Group 11 Project Proposal:

Team members and roles:

1. **Scrum Master:** Ethan Green (6930267)
2. **Product Owner:** Mark Zupan (7254956)
3. **Developers:** Mew Tanglimsmarnsuk (7304140), Mohammad Safari (7209513), Esbah Majoka (7357981), Sean Reed (7033251), Long Tong (6790125)

Project Name: AI-Powered Newsletter and Social Media Content Generator

Description:
A software-as-a-service (SaaS) platform that allows users to automatically collect, curate, and summarize news or articles based on specified sources, keywords, or topics. It then generates newsletter-ready content and social media posts for various platforms, complete with scheduling, templates, and multi-channel publishing capabilities.

Main features (basic features):

1. **User Registration & Authentication**

   Users can sign up, log in, and manage their profile through the platform.

2. **Custom News Aggregation**

   Users specify news sources (RSS feeds, links) or keywords to fetch and filter articles automatically.

3. **AI Summarization**

   Integrates with **OpenAI API** to summarize articles, generating concise text for newsletters or social media.

4. **Template Management & Rich Text Editing**

Users can create or edit newsletter templates or social media posts with a **rich text editor** (Draft.js).

5. **Scheduling & Automation**

   Automated scheduling to post or send newsletters periodically.

6. **Dashboard**

   Centralized view for users to see collected articles, edit drafts, and track recent activity.

7. **Multi-Channel Publishing**

   Allows export or direct posting to social media platforms (initially, this might be a basic “export to text” feature, later, integrate platform APIs).

Additional features:

1. **History / Revision Tracking**
   - Users can view past newsletters, previous versions of content, or social media posts.
2. **Analytics & Metrics**
   - Basic insights (e.g., email open rates, social engagement stats) for continuous improvement.
3. **User Preferences**
   - Manage language, topics, content length, and brand style settings for consistent output.
4. **Advanced Customization**
   - Add design components or branding assets (logos, colour schemes) to newsletters and posts.
5. **API Integration (Future)**
   - Allow third parties or enterprise clients to plug into the platform programmatically (Pro feature).

User stories:

1. As a New User, I want to sign up and log in securely so that I can access the platform's features.
2. As a Content Marketer, I want to add my favourite RSS feeds or news links so that the system can automatically fetch relevant articles.
3. As a Social Media Manager, I want the platform to generate AI summaries for posts, so I can quickly publish engaging content to multiple channels.
4. As a Newsletter Editor, I want to schedule my newsletters weekly, so that they go out automatically to subscribers without manual intervention.
5. As a User, I want to customize or design the email template, so that it reflects my brand's look and feel.
6. As a User, I want to review my posting or newsletter history, so I can track which content has already been sent out.

Software Engineer methodology:

We will follow an Agile approach with Scrum-like sprints:

1. **Sprint Planning:** Define sprint goals, tasks, and user stories.
2. **Weekly meetings**: A weekly meeting session to ensure alignment and address blockers.
3. **Sprint Reviews**: Demonstrate completed features to stakeholders for feedback.
4. **Sprint Retrospective:** Reflect on what went well, and what can be improved, and adjust for the next sprint.
5. **Continuous Integration/Continuous Deployment (CI/CD):** Use GitHub Actions or a similar pipeline to automate testing and deployment.

Engineering Requirements

1. **Functional requirements:**

User Management & Authentication

- Must allow for secure user registration, login, and logout (via Flask-Login).
- Users must have unique accounts to store personal preferences (sources, keywords, etc.).

Custom News Aggregation

- Must fetch articles from external sources (RSS feeds, URLs) or using keywords/topics.
- Must store fetched articles in a database (SQLite for MVP).
- Must handle duplicates (i.e., do not re-fetch or store the same article multiple times).

AI Summarization & Content Generation

- Must integrate with the **OpenAI API** to summarize, rephrase, or generate short/long-form text.
- Must display AI-generated summaries or content in a user-friendly format.

Rich Text Editing & Template Management

- Must provide a rich text editor (Draft.js) for customizing newsletter or social post content.
- Must allow saving and reusing template layouts (e.g., brand-styled newsletters, social media post formats).

Scheduling & Automation

- Must allow users to schedule content generation and distribution (newsletter sending or social post creation) at specified intervals.

Dashboard & Content History

- Must provide a dashboard where users can view recent activity, curated articles, or generated drafts.
- Must allow users to see previously created newsletters or social media posts (History page).

Multi-Channel Publishing (Basic)

- Must allow for generating content ready for multiple platforms (e.g., text format for Twitter, HTML email for newsletters).
- (Future enhancement) This could include direct publishing via each platform's API.

