CLASSES FOR STYLING.
   - CSS VARIABLES: USE FOR COLORS, FONTS, AND SPACING TO MAKE THE CODE MORE MAINTAINABLE.
   - RESPONSIVE DESIGN: USE MOBILE-FIRST MEDIA QUERIES FOR RESPONSIVE STYLING.
   - COMMENTING: ADD COMMENTS FOR CLARITY AND MAINTAINABILITY.

3. JAVASCRIPT STANDARDS:
   - VARIABLES AND FUNCTIONS: USE CAMELCASE (E.G., LOGINUSER(), ADDTOCART()).
   - CONSTANTS: USE UPPERCASE (E.G., MAX_COUNT).
   - ES6+ FEATURES: USE LET, CONST, ARROW FUNCTIONS, TEMPLATE LITERALS.
   - AVOID GLOBAL SCOPE POLLUTION: ENCAPSULATE LOGIC IN WINDOW.ONLOAD OR IIFE (IMMEDIATELY INVOKED FUNCTION EXPRESSIONS).
   - EVENT DELEGATION: PREFER EVENT DELEGATION FOR HANDLING MULTIPLE ELEMENTS.
   - ERROR HANDLING: USE TRY-CATCH BLOCKS FOR ERROR HANDLING.
   - MODULAR JAVASCRIPT: USE SEPARATE FILES FOR DIFFERENT COMPONENTS

CODING STANDARDS

4. PHP STANDARDS:

- VARIABLES: USE SNAKE_CASE (E.G., $USER_ID, $TOTAL_PRODUCTS).
- FUNCTIONS: USE CAMELCASE (E.G., LOGINUSER(), ADDTOCART()).
- CLASSES: USE PASCALCASE (E.G., USERCONTROLLER).
- FILES: USE LOWERCASE WITH UNDERSCORES (E.G., CONNECT.PHP, USERHEADER.PHP).
- STRICT TYPING: ENABLE STRICT TYPES WITH DECLARE(STRICT_TYPES=1);.
- INCLUDES: USE REQUIRE_ONCE() FOR CRITICAL INCLUDES.
- INPUT VALIDATION: USE FILTER_INPUT() FOR INPUT VALIDATION AND SANITIZATION.
- ERROR REPORTING: DIFFERENT ERROR REPORTING SETTINGS FOR DEVELOPMENT AND PRODUCTION ENVIRONMENTS.

DATABASE STANDARDS:

- TABLE AND COLUMN NAMES: USE SNAKE_CASE FOR NAMING (E.G., USER_PROFILE).
- PRIMARY KEYS: USE AUTO-INCREMENT INTEGERS FOR PRIMARY KEYS.
- SQL KEYWORDS: USE UPPERCASE FOR SQL KEYWORDS (E.G., SELECT, INSERT), AND LOWERCASE FOR TABLE AND COLUMN NAMES.
- FOREIGN KEYS: USE FOREIGN KEYS TO DEFINE RELATIONSHIPS BETWEEN TABLES.
- INDEXES: INDEX FREQUENTLY USED COLUMNS FOR PERFORMANCE OPTIMIZATION.
- CHARACTER SET: USE UTF8MB4 FOR FULL UNICODE SUPPORT.
- AUTO TIMESTAMPS: INCLUDE CREATED_AT AND UPDATED_AT TIMESTAMPS IN TABLES.
- PREPARED STATEMENTS: ALWAYS USE PREPARED STATEMENTS FOR DATABASE QUERIES.

CODING STANDARDS

6. SECURITY PRACTICES:
   - INPUT VALIDATION: USE HTMLSPECIALCHARS(), STRIP_TAGS(), AND FILTER_INPUT() TO SANITIZE INPUTS.
   - SQL INJECTION PREVENTION: USE PREPARED STATEMENTS AND PARAMETERIZED QUERIES.
   - XSS PREVENTION: ESCAPE DYNAMIC CONTENT USING HTMLSPECIALCHARS().
   - CSRF PROTECTION: INCLUDE AND VERIFY CSRF TOKENS IN CRITICAL FORMS.
   - PASSWORD SECURITY: HASH PASSWORDS USING PASSWORD_HASH() (BCRYPT) AND VERIFY WITH PASSWORD_VERIFY().
   - SESSION MANAGEMENT: USE SESSION_START(), REGENERATE SESSION IDS ON LOGIN, AND DESTROY SESSIONS ON LOGOUT.
   - ACCESS CONTROL: IMPLEMENT ROLE-BASED ACCESS CONTROLS FOR SENSITIVE PAGES (E.G., ADMIN DASHBOARDS).
   - HTTPS: ALWAYS USE HTTPS FOR ENCRYPTING DATA.
   - ERROR HANDLING: AVOID EXPOSING DETAILED ERROR MESSAGES TO THE USER; LOG THEM SECURELY.

