**Kanban Board Management System - Architecture Document**

# **1. Project Overview**

**System Name**: Kanban Board Management System

**Description**: A web-based Kanban board application designed to manage task workflows for individual users and teams. Features include user creation, task creation, status updates, task editing, and a history log for users and tasks. The app will support efficient project management with a user-friendly interface.

# **2. Objectives**

* **Scalable**: Designed to handle multiple users.
* **Reliability**: Ensures high availability with minimal downtime.
* **Security**: Implements secure user authentication and password encryption.

# **3. Technology Stack**

* **Backend**: Flask (Python framework)
* **Database**: MySQL (for relational data)
* **Frontend**: HTML/CSS/JavaScript with Flask templates
* **Containerization**: Docker
* **Version** **Control**: Git
* **Deployment**: Docker

# **4. System Requirements**

## **4.1 Functional Requirements**

* **User Management**: Register, authenticate, and manage user information.
* **Task Management**: Create, edit, delete, and update tasks.
* **Kanban Board**: Display tasks in “To Do,” “In Progress,” and “Done” columns with drag-and-drop capabilities.
* **History Log**: View task history in a modal showing changes over time.

## **4.2 Non-functional Requirements**

* **Scalability**: Use of Docker to enable horizontal scaling of services.
* **Availability**: Use container orchestration for high availability and resilience.
* **Security**: Protect data through Flask’s authentication and MySQL’s data encryption.

