**Week 1: Project Proposal**

**Project Title: VERVE**

**Problem Statement**

In today’s digital world, people receive a lot of information from various news sources, which can be overwhelming. Our platform focuses on credibility and inclusivity by featuring verified reports from official publishers. While anyone can share their views, uncertified content is kept separate to prevent misinformation and protect free speech.  
Many find it inconvenient to visit newsstands or buy newspapers. Our platform eliminates these barriers by delivering news directly to users' devices, providing a modern, engaging experience that is easy to access.

**Objectives**

* Develop a web-based application that aggregates news from multiple reliable sources.
* Implement user-specific customization to tailor news feeds according to individual interests and reading habits.
* Provide real-time updates to keep users informed about the latest developments.
* Ensure a user-friendly interface that enhances the overall news reading experience.
* Ease the availability of breaking news to users through the platform.

**Target Users**

* Individuals seeking a personalized and efficient way to consume news.
* Creates a curated feed for everyone's preference in an aesthetic way.

**Technologies and Frameworks**

**Backend**

* Java v17 with Spring Boot for building robust and scalable RESTful APIs.
* Integration with weather APIs to fetch the weather forecast.
* Database management using PostgreSQL for efficient data storage and retrieval.

**Frontend**

* HTML, CSS, and JavaScript
* Vanilla JS
* HTTP session for session IDs

**Additional Tools**

* Git for version control and collaboration.

**Expected Design Patterns**

* **Model-View-Controller (MVC):** To separate concerns between the data (Model), user interface (View), and business logic (Controller), facilitating organized and maintainable code.
* **Observer Pattern:** To implement real-time updates, allowing the system to notify users of new articles or updates in their preferred categories.
* **Singleton Pattern:** To manage a single instance of core services, such as the news-fetching service, ensuring controlled access to resources.

**Initial High-Level System Overview**

**News App Structure**

**1. User Authentication**

* Login: Username & Password

**2. Onboarding & Personalization**

* News Categories Selection
* Users must select at least one category to receive personalized recommendations.

**3. Main Interface (App Layout)**

* **Newspaper (Home Page - Recommended News)**:  
  A personalized feed based on the selected followed categories.
* **Surprise Me (Trending News)**:  
  Discover trending topics beyond the user's usual preferences.
* **Community Discussions**
  + A space for users to start discussions, ask questions, or debate news topics.
  + Users can create debates (Reddit-style threads).
  + Upvotes & Comments to boost engagement.

**4. Social Features**

* Follow Other Users – See posts & comments from followed users.
* Pinned Articles – Instead of a Like button.

**5. Publish - Create Post**

* A floating pen icon at the bottom of the page.
* Clicking it allows users to write & publish a news post.
* **Post Requirements:**
  + Title
  + News Content (Text, Media, etc.)

**6. Post / Newsletter View**

* Title & News Content (first few characters visible).
* Tap to expand
* Full article displayed in the center.
* **Underneath:** Comments section for discussions.
* **Left Side:** Live debates related to the post.
* **Live Fact-Checking** – A feature that flags misinformation in real time.

**7. Newly Added Features**

* Real-Time Alerts
* Story Rewrites (Repost)
* Multi-Language Support
* Time-Based News Mode
* Reading Mode
* Brief Reading Mode

**8. Getting Certified**

* Publishers will get certified as normal
  + ID / Business doc
  + Review social media accounts

**9. Publishing System (Journalism Mechanism)**

* **Who Can Publish?**
  + Verified publishers (news agencies, journalists, influencers).
  + Users with a high engagement/reputation score.
* **Publishing Steps:**
  + Click "Publish" Button on the navigation bar.
  + Select a category (Politics, Tech, Sports, etc.). + Hashtags
  + Write a caption (1000 characters limit) then deep dive.
  + Attach an external link (if applicable).
  + Optional Media: Attach images or videos.
  + Submit & Review: Punctuation and grammar check before publishing.

