RuralDex development documentation

1. Technique Overview

The RuralDex platform is built as a web application using a combination of front-end and back-end technologies. The front-end is built using popular web technologies such as HTML, CSS, and JavaScript, with user interface. The back-end is built using a server-side Django technology, with a scalable and reliable MySQL database used to store patient information and other data.

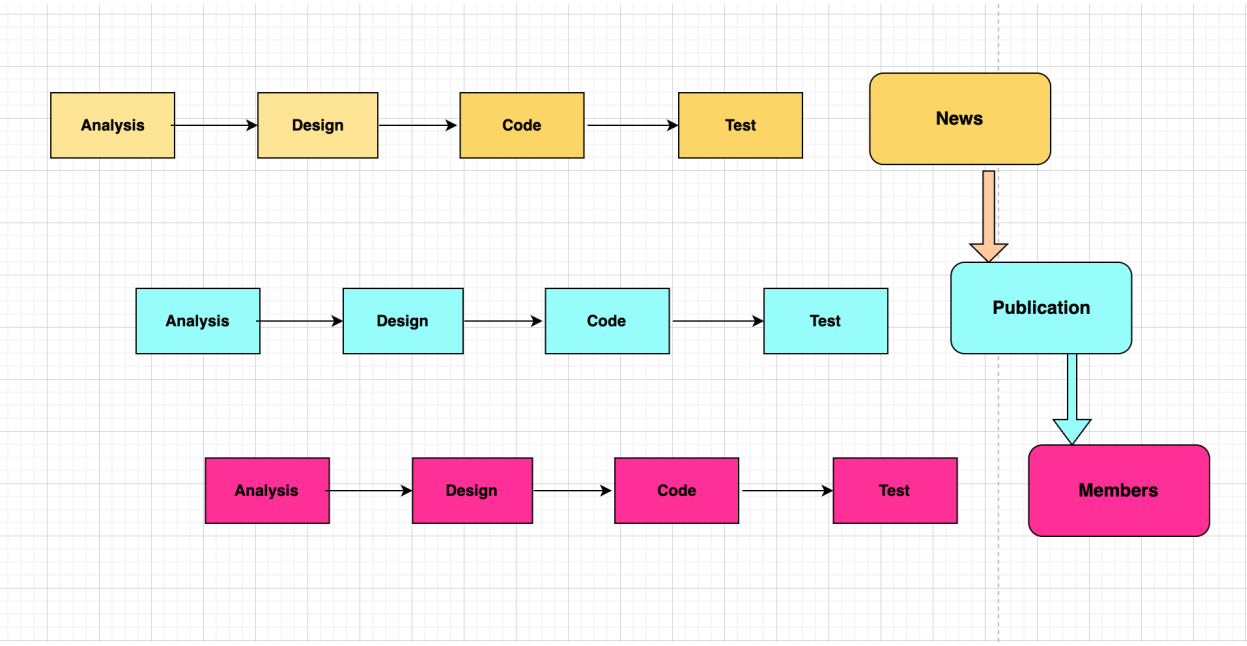
The system is deployed on a cloud-based hosting service such as AWS, which provides scalability, reliability, and security. The system is designed to be accessible from any device with an internet connection, making it easy for researcher to access the system from home or on the go.

2. Development Methodologies (Incremental Model)

The Incremental Model is an iterative software development model that breaks down the development process into smaller, more manageable pieces. Each piece is developed and delivered incrementally, with new features added in each iteration.