# **5. Architecture Components**

## **5.1 Overview Diagram**

+----------------------------------+

| Browser |

| (HTML/CSS/JavaScript - Flask |

| Templates) |

+---------------------|------------+

|

|

+-------------v--------------+

| Flask |

| (Backend) |

+-------------|--------------+

|

|

+-------------v--------------+

| MySQL |

| (Database) |

+----------------------------+

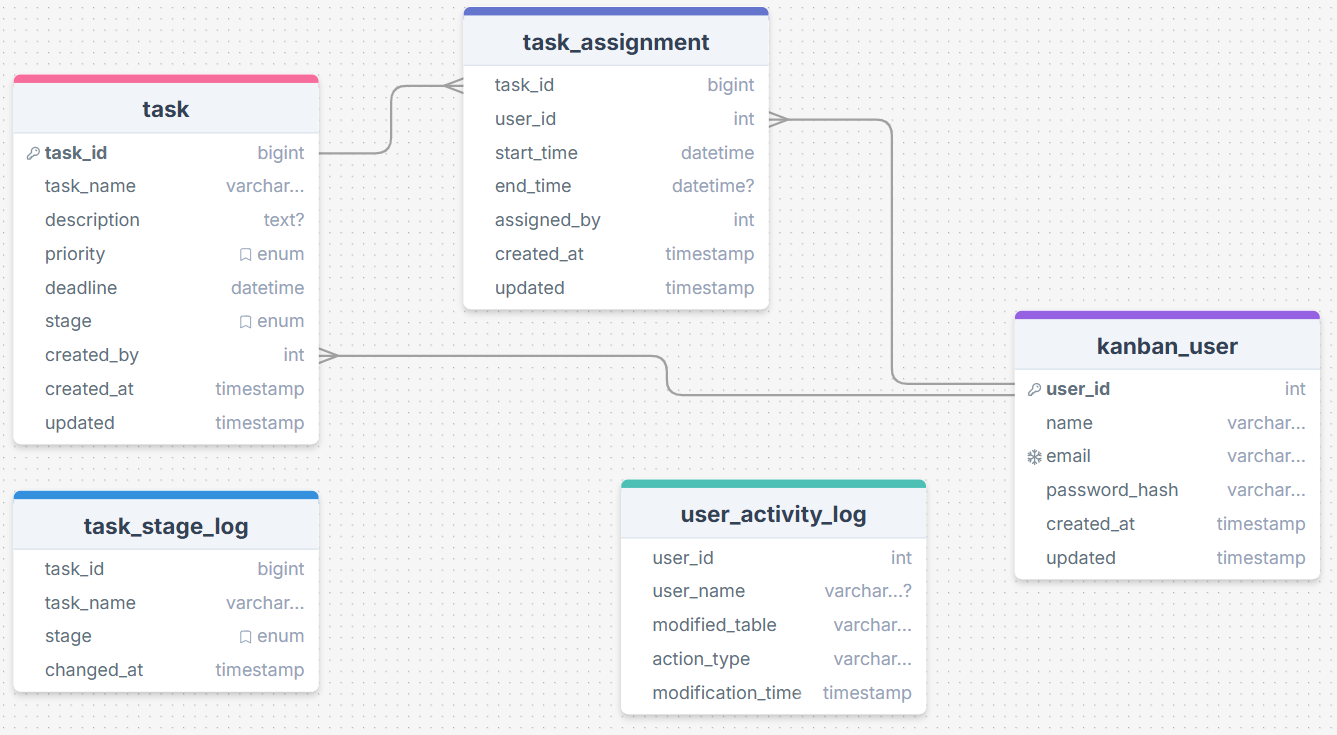
## **5.2 Component Details**

1. **Flask Application (Backend)**
   1. **Models**: Define database schemas for users, tasks and task history logs.
   2. **Views**: Business logic to handle API requests for authentication, task creation, updating, and displaying the board.
   3. **Templates**: Frontend HTML templates for displaying the Kanban board.
   4. **Authentication** **&** **Authorization**: Implements Flask-Login for user authentication, allowing different permissions for users and admins.
2. **MySQL** **Database**
   1. **Relational** **Database**: Stores data in structured tables, including kanban\_user, task, task\_assignment, user\_activity\_log and task\_stage\_log.
   2. **Data** **Backup**: Docker volumes ensure persistent storage across container restarts.
3. **Docker**
   1. **Containerization**: Flask and MySQL run in separate containers for modularity and scalability.
   2. **Persistent** **Storage**: Docker volumes ensure MySQL data persistence.
4. **Frontend (HTML/CSS/JavaScript)**
   1. **Flask** **Templates**: Dynamically generate HTML pages with Flask.
   2. **JavaScript**: Manages task interactions and visual effects, such as drag-and-drop for task status updates.

# **6. Database Design**

## **6.1 ER Diagram**

* **Users Table (kanban\_user)**: Stores user data (ID, name, email, hashed password, role).
* **Tasks Table (task)**: Manages task information (ID, title, description, due date, priority).
* **Assignments Table (task\_assignment)**: Assigns tasks to users (Task ID, User ID).
* **Task History Table (task\_stage\_log)**: Logs changes to tasks (Task ID, timestamp, status change, assigned user).
* **User History Table (user\_activity\_log)**: Logs changes by users.



## **6.2 Sample Schema**

CREATE TABLE kanban\_user

(

user\_id int UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name varchar(50) NOT NULL,

email varchar(50) NOT NULL,

password\_hash varchar(255) NOT NULL,

created\_at timestamp NOT NULL DEFAULT now(),

updated timestamp NOT NULL DEFAULT now(),

unique(email)

);

CREATE TABLE task

(

task\_id bigint UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

task\_name varchar(50) NOT NULL,

description text,

priority enum('Low', 'Medium', 'High') NOT NULL,

deadline datetime NOT NULL,

stage enum('To Do', 'In Progress', 'Done') NOT NULL,

created\_by int UNSIGNED NOT NULL,

created\_at timestamp NOT NULL DEFAULT now(),

updated timestamp NOT NULL DEFAULT now()

);

CREATE TABLE task\_assignment

(

task\_id bigint UNSIGNED NOT NULL,

user\_id int UNSIGNED NOT NULL,

start\_time datetime NOT NULL,

end\_time datetime,

assigned\_by int UNSIGNED NOT NULL,

created\_at timestamp NOT NULL DEFAULT now(),

updated timestamp NOT NULL DEFAULT now()

);

CREATE TABLE task\_stage\_log

(

task\_id bigint UNSIGNED NOT NULL,

task\_name varchar(50) NOT NULL,

stage enum('To Do','In Progress','Done') NOT NULL,

changed\_at timestamp NOT NULL DEFAULT now()

);

CREATE TABLE user\_activity\_log

(

user\_id int UNSIGNED NOT NULL,

user\_name varchar (50),

modified\_table varchar(50) NOT NULL,

action\_type varchar(50) NOT NULL,

modification\_time timestamp NOT NULL DEFAULT now()