**Week 2: Requirements Specification**

**1. Functional Requirements**

1. **User Authentication & Onboarding**
   * Users must create an account with a username and password.
   * Users must be able to log in, log out, and recover passwords.
   * Users must select at least one news category during onboarding to receive personalized recommendations.
2. **News Aggregation & Personalization**
   * Users must receive a personalized news feed based on selected interests.
   * Users must be able to follow specific publishers for tailored content.
3. **Content Management & Publishing**
   * Users must be able to create and publish news articles.
   * Posts must include a title, news content, and optional media attachments.
   * Posts must undergo a basic grammar check before publishing.
4. **Engagement Features**
   * Users must be able to like (pin), comment on, and share news articles.
   * The system must provide a "Surprise Me" section to discover trending news beyond user preferences.
   * Users must be able to start discussions and create Reddit-style threads in a newspaper styled theme.
   * The system must allow users to participate in debates topics.
   * **Fact-Checking & Moderation**
   * Users must be able to report posts for review.
   * The system must flag posts with excessive reports as misleading.
5. **Notifications & Alerts**
   * Users must receive real-time alerts for breaking news.
   * Users must have access to "Morning Digest" and "Night Recap." in the newspaper section.
6. **Certification & Verification System**
   * Publishers must provide ID/business documentation for verification.
   * The system must review social media accounts for authenticity before verification.
7. **Multi-Language & Reading Modes**
   * The system must support AI-powered translations for international news.
   * The system must provide a distraction-free reading mode for long articles.
   * The system must offer a brief reading mode for summary-based news consumption.

**2. Non-Functional Requirements**

1. **Performance & Scalability**
   * The system must handle a large number of concurrent users efficiently.
   * The backend must be optimized for real-time updates.
2. **Availability & Reliability**
   * The system must maintain high uptime for uninterrupted news delivery.
3. **Usability & Accessibility**
   * The system must have an intuitive UI with smooth navigation.
   * The platform must be mobile-responsive for accessibility across devices.
4. **Maintainability & Extensibility**
   * The system must have a modular architecture for easy updates and feature enhancements.

**3. System Constraints**

1. **APIs Dependency**
   * The system relies on external Weather APIs for content aggregation.
2. **Real-Time Processing Load**
   * Implementing real-time updates using the Observer pattern requires significant computational resources.
3. **Multi-Language Support Complexity**
   * AI translation services may introduce latency and errors in translated news articles.
4. **Content Moderation Scalability**
   * Fact-checking and content moderation require automation and manual review, potentially impacting response times.