2.1 The Incremental Model workflow

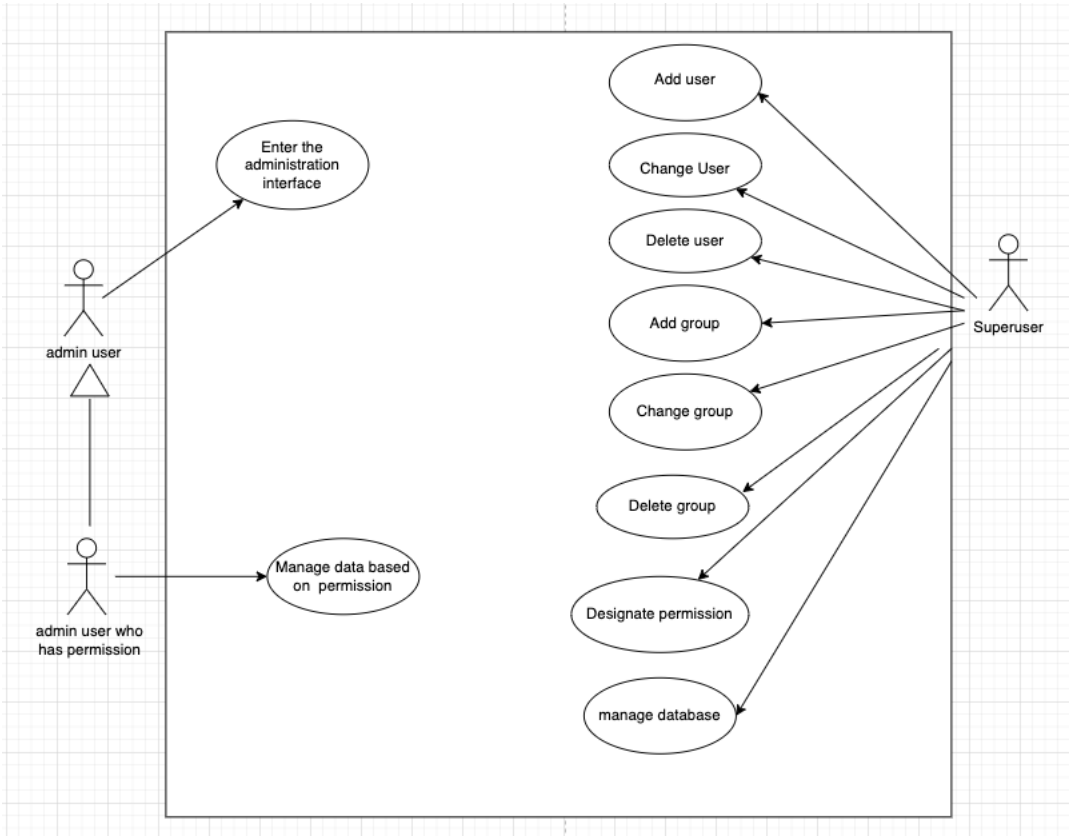
1. Identify key features and functionalities: Start by identifying the most important features and functionalities that RuralDex system needs to have.
2. Break down features into smaller increments: Once key features and functionalities have been identified, breaking them down into smaller, more manageable increments that can be developed and delivered incrementally.
3. Develop and deliver each increment: Once each increment has been defined, developing and delivering it incrementally. Each increment should be fully functional and provide value to users.
4. Test and refine: After each increment is developed and delivered, testing it thoroughly to ensure that it is working as expected. Using feedback from users to refine the system and make improvements.
5. Repeat the process: Once one increment has been completed, repeating the process for the next increment. Continuing to develop and deliver each increment, incrementally testing and refining each one.


Analysis
Design
Code
Test
News
Analysis
Design
Code
Test
Publication
Analysis
Design
Code
Test
Members

3. Requirements Analysis

3.1 Use case diagram

1. User Management in Administration



Name	User Management in Administration
Participating Actors	1. Superuser - A privileged user who has complete control over the administration system.

	2. Admin User - A user with limited access to the administration system, based on permissions assigned by the Superuser.
Entry Conditions	1. Superuser/Admin User should have a valid login credentials. 2. Superuser/Admin User has access to the administration system interface.
Flow of Events	1. Superuser/Admin User navigates to the login page of the administration system. 2. Superuser/Admin User enters their login credentials. 3. The administration system verifies the login credentials. 4. On successful verification, Superuser/Admin User gains access to the administration system. 5. Superuser chooses to manage users (add, modify, delete) or manage application data. 6. When managing users, the Superuser selects a user to add, modify, or delete. 7. The Superuser may also designate permissions to users. 8. If managing application data, the Superuser performs the necessary actions and the changes are saved in the system. 9. Admin User can manage application data as per their assigned permissions.

	10. Admin User performs the necessary actions on the application data and the changes are saved in the system.
Exit Conditions	1. Superuser/Admin User chooses to logout from the administration system. 2. Superuser/Admin User is automatically logged out due to inactivity for a specified time period.

2. Member Interactions in RuralDex Platform

Name	Member Interactions in RuralDex Platform

Participating Actors	Member - A registered user of the RuralDex platform.
Entry Conditions	1. The prospective Member has access to the RuralDex platform's registration page.
Flow of Events	For new users: 1. A prospective Member navigates to the registration page of the RuralDex platform. 2. They enter the required information to sign up. 3. The RuralDex platform verifies the entered information. 4. On successful verification, the new Member receives a confirmation, and they can now login to the RuralDex platform. For existing Members: 5. Member navigates to the login page of the RuralDex platform. 6. Member enters their login credentials. 7. The RuralDex platform verifies the login credentials. 8. On successful verification, the Member gains access to the RuralDex platform. 9. The Member can choose to manage their profile (change personal profile). 10. Member performs the necessary actions on their profile, and the changes are saved in the system. 11. Member can choose to manage their publications (create, modify, delete).

	12. Member selects a publication to create, modify, or delete. 13. Member performs the necessary actions, and the changes are saved in the system. 14. Member reads news on the RuralDex platform. 15. Member leaves a comment on a news article. 16. The comment is saved and displayed under the news article.
Exit Conditions	1. Member chooses to log out from the RuralDex platform. 2. Member is automatically logged out due to inactivity for a specified time period.