CODING STANDARDS

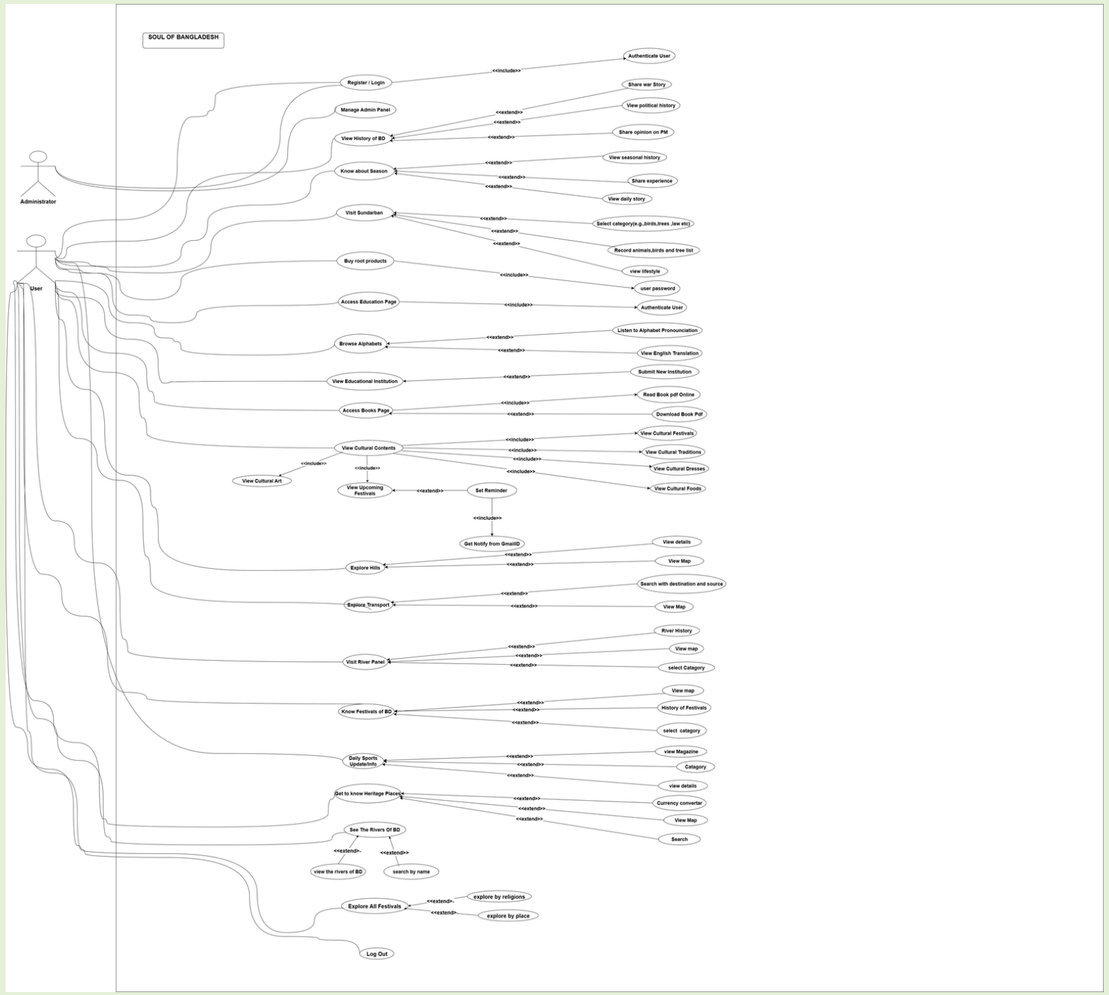
7. PROJECT STRUCTURE AND VERSION CONTROL:

- FOLDER STRUCTURE:
  - /ASSETS FOR STATIC RESOURCES (E.G., CSS, JS, IMAGES).
  - /COMPONENTS FOR REUSABLE COMPONENTS (E.G., HEADER.PHP, FOOTER.PHP).
  - /PAGES FOR DIFFERENT PAGES (E.G., HOME.PHP, CART.PHP).
  - /CONFIG FOR CONFIGURATION FILES (E.G., CONFIG.PHP).
  - /VENDOR FOR THIRD-PARTY LIBRARIES.
  - INDEX.PHP FOR THE MAIN ENTRY FILE.
- VERSION CONTROL:
  - GIT BRANCHING: USE MAIN FOR PRODUCTION, DEV FOR DEVELOPMENT, AND FEATURE-SPECIFIC BRANCHES (E.G., FEATURE/XYZ).
  - USE MEANINGFUL COMMIT MESSAGES LIKE FEAT, FIX, DOCS, REFACTOR, AND CHORE.
  - FOLLOW THE GITHUB WORKFLOW FOR PULL REQUESTS AND CODE REVIEWS.