**4. Assumptions & Dependencies**

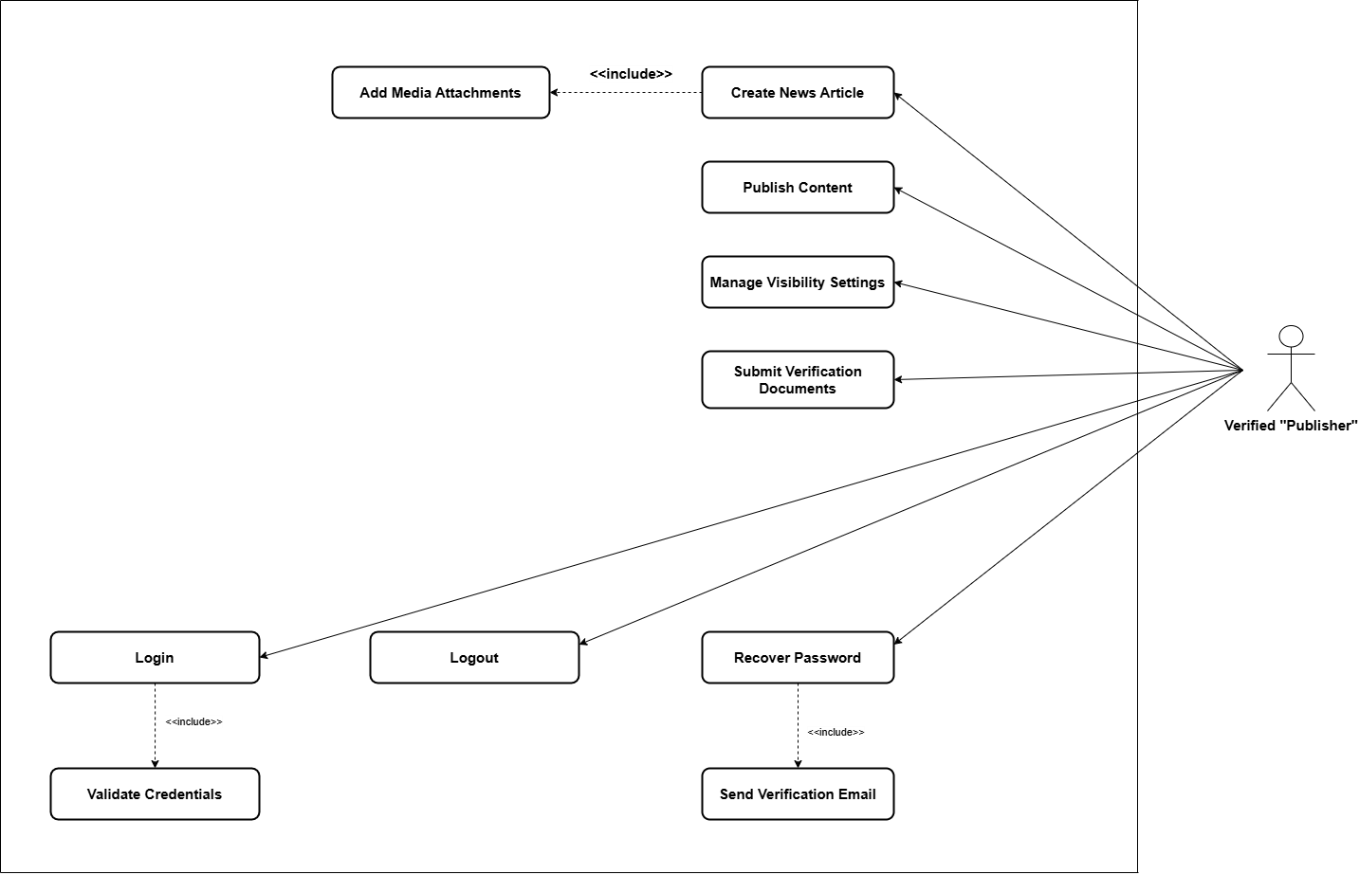
1. **User Base Growth**
   * The platform assumes an initial moderate number of users with scalability options for mass adoption.
2. **User Engagement**
   * The platform assumes users will actively participate in discussions, voting, and reporting content.
3. **Regulatory Compliance**
   * The platform assumes that all content adheres to data privacy laws and moderation policies.
   * The system assumes users have stable internet connections for optimal performance.
4. **Cross-Platform Compatibility**
   * The platform assumes users will primarily access the web application via desktop and mobile browsers.

**Use Case Diagram:**

A screenshot of a computer screen

AI-generated content may be incorrect.

Use Case Diagram 1 of 3

  
Use Case Diagram 2 of 3

A screenshot of a computer

AI-generated content may be incorrect.

Use Case Diagram 3 of 3

**Week 3: System Architecture Design**

**High-Level Architecture Diagram**

A diagram of a software application

AI-generated content may be incorrect.

**Selected Design Patterns & Justification**

**1. Observer Pattern – Frontend & Backend**

The Observer pattern would be implemented to handle notifications in the Verve application by establishing a publisher-subscriber relationship between system events and notification recipients.

**2. Strategy Pattern**

**Justification:** This pattern allows the application to dynamically select different algorithms for filtering articles based on various criteria. It can use different strategies without modifying its code, adhering to the Open/Closed Principle.

