2. SYSTEM REQUIREMENT STUDY

2.1 User Characteristics

For the DAYGRAM application, the user characteristics would include a variety of attributes and traits to ensure the app meets the needs of its diverse user base. Here are some key user characteristics:

Demographics

- **Age:** 18-45 years.
- **Gender:** All genders, with female-focused features like the pink calendar.
- **Occupation:** Students, professionals, and homemakers.

Behavioral Patterns

- Regular or occasional use for reminders, recipes, blogs, and games.
- Preference for intuitive, mobile-friendly navigation.

Goals and Motivations

- **Goals:** Improve wellness, build confidence, and access actionable self-care resources.
- **Motivations:** Create a balanced lifestyle, engage in community support, and track personal growth privately.

Pain Points

- **Challenges:** Inconsistent routines, lack of motivation, and unreliable health information.
- **Needs:** Reliable reminders, personalized tools, and a supportive community.

By understanding these traits, DAYGRAM ensures a user-centric, effective platform for self-care and wellness.

2.2 Software and Hardware Requirements

Software and hardware Requirements are used to describe the minimum hardware and software requirements to run the Software. These requirements are described below:

2.2.1 Software Requirements

1. **Backend Development:**
    - **Programming Language**: Python 3.8+ (compatible with Django).
    - **Framework**: Django 3.2+ for building the backend.
    - **Database:**
        - Default: SQLite (for development).
    - **Web Server**: Gunicorn or uWSGI for running the Django application.
2. **Frontend Development:**
    - HTML, CSS, and JavaScript for user interface.
    - **Framework/Library:** plain JavaScript for dynamic content.
3. **Package Management:**
    - **Pip:** To install required Python packages.
    - **Virtual Environment:** Virtualenv or Conda for managing dependencies.
4. **Other Tools:**
    - **Django Libraries:**
        - Django Rest Framework (DRF) for APIs.
        - Celery for task scheduling (e.g., sending reminders).
        - Pillow for image processing.
    - **Authentication**: Django Allauth or custom authentication for user accounts.
    - **Notifications:** Firebase Cloud Messaging (FCM) or custom services for reminders.
5. **Hosting and Deployment:**
    - **Cloud Services:** AWS, Google Cloud, DigitalOcean, or Heroku**.**
    - **Docker:** For containerization during development and deployment.
6. **Version Control:**

- Git and platforms like GitHub, GitLab.

7. **Development Tools:**
   - **IDE:** Visual Studio Code.
   - **Debugging Tools:** Django Debug Toolbar.
8. **AI and Chatbot Integration:**
   - Natural Language Processing (NLP): OpenAI API or Google Dialogflow for AI chatbot functionality.

2.2.2 Hardware Requirements

1. **Server-Side (Hosting):**
   - **Processor:** Multi-core CPU (Intel Xeon or AMD EPYC recommended).
   - **RAM:**
     - **Development:** 4GB minimum.
     - **Production:** 8GB-16GB or higher, depending on traffic.
   - **Storage:** SSD with a minimum of 250GB for development and 500GB+ for production.
   - **Bandwidth**: High-speed connection to support multiple simultaneous user requests.
2. **Development Workstations:**
   - **Processor:** Quad-core CPU or higher (e.g., Intel i5 or i7).
   - **RAM:** 8GB minimum (16GB recommended).
   - **Storage:** SSD with 256GB or more.
   - **OS Compatibility**: Windows 10+, macOS 10.13+, or Linux.
3. **Client-Side (User Devices):**
   - **Desktop/Laptop:**

        - **Browser:** Latest versions of Chrome, Firefox.
    - **Mobile Devices:**
        - **Operating Systems:** Android 8.0+ or iOS 12+**.**

2.2.3 Functional Requirements

1. **Search and Filter**
    - Users can search and filter content (blogs, recipes, remedies) by keywords and categories.
2. **User Registration and Authentication**
    - Users can sign up, log in, and reset passwords securely via email/phone.
3. **Notifications**
    - Users receive reminders for hydration, exercise, skincare, and menstrual cycles.
    - Notification preferences can be customized.
4. **Blog and Content Management**
    - Users can view, post, comment, like, and share blog content.
5. **Pink Calendar**
    - Female users can input menstrual cycle dates and receive automated reminders.
6. **Healthy Recipes and Diet Plans**
    - Users can browse recipes and receive personalized diet plans.
7. **Stress-Busting Games**
    - Interactive stress-relief games available, with time-tracking for wellness.
8. **Security**
    - User data is encrypted, and secure logouts are enabled.