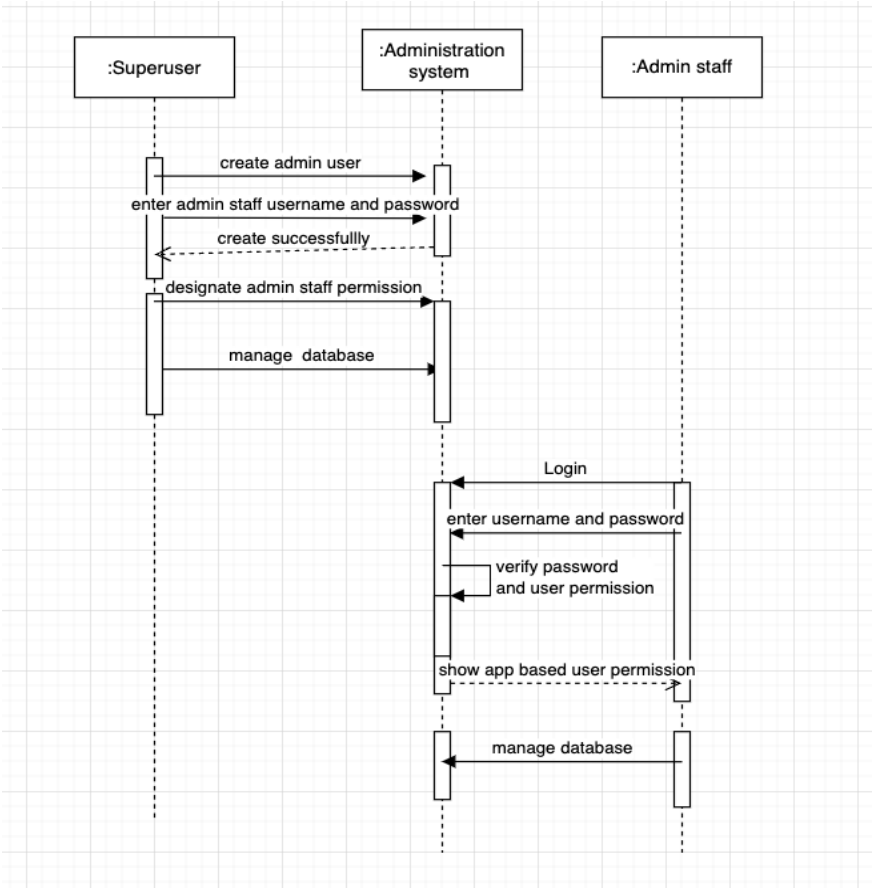
3.2 Sequence diagram

A sequence diagram is a type of interaction diagram in UML (Unified Modeling Language) that depicts the flow of messages and interactions between different objects or components in a system. It represents the behavior of a single use case or a specific scenario within a system.

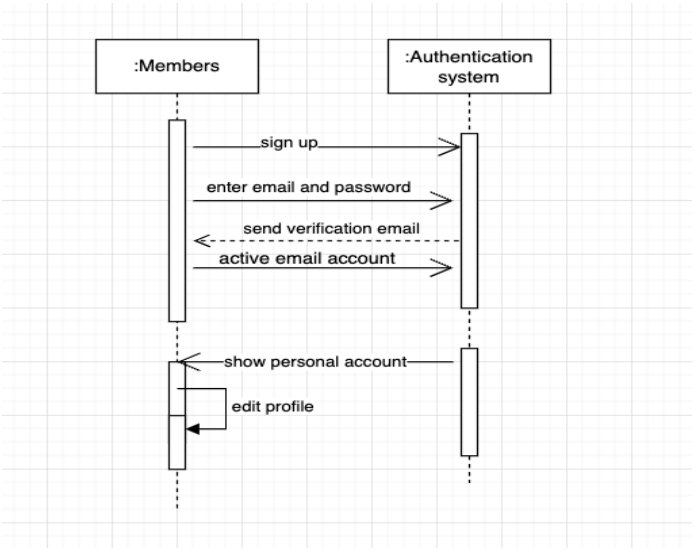
In a sequence diagram, objects are represented as vertical lifelines, and messages between objects are depicted as arrows. The sequence of messages indicates the order of interactions between objects over time. Each message can be labeled with the name of the operation or method being invoked, along with any parameters or return values.

Sequence diagrams are useful for visualizing and understanding the dynamic behavior of a system, capturing the interactions between objects, and identifying the order of operations or method calls during the execution of a use case.