**3. Data Transfer Object (DTO) Pattern**

**Justification:** This pattern separates the data transfer objects from the domain model, allowing different representations for communication with the client vs. internal data storage. It provides a clear boundary between the API layer and the domain layer.

**4. Model-View-Controller (MVC) Pattern**

**Justification:** This pattern organizes the application into logical components, separating data management (models), user interface (views), and request handling (controllers), making the codebase more modular and maintainable.

**5. Singleton Pattern**

**Justification:** Spring manages service objects (like ArticleService, AuthService) as singletons, ensuring only one instance exists throughout the application lifecycle, which optimizes resource usage and maintains consistent state.

**6. Factory Method Pattern**

**Justification:** The @Bean methods act as factory methods creating and configuring objects (like CommandLineRunner), separating object creation from its use and providing a clear creation process with proper initialization. These patterns work together to create a modular, maintainable, and scalable application architecture that follows best practices in software design.

**UML Diagrams**

1. **Component Diagram**

A screenshot of a computer screen

AI-generated content may be incorrect.  
  
  
UML Component Diagram

1. **Deployment Diagram**

A group of white cubes with black text

AI-generated content may be incorrect.

UML Deployment Diagram

**Data Flow Overview**

**A diagram of a website

AI-generated content may be incorrect.**

**Week 4: Class Diagram and Initial Implementation**

**Class diagram with Applied Design Patterns**

A black screen with white rectangles

AI-generated content may be incorrect.Class diagram with Applied Design Patterns

**Explanations of Responsibilities per Class**

|  |  |  |
| --- | --- | --- |
| Class Name | Role | Responsibilities |
| NewsAggregator | Serves as the central coordinating class for the application. | Manages the overall flow of the news aggregation process. Initializes and coordinates the interaction between data sources, processing modules, and presentation layers. |
| NewsSource | Represents an external source for retrieving news articles. | Fetches news articles from a specific external source. Encapsulates source-specific fetching logic to ensure modularity. |
| NewsArticle | Encapsulates data and metadata of a single news article. | Stores key attributes such as title, content, URL, and publication date. May include methods for formatting or filtering article content for display purposes. |
| NewsCategory | Categorizes and organizes news by topic. | Classifies news articles into relevant categories (e.g., Technology, Sports, Politics). Facilitates filtering and sorting of articles according to category or preferences. |
| UserPreferences | Captures and applies user-specific settings. | Stores user-specific preferences such as preferred categories, sources, and content filters. Filters and tailors news content based on stored preferences. |
| NewsDisplay | Manages presentation of news articles in the UI. | Formats and displays articles in a user-friendly layout. Provides mechanisms for displaying filtered or categorized news based on user preferences. |
| NewsFetcher | Abstracts the logic of retrieving data from multiple sources. | Centralizes and encapsulates the logic for fetching news from various APIs or sources. Provides an extensible structure for integrating new types of news sources. |

**Class Relationships**

|  |  |
| --- | --- |
| Relationship | Explanation |
| **NewsAggregator → NewsSource** | The aggregator uses multiple sources to collect articles; it coordinates the fetching process by interacting with different NewsSource implementations, each handling its own logic. |
| **NewsCategory → NewsArticle** | Each category holds a list of related articles; NewsCategory groups NewsArticle objects based on topics (e.g., Technology, Sports) for easier filtering and display. |
| **UserPreferences → NewsCategory** | User preferences define which categories are relevant; UserPreferences stores preferred categories and filters out others, interacting with NewsCategory to access selected topics. |
| **UserPreferences → NewsArticle** | Filters and retrieves only the relevant articles; indirectly fetches NewsArticle objects via selected NewsCategory items that match the user’s interests. |
| **NewsDisplay → UserPreferences** | Displays content based on what the user prefers; NewsDisplay pulls the filtered list of articles from UserPreferences and renders them on the frontend. |

**Week 6: Design Pattern Implementation Report**

**Week 9: Database Design Document**

1. ER Diagram

A black screen with white text

AI-generated content may be incorrect.