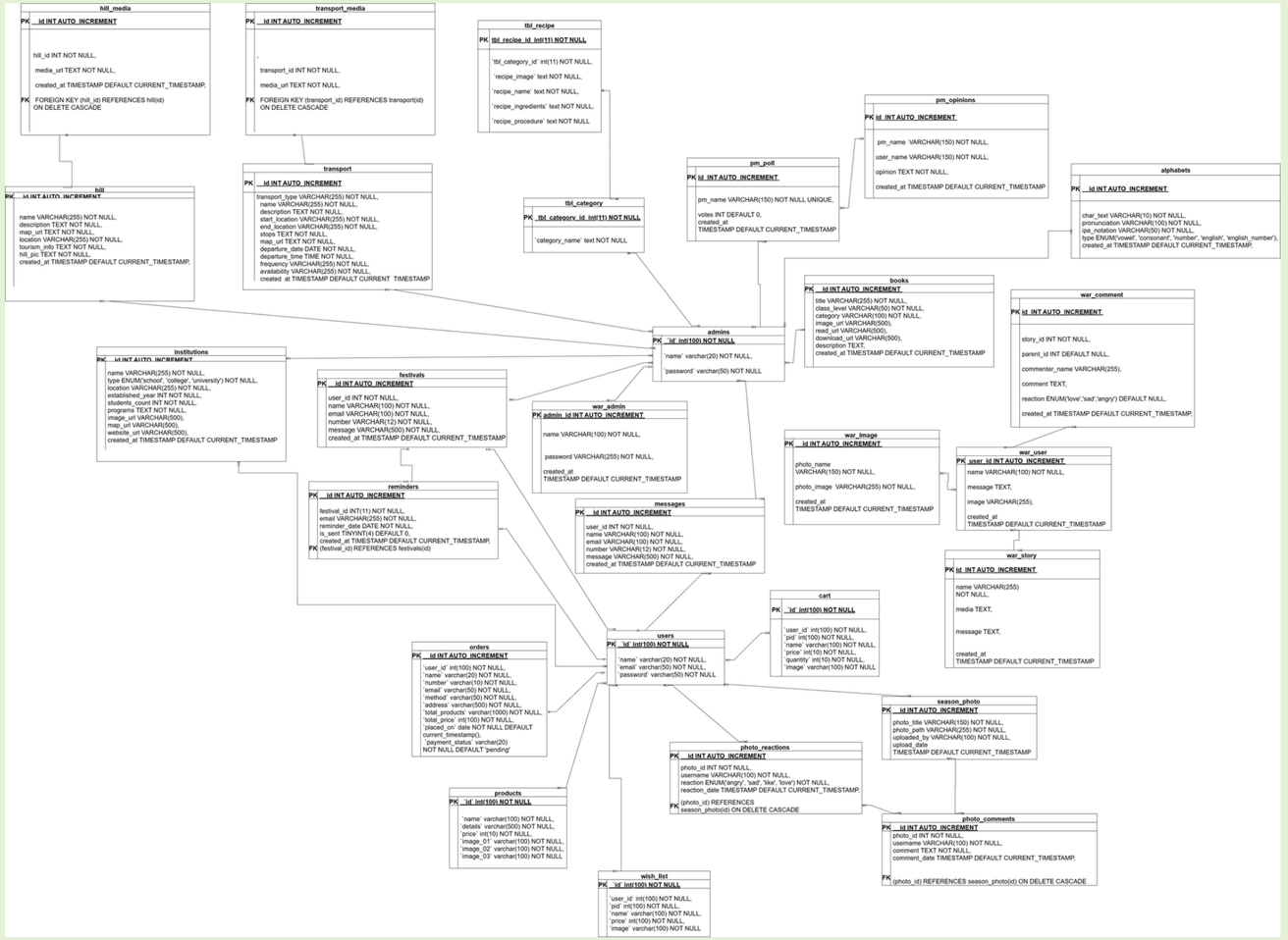
DOCUMENTATION

DESIGN

USE CASE DIAGRAM



SCHEMA DIAGRAM

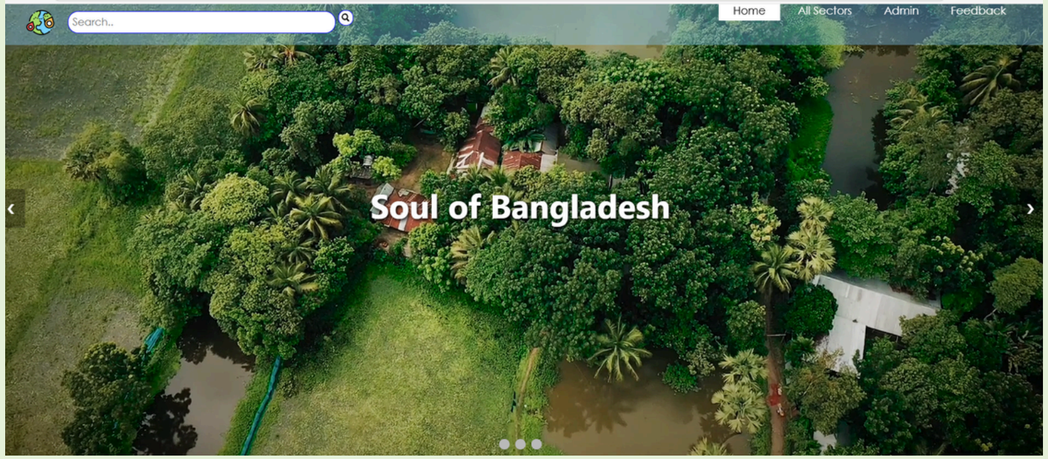


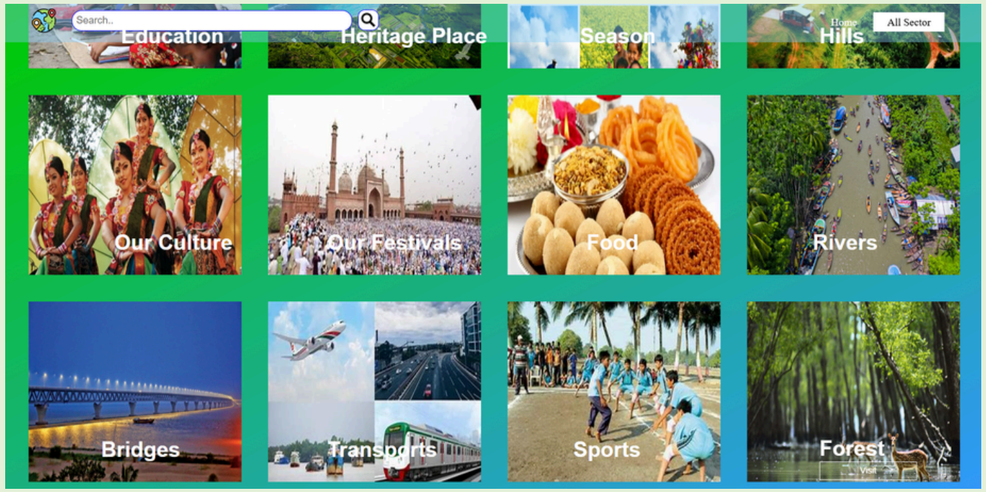
NIELSEN'S HEURISTICS

Nielsen's heuristics are general principles, meaning that they do not determine specific usability rules. Instead, the heuristics are general rules of thumb you can follow to help create more accessible, user-friendly, and intuitive digital products.