9. **Settings and Customization**
    - Users can customize themes, notification settings, and data preferences**.**
10. **AI Chatbot**
    - The chatbot offers personalized advice and suggests content based on user input.
11. **Help and Support**
    - Access to a help section and support for unresolved issues.
12. **Activity Tracking**
    - Users can track liked and commented posts, and manage their activities.
13. **Follower and Community Engagement**
    - Users can follow others and receive notifications for community interactions.

2.2.4 Non Functional Requirements

The non-functional requirements for the Daygram website describe how the system should perform and the quality attributes it must meet. Here are some key non-functional requirements for Daygram:

Non-Functional Requirements for DAYGRAM (Shortened)

1. **Performance**
    - Website should load within 3 seconds and support 1,000 concurrent users.
2. **Scalability**
    - Platform should scale easily to handle more users and data.
3. **Availability**
    - 99.9% uptime with failover mechanisms for system failures.
4. **Security**
    - Data encrypted in transit and at rest; secure authentication methods.

- Protection against common security threats (e.g., SQL injection, XSS).

5. **Usability**
    - Clean, intuitive design for easy navigation and task completion.
6. **Accessibility**
    - Adhere to WCAG 2.1 guidelines for users with disabilities.
7. **Compatibility**
    - Compatible with major browsers and mobile devices; responsive design.
8. **Backup and Recovery**
    - Regular data backups and recovery procedures in place.
9. **Localization and Internationalization**
    - Support for multiple languages and regional settings.
10. **Legal and Compliance**

- Compliant with data privacy regulations (e.g., GDPR, CCPA).

11. **Maintainability**

- Well-documented, industry-standard code for easy updates and fixes.

12. **Monitoring and Logging**

- Real-time monitoring and logs for system health and troubleshooting.

2.3 Constraints

2.3.1. Technical Constraints

1. **Platform Compatibility**
    - Works seamlessly across major browsers, devices, and varying screen sizes.
2. **Performance Limitations**
    - Optimized for smooth use on older devices with limited resources.
3. **Backend Framework**

- Uses Python Django exclusively for backend operations.

4. **Database**
   - Relies on PostgreSQL or MySQL, compatible with Django ORM.
5. **Third-Party Integrations**
   - Supports only APIs and libraries compatible with Django and frontend tech.
6. **Security**
   - Implements SSL/TLS for data in transit and AES for data at rest.
7. **Hosting**
   - Restricted to cloud platforms supporting Django and scalable deployment.

2.3.2. Resource Constraints

1. **Team**
   - Limited developers skilled in Python Django and frontend tech.
2. **Budget**
   - Restricted funding for hosting, APIs, and tools.
3. **Time**
   - Tight deadlines for development and deployment.
4. **Hardware**
   - Standard machines; no high-performance testing servers.
5. **Testing**
   - Limited devices for cross-platform testing.
6. **Maintenance**
   - Small team for post-launch updates.

2.3.3. Security Constraints

1. **Data Encryption**

- All user data must be encrypted during transmission (SSL/TLS) and storage (AES).

2. **Authentication**
    - Only secure authentication methods (e.g., hashed passwords, OAuth) are allowed.
3. **Access Control**
    - Role-based access to ensure users only access authorized features and data.
4. **Compliance**
    - Must adhere to regulations like GDPR and CCPA for user data protection.
5. **Vulnerability Mitigation**
    - Protection against common threats such as SQL injection, XSS, and CSRF attacks.
6. **Logging and Monitoring**
    - Secure logging to detect and respond to unauthorized access or data breaches.

2.3.4. Legal and Regulatory Constraints

1. **Data Privacy**
    - Protect user information like names, email IDs, and other personal details; avoid sharing or selling data.
2. **User Consent**
    - Clearly explain why data is collected and get user permission during registration.
3. **Age Verification**
    - Ensure users confirm they are of legal age to use the platform (e.g., 13+).
4. **Content Moderation**
    - Monitor posts and comments to prevent inappropriate or harmful content.

