MVP Implementation

Phase 3

1. Project Setup

Setting up the project correctly is critical before implementing any core functionality. A well-structured setup ensures scalability, maintainability, and smooth collaboration among team members.

- **Choosing the Technology Stack**:
  For user registration with validation, the tech stack could vary depending on project requirements. Commonly used stacks include:
  - **Frontend**: React, Angular, or Vue.js.
  - **Backend**: Node.js (Express), Django (Python), or Spring Boot (Java).
  - **Database**: MongoDB (NoSQL) or PostgreSQL/MySQL (SQL).
- **Environment Configuration**:
  - Install Node.js, npm/yarn, or Python depending on the stack.
  - Setup IDE/Editor (VS Code, IntelliJ, or PyCharm).
  - Install version control (Git) and connect with GitHub.
  - Use package managers to install dependencies.
- **Dependencies**:
  For validation and security, install:
  - **Backend**:
    - bcrypt or argon2 for password hashing.
    - joi, express-validator, or yup for input validation.
    - jsonwebtoken (JWT) for authentication.
  - **Frontend**:
    - Form libraries like Formik or React Hook Form.
    - Validation helpers like Yup.
- **Folder Structure** (Example for Node.js + React):

```
backend/
  controllers/
  models/
  routes/
  middlewares/
```

```
  tests/
frontend/
  src/components/
  src/pages/
  src/hooks/
  src/services/
```

2. Core Features Implementation

The **user registration system** is one of the most critical parts of any application. It must handle input validation, security, error handling, and persistence.

- **Frontend Registration Form**:
  - Input fields: Name, Email, Password, Confirm Password.
  - Validation rules:
    - Name: Not empty, minimum length.
    - Email: Must follow standard email format.
    - Password: Strong (min 8 characters, one uppercase, one number, one special character).
    - Confirm Password: Must match the password field.
  - User Experience (UX): Show inline validation errors, disable the submit button until the form is valid.
- **Backend API for Registration**:
  - **Endpoint**: POST /api/auth/register.
  - **Steps**:
    1. Validate request body (name, email, password).
    2. Check if the email already exists in the database.
    3. Hash the password using bcrypt before saving.
    4. Store user data in the database.
    5. Return a success response (201 Created) with basic user info (excluding password).
- **Error Handling**:
  - Invalid input → 400 Bad Request.
  - Duplicate email → 409 Conflict.
  - Unexpected server errors → 500 Internal Server Error.
- **Security Considerations**:
  - Store only hashed passwords.
  - Sanitize inputs to prevent SQL Injection / XSS.
  - Use HTTPS for secure data transfer.

3. Data Storage (Local State / Database)

Storing user registration data properly ensures data consistency and security.

- **Frontend (Local State Management)**:
  - Use React useState or Redux to manage form values and error messages.
  - Reset the form after successful registration.
  - Never store sensitive information like passwords in local storage or session storage.
- **Backend Database Storage**:
  - **Schema Example (MongoDB)**:
  - {
  - "id": "uuid",
  - "name": "string",
  - "email": "string (unique)",
  - "password": "hashed string",
  - "createdAt": "timestamp",
  - "updatedAt": "timestamp"
  - }
  - **Constraints**:
    - email must be unique.
    - password must be hashed (bcrypt with salt).
    - Indexing on email for faster lookup.
- **Data Protection**:
  - Follow GDPR and data protection laws (do not store unnecessary personal data).
  - Use environment variables to manage sensitive database credentials.

4. Testing Core Features

Testing ensures that the user registration process works as expected under all scenarios.

- **Frontend Testing**:
  - Unit Tests: Verify validation logic (e.g., password length, email format).
  - UI/UX Testing: Ensure error messages display correctly.
  - End-to-End Testing: Simulate the full flow of filling the form and submitting it.
- **Backend Testing**:
  - Unit Tests: Validate request data, password hashing, duplicate user checks.
  - Integration Tests: Test the full /register API endpoint with mock databases.
  - Security Tests: Ensure that passwords are hashed and API does not leak sensitive information.
- **Manual Testing Cases**:
  1. Register with invalid email → should fail.
  2. Register with weak password → should fail.

3. Register with duplicate email → should fail.
4. Register with valid details → should succeed.

5. Version Control (GitHub)

Version control ensures smooth collaboration and code management.

- **Branching Model**:
  - main → production-ready code.
  - dev → ongoing development.
  - feature/user-registration → work on user registration specifically.
- **Commit Practices**:
  - Write meaningful commit messages:
    - feat(auth): implement user registration API
    - fix(auth): resolve password validation bug
  - Commit frequently with small, logical changes.
- **Collaboration**:
  - Pull Requests (PRs) reviewed before merging.
  - Use GitHub Issues to track bugs and improvements.
  - GitHub Actions for continuous integration (running automated tests).

LINK : GitHub - Riyasantro/SMTEC-NM-IBM-USERS-REGISTRATION-WITH-VALIDATION

Conclusion

Implementing **User Registration with Validation** is a foundational step in building a secure and user-friendly application. By setting up the project environment properly, implementing strong validation both on the frontend and backend, securely storing data, and thoroughly testing all scenarios, we ensure the system is **robust, reliable, and secure**.

Version control using GitHub provides collaboration and ensures code quality. At the end of **Week 8**, the MVP will successfully allow new users to register with their details validated, securely stored, and ready for further enhancements such as **login, authentication (JWT), role-based access, and profile management**.