### Author

Srijan Shukla

21f1000671

21f1000671@student.onlinedegree.iitm.ac.in

Greetings,

I am a final year student at the National Rail and Transportation Institute, Vadodara. I have also been provisionally admitted into the degree level until the project submissions at IITM. This is my submission for the MAD II project, ‘The JanKanBan board. I hope it is of the required standards. Thank you.

### Description

JanKanBan, a comprehensive project management platform that facilitates the organisation and tracking of team tasks. It is constructed using the Flask web framework, the Vue JavaScript library, and SQLAlchemy. It enables users to create cards for individual tasks and move them through different stages of the workflow. Utilises asynchronous tasks using celery to maintain high levels of application performance and responsiveness and uses redis to improve performance.

### Technologies used

Authentication has been done by server side sessions where the cookies are cryptographically stored in the database.

Technologies and extensions used:

* Python
* Flask (web framework)
* Flask-Caching (extension for caching)
* Flask-Session (extension for managing sessions)
* Flask-RESTful (extension for building REST APIs)
* Flask-SQLAlchemy (extension for integrating with SQL databases)
* Flask-CORS (extension for handling Cross-Origin Resource Sharing)
* Celery (task queue)
* Redis (in-memory data store)
* Jinja2 (template engine)
* WeasyPrint (library for generating PDFs)
* UUID (library for generating universally unique identifiers)
* Flask-Login (extension for handling user authentication and authorization)

### DB Schema Design

The schema defines three database models: UserModel, ListModel, and CardModel.

The UserModel represents a user of the system, with the following fields:

* id: an integer primary key for the user
* name: a string representing the name of the user, must be unique and cannot be null
* password: a string representing the password of the user, cannot be null

The ListModel represents a list of cards, with the following fields:

* lid: an integer primary key for the list
* lname: a string representing the name of the list, must be unique and cannot be null
* user: an integer representing the user who owns the list, cannot be null
* ldescription: a string representing a description of the list, can be null

The CardModel represents a card in a list, with the following fields:

* cid: an integer primary key for the card
* cname: a string representing the name of the card, cannot be null
* lid: an integer representing the list that the card belongs to, cannot be null
* user: an integer representing the user who owns the card, cannot be null
* cdescription: a string representing a description of the card, cannot be null
* completed: a boolean indicating whether the card has been completed or not, cannot be null
* creation\_date: a DateTime representing the date and time when the card was created, default is the current date and time, cannot be null
* deadline\_date: a DateTime representing the deadline for completing the card, cannot be null
* completion\_date: a DateTime representing the date and time when the card was completed, can be null

This schema seemed the most sane in terms of implementing import jobs in the future.

### API Design

The Login resource has a method for logging in a user with a specified username and password.

The Register resource has a method for registering a new user with a specified username, password, and email address.

The Logout resource has a method for logging out a user.

The List resource has four methods for getting, adding, updating, and deleting lists. These methods may also operate on the cards belonging to each list.

The Card resource has four methods for getting, adding, deleting, and updating cards.

The CardPending resource has a method for marking a card as completed.

The Summary resource has a method for getting summary information about a user's lists and cards, which is cached for 20 seconds.

The CompDate resource has a method that retrieves a summary of completed cards for a particular user, including the completion date and the time taken to complete each card.

The ListExport resource has a method that exports a user's lists to a CSV file using a background Celery task, and returns the CSV file as a download.

The CardExport resource has a method that exports a user's cards to a CSV file using a background Celery task, and returns the CSV file as a download.

The ltocsv task exports a user's lists to a CSV file. It retrieves all the lists belonging to a particular user, and writes them to a CSV file with columns for the list ID, name, user, and description.

The ctocsv task exports a user's cards to a CSV file. It retrieves all the cards belonging to a particular user, and writes them to a CSV file with columns for the card ID, name, list ID, user, description, completion status, creation date, deadline date, and completion date.

Both tasks are called asynchronously.

There are asynchronously running scheduled tasks for daily reminders and monthly reminders too.

Vue CLI has been used as the frontend.

### Architecture and Features

Everything regarding the backend is in the ‘app.py’ file except the celery initialization and the flask-cache configs which are later imported into ‘app.py’.

The frontend files reside in the ‘FRONTEND’ folder which is in the root directory.

The application has an additional feature of ‘forgot password’ which aids a registered user by sending them their password in an email.

### Video

<https://drive.google.com/file/d/1GaW0RdhJpTNewCusWBoajkYLHOquYBsl/view?usp=sharing>