#### \*\*1. Importing Libraries and Setting Up the Blueprint\*\*

- \*\*`from flask import render\_template, request, Blueprint, jsonify`:\*\*

- `render\_template` is used to render HTML templates.

- `request` is used to handle incoming HTTP requests.

- `Blueprint` is used to organize routes and views for modularity.

- `jsonify` is used to return JSON responses.

- \*\*`from flask\_babel import \_`:\*\*

- The `\_` function is used for translation, enabling internationalization of text in the application.

- \*\*`from flask\_login import current\_user, login\_required`:\*\*

- `current\_user` represents the logged-in user and can be used to access user-specific data.

- `login\_required` is a decorator that restricts access to the route to only logged-in users.

- \*\*`from todoism.extensions import db`:\*\*

- `db` is used to interact with the database (e.g., creating, reading, updating, and deleting tasks).

- \*\*`from todoism.models import Item`:\*\*

- `Item` is the model that represents each to-do task in the database.

- \*\*`todo\_bp = Blueprint('todo', \_\_name\_\_)`:\*\*

- This creates a Blueprint for the `todo` module, allowing all routes for this module to be grouped under one namespace.

---

#### \*\*2. Home Page Route (`/app`)\*\*

- \*\*`@todo\_bp.route('/app')`:\*\*

- This decorator defines the route for the app’s main page. It is accessible via `/app` in the browser.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can access this route.

- \*\*`def app():`\*\*

- This function handles the logic for rendering the main app page.

- \*\*`all\_count = Item.query.with\_parent(current\_user).count()`:\*\*

- This query counts the total number of tasks assigned to the current user.

- \*\*`active\_count = Item.query.with\_parent(current\_user).filter\_by(done=False).count()`:\*\*

- This counts the tasks that are not completed (`done=False`).

- \*\*`completed\_count = Item.query.with\_parent(current\_user).filter\_by(done=True).count()`:\*\*

- This counts the tasks that are completed (`done=True`).

- \*\*`return render\_template('\_app.html', items=current\_user.items, all\_count=all\_count, active\_count=active\_count, completed\_count=completed\_count)`:\*\*

- This returns the HTML template for the main app page, passing the task counts and user-specific items to the template for rendering.

---

#### \*\*3. Edit Task Route (`/item/<item\_id>/edit`)\*\*

- \*\*`@todo\_bp.route('/item/<int:item\_id>/edit', methods=['PUT'])`:\*\*

- This decorator defines a route for editing an existing task. It listens for `PUT` requests at `/item/<item\_id>/edit`.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can edit tasks.

- \*\*`def edit\_item(item\_id):`\*\*

- This function handles the editing of an existing task with the specified `item\_id`.

- \*\*`item = Item.query.get\_or\_404(item\_id)`:\*\*

- This retrieves the task from the database based on the `item\_id`. If the task does not exist, it returns a 404 error.

- \*\*`if current\_user != item.author:`\*\*

- This checks if the logged-in user is the author (owner) of the task.

- \*\*`return jsonify(message=\_('Permission denied.')), 403`:\*\*

- If the logged-in user is not the author, it returns a 403 Forbidden error with a permission denied message.

- \*\*`data = request.get\_json()`:\*\*

- This retrieves the JSON data sent in the request body.

- \*\*`if data is None or data['body'].strip() == '':`\*\*

- This checks if the body of the task is empty or missing.

- \*\*`return jsonify(message=\_('Invalid item body.')), 400`:\*\*

- If the body is invalid, it returns a 400 Bad Request error with a message indicating the body is invalid.

- \*\*`item.body = data['body']`:\*\*

- This updates the task's `body` with the new value from the request.

- \*\*`db.session.commit()`:\*\*

- This commits the changes to the database.

- \*\*`return jsonify(message=\_('Item updated.'))`:\*\*

- This returns a success message indicating that the task has been successfully updated.

---

#### \*\*4. Toggle Task Status Route (`/item/<item\_id>/toggle`)\*\*

- \*\*`@todo\_bp.route('/item/<int:item\_id>/toggle', methods=['PATCH'])`:\*\*

- This decorator defines a route for toggling the status of a task. It listens for `PATCH` requests at `/item/<item\_id>/toggle`.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can toggle the task status.

- \*\*`def toggle\_item(item\_id):`\*\*

- This function handles the toggling of a task’s completion status.