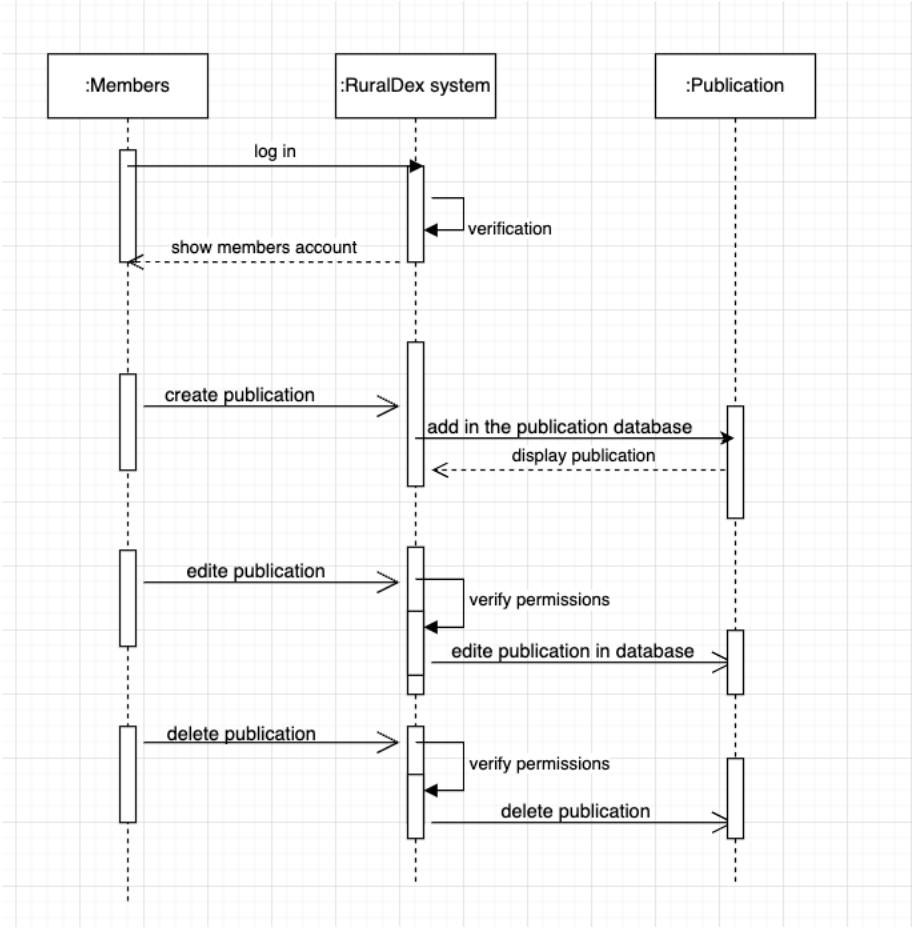
1. Administration sequence diagram



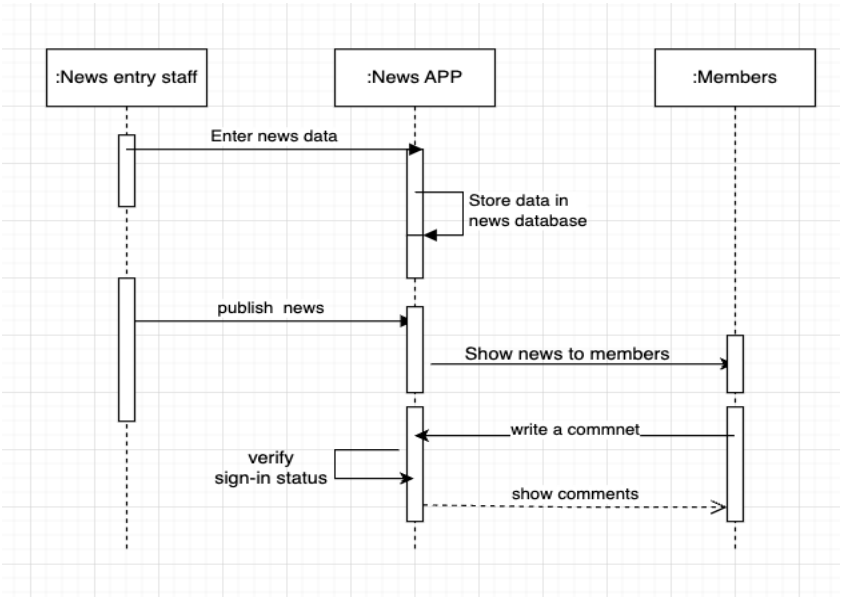
2. Authentication sequence diagram



3. Publication sequence diagram



4. News sequence diagram



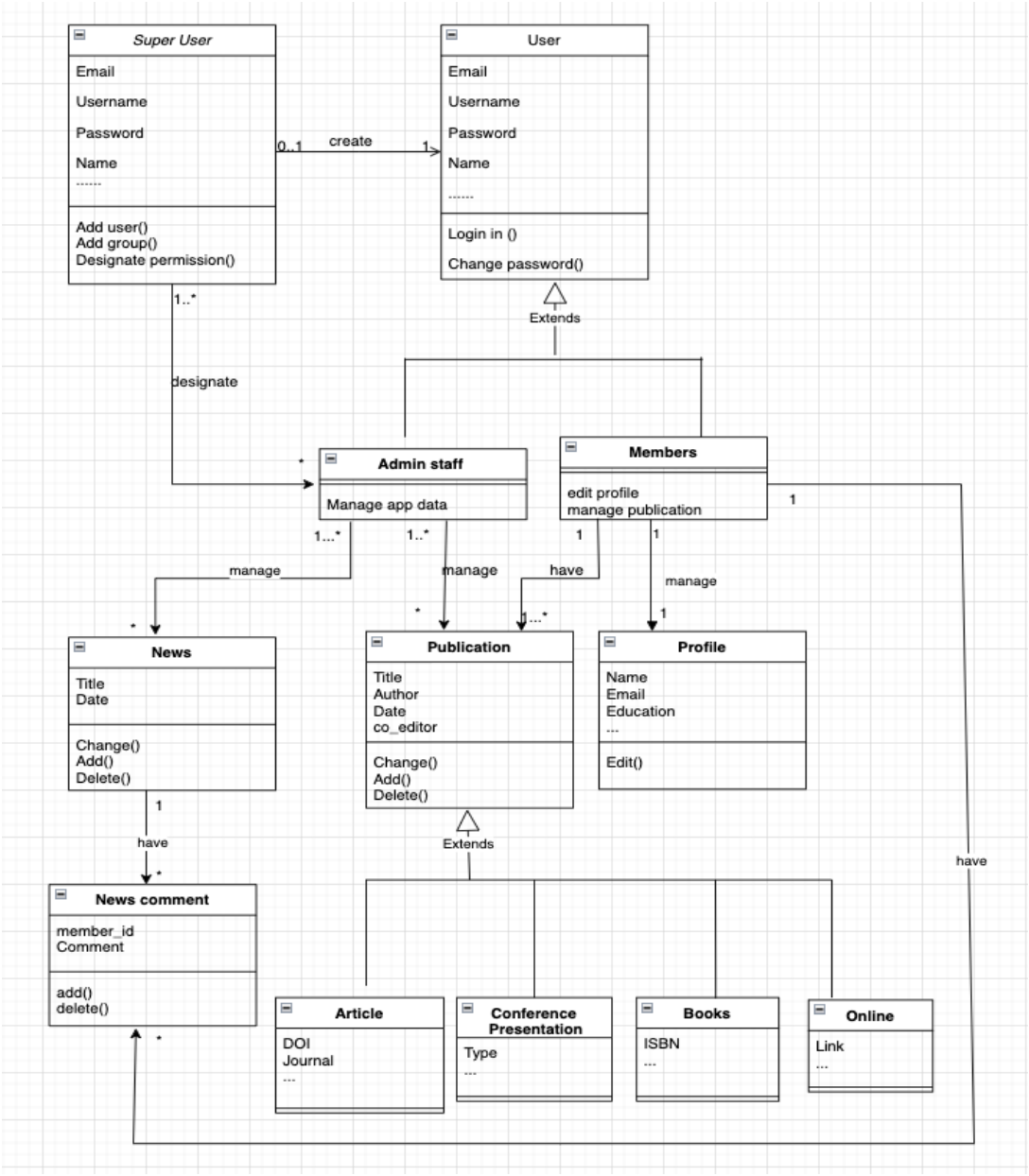
3.3 Class diagram

A class diagram is a type of static structure diagram in UML (Unified Modeling Language) that illustrates the structure and relationships among classes in a system. It provides a visual representation of the classes, their attributes, methods, and the associations or relationships between them.

In a class diagram, classes are represented as rectangles, with the class name written inside the rectangle. The class attributes (variables) are listed below the name, and the methods (functions or operations) are listed above the name. The associations between classes are depicted as lines connecting the classes, often with arrows indicating the direction of the relationship.

Class diagrams also allow for the representation of inheritance or generalization relationships using inheritance arrows, indicating that one class inherits from another.

Class diagrams are commonly used during the design and analysis phase of software development to model the static structure of a system, define the classes and their relationships, and aid in understanding the overall architecture of the system. They serve as a blueprint for implementing the system's functionality and are also useful for documentation and communication among stakeholders.