5. **Copyright Respect**
    - Use only original or freely available resources (images, texts) to avoid copyright issues.
6. **Accessibility**
    - Make the website simple and usable for all, including users with basic devices or internet access.
7. **Health Disclaimer**
    - Add a note that tips and advice on the website are not professional medical recommendations.
8. **Terms of Use**
    - Write clear rules about how users can interact with the platform to prevent misuse.

2.3.5. Operational Constraints

1. **Hosting Limitations**
    - Use free or low-cost hosting platforms (e.g., Heroku, Render, or Firebase) due to budget constraints.
2. **Team Size**
    - Limited team members, requiring multitasking across design, development, and testing roles.
3. **Time Constraints**
    - Strict deadlines for project completion within academic schedules.
4. **Resource Availability**
    - Dependence on readily available tools and technologies, such as open-source libraries and free APIs.
5. **Testing Environment**
    - Limited access to devices for comprehensive testing; rely on personal laptops or mobile phones.

6. **Scalability**
    - Focus on small-scale functionality, as large-scale user traffic may not be feasible with available resources.
7. **Maintenance**
    - Minimal post-deployment support, with updates and fixes dependent on academic timelines.
8. **Documentation**
    - Simplified documentation with a focus on essential features and basic user guides.

2.3.6. User Constraints

1. **Technical Knowledge**
    - Users may have limited technical expertise, so the platform must be simple and intuitive.
2. **Device Limitations**
    - Users may access the platform on low-end devices with limited processing power and screen sizes.
3. **Internet Access**
    - Dependence on stable internet connectivity, which might not always be available for all users.
4. **Time Availability**
    - Users may have limited time to engage with the platform due to academic or personal commitments.
5. **Customization Needs**
    - Users might expect basic customization options but not extensive personalization features.
6. **Data Privacy Concerns**
    - Users will expect their data to remain secure and not be shared or misused.

7. **Engagement Level**
    - Users might prefer quick and straightforward interactions over complex workflows.

2.3.7. Environmental Constraints

1. **Development Environment**
    - Limited to personal laptops or basic lab systems.
2. **Hosting Environment**
    - Use of free or low-cost hosting platforms with restricted resources.
3. **Testing Environment**
    - Reliance on personal devices for testing, limiting variety.
4. **Network Dependency**
    - Stable internet required for platform operation.
5. **Collaboration Tools**
    - Dependence on free tools like GitHub or Google Drive for teamwork.

2.3.8. Integration Constraints

1. **Backend Integration**
    - Must use Python Django for backend operations.
2. **Database**
    - Limited to relational databases like PostgreSQL or MySQL, compatible with Django.
3. **Third-Party APIs**
    - Only integrate free or low-cost APIs compatible with Django and frontend frameworks.
4. **Frontend Framework**

- Restricted to simple frameworks (e.g., HTML, CSS, JavaScript) for ease of use.

5. **Cloud Services**
    - Use affordable or free cloud platforms for deployment and storage.

2.4 Assumptions and Dependencies

2.4.1 Assumptions

When developing the Daygram application, several assumptions might be made to guide the project. Here are some potential assumptions for the Daygram application:

1. **Device Access**
    - Users have devices with stable internet connectivity.
2. **Technical Skills**
    - Users can navigate a simple and intuitive platform.
3. **Data Privacy**
    - Users trust the platform to handle personal information securely.
4. **Usage Frequency**
    - Users will engage with the platform regularly for self-care.

2.4.2 Dependencies

Dependencies for DAYGRAM

1. **Backend Framework**
    - Relies on Python Django for backend development.
2. **Database**
    - Dependent on PostgreSQL or MySQL for storing user data and application content.
3. **Frontend**
    - Uses HTML, CSS, and JavaScript for building a user-friendly interface.

4. **Hosting Platform**
    - Requires free or low-cost hosting services (e.g., Heroku, Render, Firebase).
5. **Third-Party APIs**
    - Relies on APIs for additional features like notifications, AI chatbot, and reminders.
6. **Libraries and Tools**
    - Dependent on open-source libraries for efficient development.
7. **Internet Connectivity**
    - Stable internet is essential for app functionality, hosting, and user access.