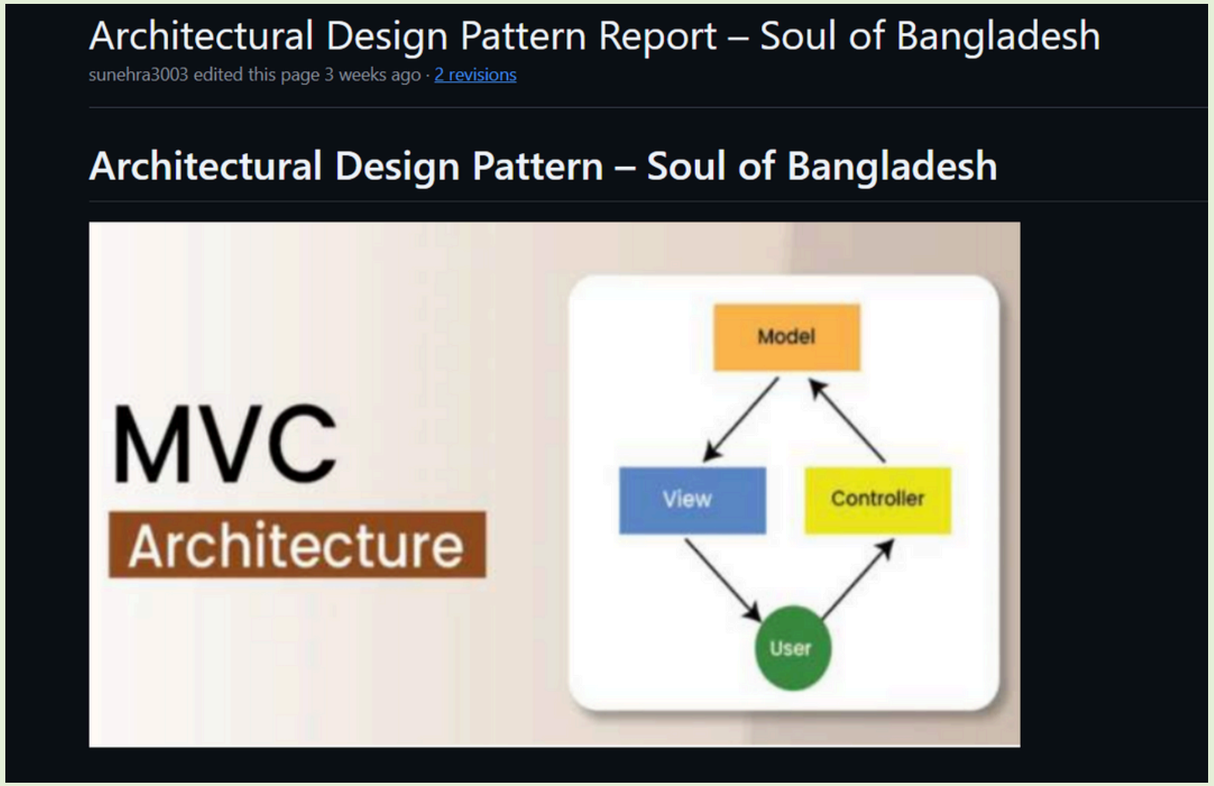
1. Visibility of system status;
2. Match between system and the real world;
3. User control and freedom;
4. Consistency and standards;
5. Error prevention;
6. Recognition rather than recall;
7. Flexibility and efficiency of use;
8. Aesthetic and minimalist design;
9. Help users recognize, diagnose, and recover from errors;
10. Help and documentation.