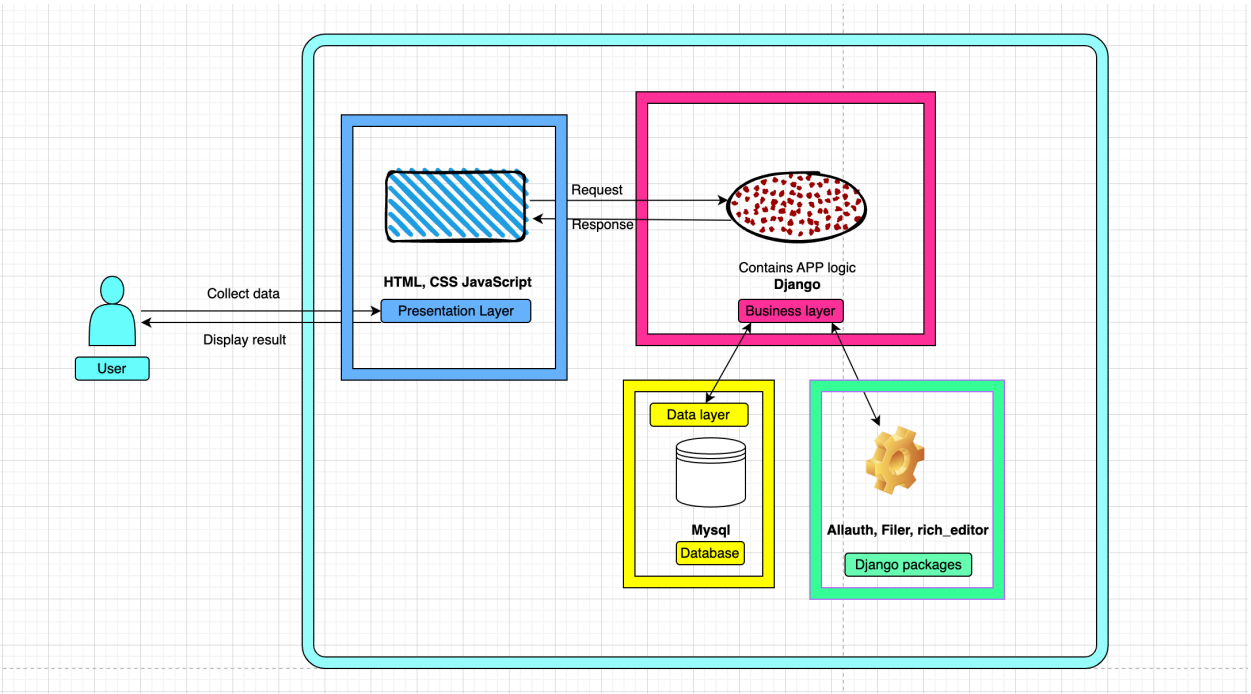

Super User
Email
Username
Password
Name

Add user()
Add group()
Designate permission()
0..1
create
1
User
Email
Username
Password
Name

Login in ()
Change password()
1..*
designate
Extends
Admin staff
Manage app data
Members
edit profile
manage publication
1...*
1..*
1
1
manage
manage
have
manage
1...*
News
Title
Date
Change()
Add()
Delete()
Publication
Title
Author
Date
co_editor
Change()
Add()
Delete()
Profile
Name
Email
Education
...
Edit()
1
have
Extends
have
News comment
member_id
Comment
add()
delete()
Article
DOI
Journal
...
Conference Presentation
Type
...
Books
ISBN
...
Online
Link
...

4. System Design

4.1 System architecture



1. Presentation layer: This layer is responsible for presenting the user interface to members and administrators. It includes web pages and other front-end components that allow users to interact with the system.
2. Business layer: This layer handles the system's business logic, including news, publication, members and system administration. It includes the back-end server and database management systems.
3. Integration layer: This layer manages the communication between the application layer and external Django packages. It includes APIs and other interfaces for integrating with third-party services such as Django-Allauth, Django-Filer.

4. Data layer: It is designed to efficiently store and retrieve data, protect sensitive information, handle increasing amounts of data, and ensure data quality.