Responsive Front-End

- The front-end (React, Material UI) must be responsive and accessible from modern desktop and mobile browsers.

2. **<u>Non-functional requirements:</u>**

Performance

- The system must fetch and summarize articles within a reasonable time (e.g., a few seconds per request).
- Must handle moderate concurrency (multiple users pulling content simultaneously).

Scalability & Modularity

- Must be built in a way that allows switching the database from SQLite to a more robust system (e.g., PostgreSQL) if usage grows.
- The AI integration (OpenAI API) should be modular so it can be swapped or extended with other NLP/LLM services in the future.

Reliability

- The aggregator and summarization services must reliably run without crashing, even if certain sources or APIs fail.
- The scheduling feature must handle unexpected downtime or restarts without losing track of queued tasks.

Security

- Must protect user data with secure authentication and password hashing (Flask-Login, HTTPS, etc.).
- Must store secrets (OpenAI API key, database credentials) in environment variables or a secure location.
- Must prevent unauthorized access to user accounts and avoid exposing sensitive data (like personal email lists).

Maintainability

- Code must be well-documented; particularly the AI integration logic and data flow.

- Project structure (React front-end + Flask back-end) must be clear and organized, enabling future developers to iterate on new features.

Testability

- Must include unit tests and integration tests (front-end with React Testing Library or Jest; back-end with pytest or unittest).
- Key workflows (e.g., user signup, article aggregation, AI summarization) must be covered by automated tests.

Development stacks

- **Front End**
  - **Framework**
    - React.js
  - **UI Library(Styling)**
    - Material-UL (MUI)
    - New Free React Templates - Material UI
    - Login / Sign up Page
    - Dashboard / Homepage
    - Editor Page
    - History Page
  - **Tools and Libraries**
    - Rich Text Editor
      - Draft.js
      - Overview | Draft.js
    - API Requests
      - Axios API
      - Axios API | Axios Docs
    - Routing
      - React Router
      - React Router Official Documentation
- **Back End**

- **Framework**
  - Flask (Python)
- **Database**
  - SQLite / PostgreSQL / MongoDB
- **Authentication System**
  - Flask-Login
- **AI Integration**
  - OpenAI API for summarization and content generation
  - Summaries of scraped or fetched articles.
  - Possibly short or long-form content generation for social media, headlines, etc.

Software Engineer Process:

1. **Requirement Analysis:** Finalize user stories, and prioritize features
2. **Design & Architecture:**
   - High-level system diagram (React front-end, Flask API, AI integration, database).
   - Data models (tables for users, articles, schedules, etc.).
3. **Implementation:**
   - Set up front-end routes, Material UI components, Editor with Draft.js.
   - Build Flask endpoints for user auth, article fetching, scheduling, AI requests.
   - Integrate OpenAI API for summarization.
4. **Testing:**
   - Unit tests (front-end components, Flask routes).
   - Integration tests (API calls to endpoints).
   - User acceptance tests (manual checks on the UI).
5. **Deployment:**
   - Host on a cloud platform (e.g., Heroku, AWS, or Azure).
   - Configure environment variables (OpenAI API key, DB connection).
6. **Maintenance & Iteration:**
   - Gather user feedback.
   - Prioritize enhancements (e.g., advanced analytics, multi-language support).

GitHub page:

MrGreen-ie/COSC-4P02-Group-11

Members of Github Page:

Name	Username
Long Tong	longtongvophi
Sean Reed	reedse_7033251
Mew Tanglimsmarnsuk	uboza10300
Mohammad Safari	Mohammad-150
Esbah Majoka	esbah2003
Ethan Green	MrGreen-ie
Mark Zupan	Zeuop

Meeting schedule, timetable, etc

Meeting 1: Friday January 10th, 2025 - 6:15 PM to about 7:05 PM

- Project Idea: #1 from the slides (AI-Powered Newsletter and Social Media Content Generator)
    - Discussed project ideas, features of the project, roles of each member, the proposal, and future schedules/meetings
- Agile development method (Scrum), Mainly programmed in Python with some HTML, php, javascript, pytest for testing (need to test our system)
- Check out the proposal/information here: https://docs.google.com/document/d/1OKaY8_etTr5CfeDOumQp6w2PKENFJghi34vnQheenSo/edit?usp=sharing

Next Meeting: Tuesday January 14th @ 10AM (During class time).

- Main Topic: Release Planning Meetings (discuss user requirements of our system)

Future meetings: Plan on having typical scrum meetings every other Tuesday starting January 21st. We will discuss requirements, what will be developed, any issues with development, and log what is done.