);

ALTER TABLE task

ADD CONSTRAINT task\_created\_by\_user\_id\_fk FOREIGN KEY (created\_by) REFERENCES kanban\_user(user\_id) ON DELETE CASCADE;

ALTER TABLE task\_assignment

ADD CONSTRAINT task\_assignment\_task\_id\_fk FOREIGN KEY (task\_id) REFERENCES task(task\_id) ON DELETE CASCADE,

ADD CONSTRAINT task\_assignment\_user\_id\_fk FOREIGN KEY (user\_id) REFERENCES kanban\_user(user\_id) ON DELETE CASCADE,

ADD CONSTRAINT task\_assignment\_assigned\_by\_user\_id\_fk FOREIGN KEY (assigned\_by) REFERENCES kanban\_user(user\_id) ON DELETE CASCADE;

## **6.3 Triggers Schema**

delimiter //

CREATE PROCEDURE insert\_task\_stage\_log (IN task\_id bigint, IN task\_name varchar(50), IN stage varchar(15))

BEGIN

INSERT INTO task\_stage\_log (task\_id, task\_name, stage)

VALUES (task\_id, task\_name, stage);

END;

//

CREATE TRIGGER task\_update

BEFORE UPDATE ON task

FOR EACH ROW

BEGIN

SET NEW.updated = current\_timestamp();

END;

//

CREATE TRIGGER update\_task\_stage\_log

AFTER UPDATE ON task

FOR EACH ROW

BEGIN

CALL insert\_task\_stage\_log (NEW.task\_id, NEW.task\_name, NEW.stage);

END;

//

CREATE TRIGGER insert\_task\_stage\_log

AFTER INSERT ON task

FOR EACH ROW

BEGIN

CALL insert\_task\_stage\_log (NEW.task\_id, NEW.task\_name, NEW.stage);

END;

//

CREATE PROCEDURE insert\_user\_activity\_log (IN user\_id int, IN user\_name varchar(50), modified\_table varchar(50), action\_type varchar(50))

BEGIN

INSERT INTO user\_activity\_log (user\_id, user\_name, modified\_table, action\_type)

VALUES (user\_id, user\_name, modified\_table, action\_type);

END;

//

# **7. Deployment Architecture**

## **7.1 Docker Configuration**

**Dockerfile (Flask App)**

# Use an official Python runtime as a parent image

FROM python:3.9

# Set the working directory in the container

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Set environment variables

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

ENV FLASK\_RUN\_PORT=5000

# Expose port 5000 for Flask

EXPOSE 5000

# Run the Flask app

CMD ["flask", "run"]

**Docker Compose**

version: '3.8'

services:

web:

build: .

container\_name: flask\_app

ports:

- "5000:5000"

environment:

FLASK\_APP: app.py

FLASK\_RUN\_HOST: 0.0.0.0

FLASK\_RUN\_PORT: 5000

MYSQL\_DATABASE: kanban\_ms

MYSQL\_USER: root

MYSQL\_PASSWORD: root

MYSQL\_ROOT\_PASSWORD: root

depends\_on:

- db

db:

image: mysql:9.0

container\_name: mysql\_db

environment:

MYSQL\_DATABASE: kanban\_ms

MYSQL\_USER: root

MYSQL\_PASSWORD: root

MYSQL\_ROOT\_PASSWORD: root

ports:

- "3306:3306"

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

## **7.2 Deployment Steps**

1. **Build Containers**: docker-compose build to build the Docker images for Flask and MySQL.
2. **Run Containers**: docker-compose up to start the services.
3. **Initialize Database**: Run included DDL script to create tables.
4. **Access Application**: Access the app at <http://localhost:5000>.

# **8. Security Considerations**

* **Data Protection**: Use an SHA-256 hash for the passwords; randomly generate Flask’s SECRET\_KEY.

# **9. Monitoring and Logging**

* **Monitoring:** Implement monitoring for Docker containers (Prometheus/Grafana).
* **Logging:** Enable logging for Flask and MySQL for audit trails and troubleshooting.

# **10. Future Considerations**

* **Scalability:** Plan for container orchestration with Kubernetes.
* **Microservices:** Break down into smaller services if project needs grow, isolating components like history logging as separate services.