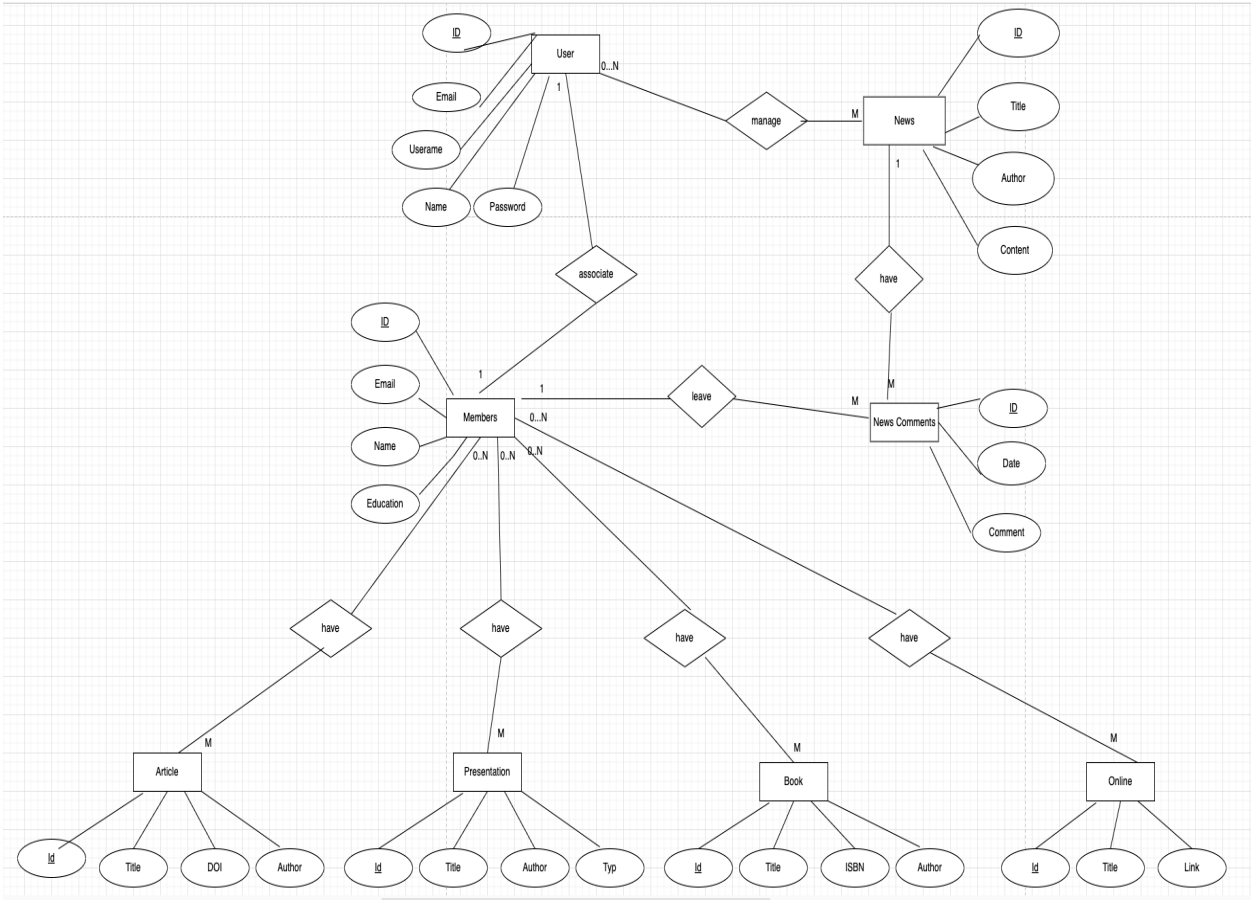
4.2 Database design

An ER (Entity-Relationship) diagram is a visual representation of the entities (objects or concepts), attributes, and relationships within a database or information system. It is commonly used in database design to model the structure and organization of data.

In an ER diagram, entities are represented as rectangles, and their attributes are listed within the rectangles. Relationships between entities are depicted as lines connecting the entities, often with labels indicating the nature of the relationship, such as "has," "belongs to," or "is associated with." Cardinality, denoting the number of instances of one entity that can be related to instances of another entity, is often shown using notations such as "1" or "N."

ER diagrams can also include additional notations such as primary keys, foreign keys, and weak entities. Primary keys uniquely identify each instance of an entity, while foreign keys establish relationships between entities. Weak entities are entities that depend on another entity for their existence.

ER diagrams help in visualizing and designing the structure of a database or information system, including the entities involved, their attributes, and the relationships between them. They assist in capturing the data requirements and constraints of a system and serve as a blueprint for database implementation and development. ER diagrams are widely used in the analysis, design, and documentation of databases and are an essential tool in database management and software engineering.


ID
User
0...N
Email
1
manage
M
News
ID
Title
Username
Author
Name
Password
1
Content
associate
have
ID
1
Email
1
leave
M
M
News Comments
ID
Members
0...N
Name
0..N
0..N
0..N
Date
Education
Comment
have
have
have
have
M
M
M
M
Article
Presentation
Book
Online
Id
Title
DOI
Author
Id
Title
Author
Typ
Id
Title
ISBN
Author
Id
Title
Link

5. Implementation and coding

5.1 Development platform

A development platform is a combination of hardware and software tools that provide a complete environment for developing, testing, and deploying software applications.

5.2 Operating System

The development platform can be hosted on any modern operating system such as Windows, macOS, or Linux.

5.3 Server-Side Framework

Django (4.1.7), a Python-based web framework, is used to develop the back-end of the system. It is a high-level web framework for building web applications in Python. It follows the Model-View-Controller (MVC) architectural pattern and provides a rich set of tools and libraries to simplify web development. One of the key features of Django is the Django Admin interface, which provides a built-in administrative interface for managing the application's data.

Django Admin is a powerful tool that allows developers to quickly create a webbased administrative interface for managing their application's data. The Django Admin interface is generated automatically based on the models defined in the application and provides a user-friendly interface for performing CRUD (Create, Read, Update, Delete) operations on the data.

5.4 Client-Side Frameworks and Libraries

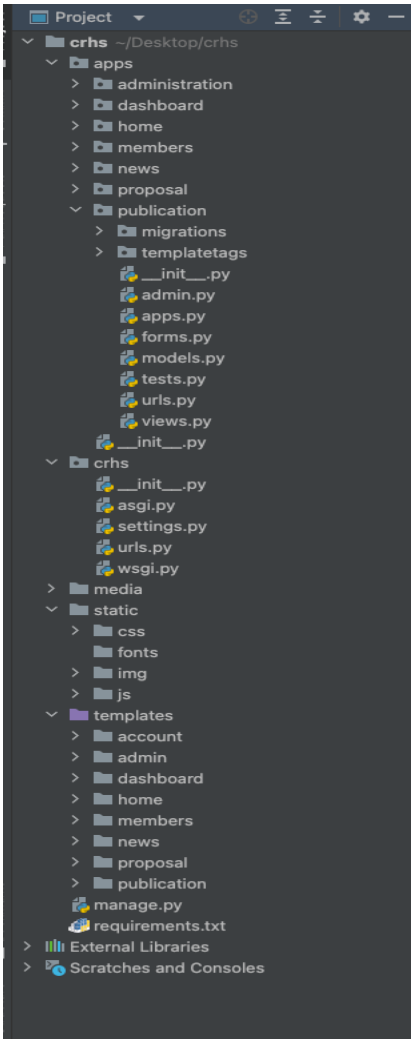
The front-end of the system is developed using HTML, CSS, and JavaScript. To

simplify development and improve user experience, popular client-side frameworks and libraries like Bootstrap, jQuery be used.

5.5 Database Management System

MySQL is used as the database management system to store patient and doctor data, appointment schedules, and other related data. MySQL is a popular opensource relational database management system (RDBMS) that is widely used in web development. MySQL is known for its reliability, scalability, and ease of use, making it a popular choice for both small and large applications.