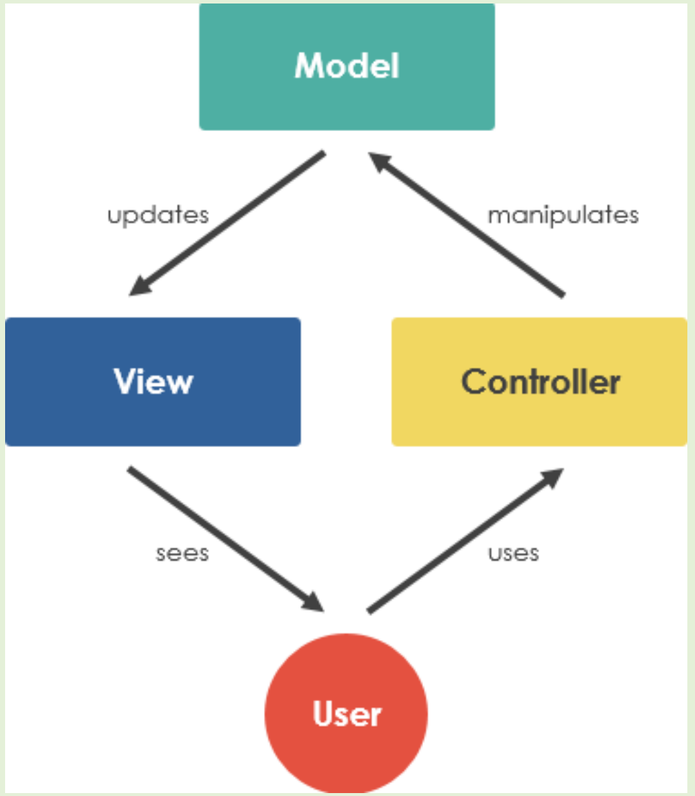
UI OF THE PROJECT





ARCHITECTURE PATTURN MVC





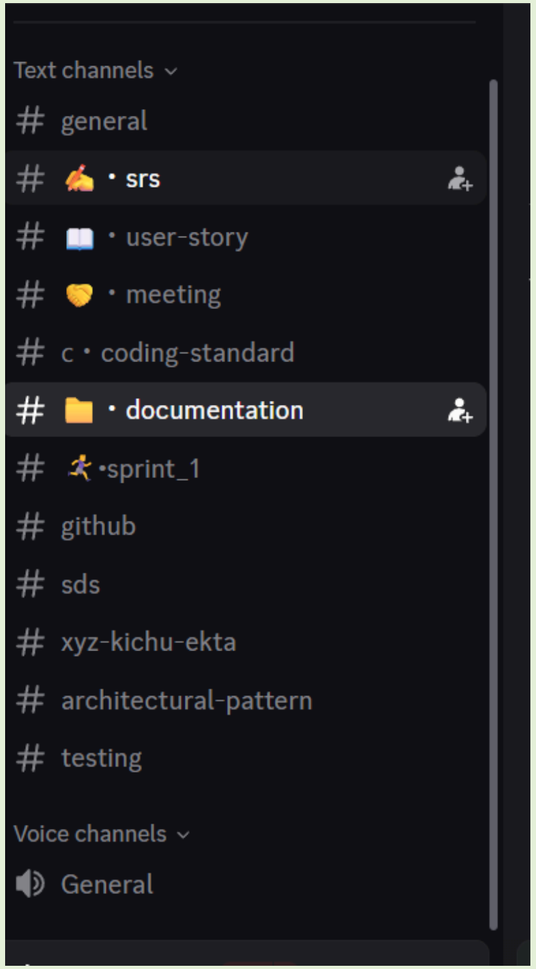
ARCHITECTURE PATTURN MVC

Model	Handles data logic, database queries, and business rules
View	Manages UI/UX and displays content to users (HTML, CSS, Tailwind, JS)
Controller	Works as the bridge between Model and View, handles user requests and system responses