- \*\*`item = Item.query.get\_or\_404(item\_id)`:\*\*

- This retrieves the task based on the `item\_id`. If the task does not exist, it returns a 404 error.

- \*\*`if current\_user != item.author:`\*\*

- This checks if the logged-in user is the task's author.

- \*\*`return jsonify(message=\_('Permission denied.')), 403`:\*\*

- If the logged-in user is not the author, it returns a 403 error with a permission denied message.

- \*\*`item.done = not item.done`:\*\*

- This toggles the `done` status of the task (from completed to not completed, or vice versa).

- \*\*`db.session.commit()`:\*\*

- This commits the changes to the database.

- \*\*`return jsonify(message=\_('Item toggled.'))`:\*\*

- This returns a success message indicating the task status has been toggled.

---

#### \*\*5. Delete Task Route (`/item/<item\_id>/delete`)\*\*

- \*\*`@todo\_bp.route('/item/<int:item\_id>/delete', methods=['DELETE'])`:\*\*

- This decorator defines a route for deleting a task. It listens for `DELETE` requests at `/item/<item\_id>/delete`.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can delete tasks.

- \*\*`def delete\_item(item\_id):`\*\*

- This function handles the deletion of a task.

- \*\*`item = Item.query.get\_or\_404(item\_id)`:\*\*

- This retrieves the task based on the `item\_id`. If the task does not exist, it returns a 404 error.

- \*\*`if current\_user != item.author:`\*\*

- This checks if the logged-in user is the author of the task.

- \*\*`return jsonify(message=\_('Permission denied.')), 403`:\*\*

- If the logged-in user is not the author, it returns a 403 error with a permission denied message.

- \*\*`db.session.delete(item)`:\*\*

- This deletes the task from the database.

- \*\*`db.session.commit()`:\*\*

- This commits the deletion to the database.

- \*\*`return jsonify(message=\_('Item deleted.'))`:\*\*

- This returns a success message indicating the task has been deleted.

---

#### \*\*6. Clear Completed Tasks Route (`/item/clear`)\*\*

- \*\*`@todo\_bp.route('/item/clear', methods=['DELETE'])`:\*\*

- This decorator defines a route for clearing all completed tasks. It listens for `DELETE` requests at `/item/clear`.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can clear tasks.

- \*\*`def clear\_items():`\*\*

- This function handles the clearing of all completed tasks for the logged-in user.

- \*\*`items = Item.query.with\_parent(current\_user).filter\_by(done=True).all()`:\*\*

- This retrieves all tasks that are completed (`done=True`) for the logged-in user.

- \*\*`for item in items:`\*\*

- This loops through each completed task.

- \*\*`db.session.delete(item)`:\*\*

- This deletes each completed task from the database.

- \*\*`db.session.commit()`:\*\*

- This commits the deletion of all tasks.

- \*\*`return jsonify(message=\_('All clear!'))`:\*\*

- This returns a success message indicating that all completed tasks have been cleared.

---

#### \*\*7. Add New Task Route (`/items`)\*\*

- \*\*`@todo\_bp.route('/items', methods=['POST'])`:\*\*

- This decorator defines a route for adding a new task. It listens for `POST` requests at `/items`.

- \*\*`@login\_required`:\*\*

- Ensures that only logged-in users can add new tasks.

- \*\*`def new\_item():`\*\*

- This function handles creating a new task.

- \*\*`data = request.get\_json()`:\*\*

- This retrieves the JSON data sent with the `POST` request.

- \*\*`body = data.get('body')`:\*\*

- This gets the task description (body) from the request.

- \*\*`category = data.get('category', 'General')`:\*\*

- This gets the category of the task, defaulting to `'General'` if not provided.

- \*\*`notes = data.get('notes', '')`:\*\*

- This gets any notes related to the task, defaulting to an empty string if not provided.

- \*\*`importance\_rank = data.get('

importance\_rank', 1)`:\*\*

- This gets the importance ranking of the task, defaulting to `1` if not provided.

- \*\*`new\_item = Item(body=body, category=category, notes=notes, importance\_rank=importance\_rank, author=current\_user)`:\*\*

- This creates a new task with the provided data.

- \*\*`db.session.add(new\_item)`:\*\*

- This adds the new task to the database.

- \*\*`db.session.commit()`:\*\*

- This commits the new task to the database.

- \*\*`return jsonify(message=\_('Item added.'))`:\*\*

---