5.6 Project structure



• Apps folder

This folder contains all the business logic of the application, organized into separate apps. Each app represents a specific feature or functionality of the application. Such as Publication, News, Members, Home, Proposal, Members…

Each app folder contains files such as models.py (defining the database models), views.py (defining the views or logic for handling HTTP requests), and urls.py (mapping URLs to the appropriate views).

1. migrations:

   This folder contains database migration files. Migrations are used to manage changes to your database schema over time. Django automatically generates migration files when you make changes to your models.

2. models:

   This file typically is models.py. The models.py file defines the database models for the app. Models are Python classes that represent database tables and their fields.

3. views:

   This file typically is named views.py. The views.py file contains the views or logic for handling HTTP requests. Views receive HTTP requests, process the data, and return responses. Views can render HTML templates or return data in other formats like JSON.

4. urls:

   This file is urls.py. The urls.py file maps URLs to the appropriate views. It defines the URL patterns for the app and specifies which view should handle each URL.

5. forms:

This file is used to store form classes. Forms are used for data validation and handling user input. They can be defined in a file named forms.py.

6. tests:

   This file is used to store test cases. Test cases are written to ensure that your app functions as expected. They can be defined in a file named tests.py.

7. admin:

   This file is used to customize the Django admin interface. It typically contains a file named admin.py. The admin.py file allows you to register your models with the admin interface and define their display and behavior.

• The settings.py

This file is a crucial part of a Django project. It is located in the project root directory and contains various configuration settings that control the behavior of your Django project.
This file contains all the settings for the application, such as database configuration, installed apps, and middleware. Overall, the project structure of a Django application is designed to be modular and organized, making it easy to add new features and functionality to the application as it grows.

1. Database Settings:

   The DATABASES setting defines the configuration for your project's database connection. You can specify database engines (MySQL) and connection details like host, port, username, password, and database name.

2. Installed Apps:

   The INSTALLED_APPS setting is a list of all the Django apps installed in your project.

It includes both built-in apps provided by Django and custom apps you create. Each app must be added to this list for Django to recognize and include its functionality.

3. Middleware:

   The MIDDLEWARE setting defines a list of middleware classes. Middleware is a way to process HTTP requests and responses globally across your project. It can perform functions such as authentication, session management, or modifying headers.

4. Template Settings:

The TEMPLATES setting contains a list of template engines and their configurations. You can specify template directories, context processors, and other settings related to template rendering.

5. Internationalization and Localization:

   The LANGUAGE_CODE setting defines the default language code for your project. The TIME_ZONE setting specifies the default time zone. These settings affect how dates, times, and other localized data are displayed.

6. Security and Authentication:

The SECRET_KEY setting is a unique key used for cryptographic signing and securing session data. The AUTHENTICATION_BACKENDS setting defines the authentication backends used for user authentication.

• Templates folder:

This folder contains all the HTML templates used by the application. Each app can have its own templates folder, which contains templates specific to that app. Such as publication pages, news pages, members pages… The templates folder in a Django project is used to store HTML

templates that are used to generate dynamic web pages. Templates provide a way to separate the presentation logic from the business logic in your Django views.

• Static folder: This folder contains all the static files used by the application, such as CSS, JavaScript, and images.

• Media folder:

This folder is used to store user-generated content, such as images uploaded by users.

• The requirements.txt:

This file is a commonly used convention in the Python ecosystem to specify the dependencies of a project. It allows you to define a list of external libraries and their specific versions required for your project to run correctly. Running pip install -r requirements.txt to install all the listed libraries.

6. Hosting and Deploying

Deploying a Django web application on AWS can provide a reliable and scalable hosting environment that offers high performance, security, and cost-effectiveness. AWS also offers a variety of tools and services that can simplify the deployment and management process, allowing developers to focus on building and improving their applications.

1. Set up an AWS EC2 instance: Launch an EC2 instance and choose an Ubuntu Server AMI. Make sure to configure security groups and open the necessary ports for HTTP and HTTPS traffic.
2. Install required packages: Install required packages like Python, pip, and virtualenv.
3. Create a virtual environment: Create a virtual environment for Django application to manage dependencies and ensure compatibility between different projects.
4. Clone Django project: Clone Django project repository onto EC2 instance.
5. Install dependencies: Install dependencies for project using pip, which can be done by activating virtual environment and running pip install -r requirements txt.
6. Configure Django application: Configure Django application by updating settings file with the appropriate database information and static file settings.
7. Run database migrations: Run database migrations using the python manage. py migrate command to create necessary database tables.
8. Collect static files: Collect static files using the python manage.py collectstatic command to ensure static files are served correctly.
9. Test application: Test application by running the development server using

the python manage.py runserver command and accessing application in a web browser.

10. Configure web server: Configure web server (Nginx) to serve Django application by creating a virtual host configuration file and enabling the necessary modules.
11. Configure HTTPS: Configure HTTPS using an SSL certificate from a certificate authority like Let's Encrypt.

RuralDex URL: www.ruraldex.com