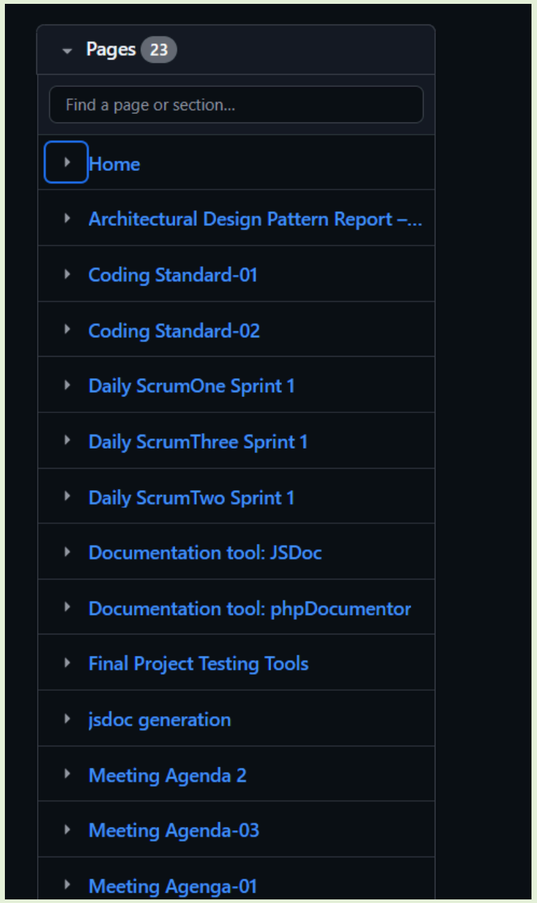
ARCHITECTURE PATTURN MVC

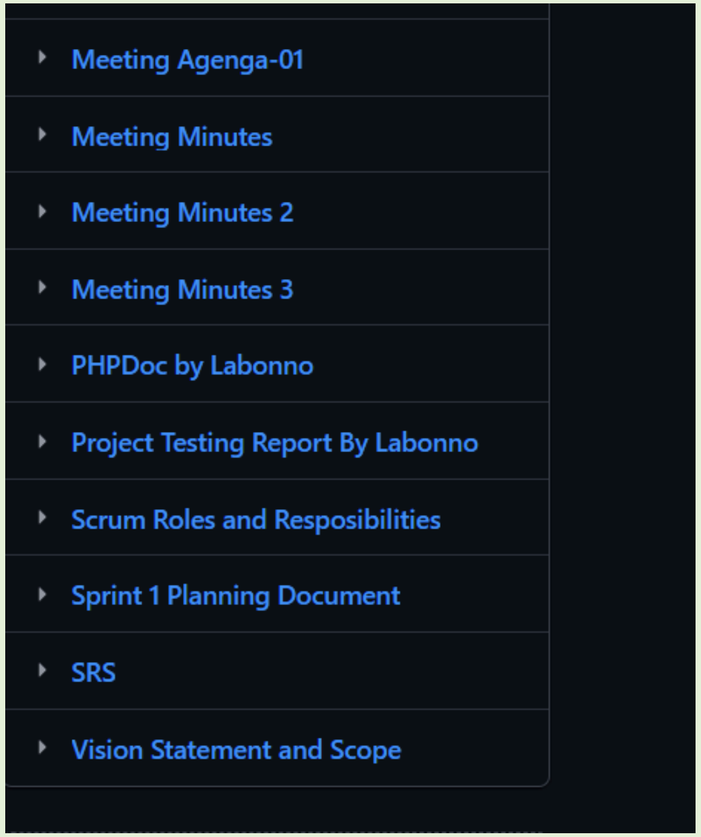
1. **CLEAN SEPARATION OF LOGIC:** NO MIXING OF UI CODE WITH DATABASE OR PROCESSING LOGIC. MAKES CODE NEAT AND PRO-LEVEL.
2. **SCALABILITY:** NEW FEATURES LIKE LANGUAGE LEARNING, SEASONAL MODULES, VOTING, OR CHATBOT INTEGRATION CAN BE ADDED WITHOUT BREAKING EXISTING CODE.
3. **EASY MAINTENANCE:** BUGS ARE EASIER TO TRACK BECAUSE EACH PART (UI, LOGIC, DB) IS ISOLATED.
4. **REUSABILITY:** COMPONENTS LIKE DATABASE QUERIES, CONTROLLERS, AND UI TEMPLATES CAN BE REUSED ACROSS MULTIPLE SECTIONS OF THE WEBSITE.
5. **FASTER COLLABORATION:** SINCE CODE IS MODULAR, MULTIPLE DEVELOPERS CAN WORK SIMULTANEOUSLY ON DIFFERENT PARTS—UI, LOGIC, AND DATABASE—WITHOUT CONFLICTS.
6. **BETTER PERFORMANCE & SECURITY:** USER INPUTS ARE HANDLED IN CONTROLLERS, SANITIZED IN MODELS, AND ONLY SAFE OUTPUTS REACH THE VIEWS, REDUCING SECURITY RISKS LIKE SQL INJECTION AND LOGIC LOOPHOLES.

DISCORD CHANNELS FOR PROJECT

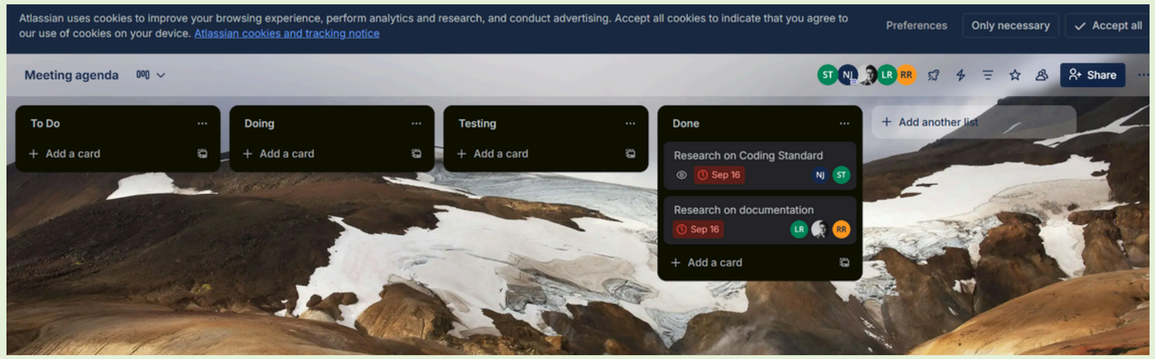


THE PAGES OF PROJECT IN GITHUB

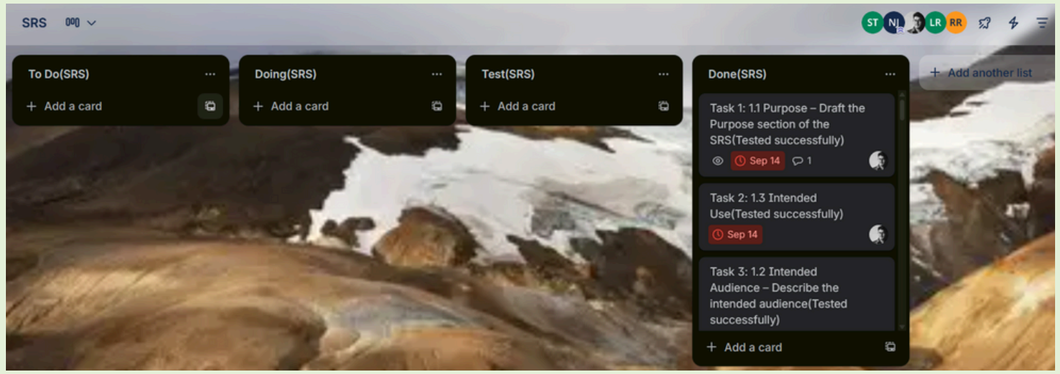




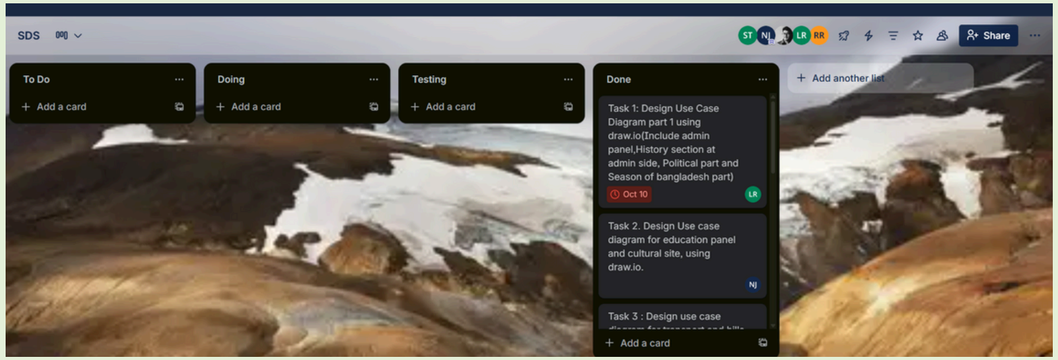
TRELLO TASK CARD FOR PROJECT



Meeting Agenda

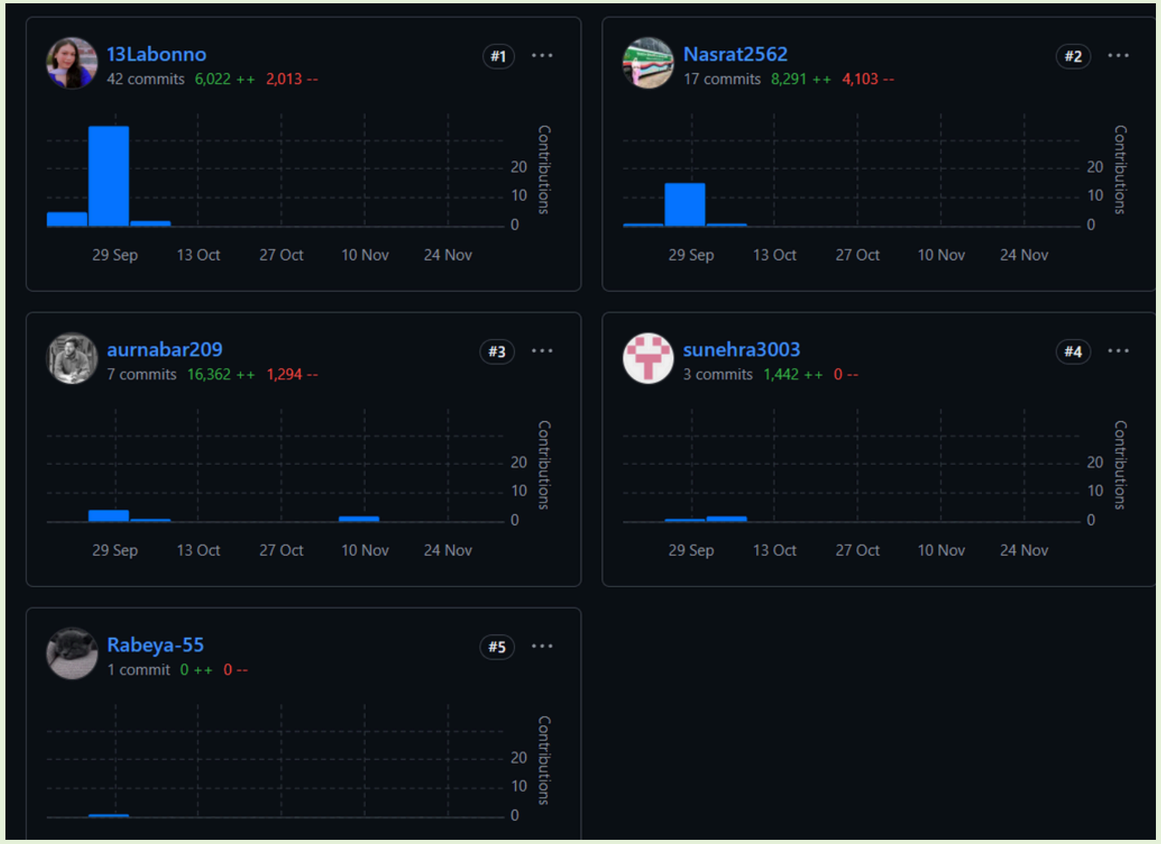


SRS



SDS

THE CONTRIBUTORS OF PROJECT



THANK YOU