Introduction to CRUD

What is CRUD?

- **C**reate, **R**ead, **U**pdate, and **D**elete are the four basic operations for managing data in a database.
- These operations allow users to add, retrieve, modify, and remove data effectively.

2: Setting Up the Flask Application

Why Set Up Flask?

- Flask is a lightweight web framework that helps build web applications efficiently.
- **Flask-SQLAlchemy** integrates Flask with a SQL-based database, simplifying database operations.

Steps:

Install Flask and Flask-SQLAlchemy using pip:
bash
Copy code

```
pip install flask flask-sqlalchemy
```

1.

Configure the app and database in `app.py`:
python
Copy code

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data.db'
db = SQLAlchemy(app)
```

2.

3: Creating the Database Model

Why Create a Database Model?

- The model defines the structure of your database (tables and their columns).
- It serves as a blueprint for storing data in the database.

Example:

In `models.py`, create a table `Item`:

python
Copy code
```
class Item(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # Unique ID
    name = db.Column(db.String(80), nullable=False)  # Name field
    description = db.Column(db.String(200), nullable=False)  #
Description field
```

Note: `db.create_all()` initializes the database and creates tables.

4: Understanding "C" - Create

Why Create?

- To allow users to add new data to the database, e.g., adding a new item to a product list.

Implementation:

Create a route to handle form submission:
python
Copy code
```
@app.route('/create', methods=['GET', 'POST'])
def create_item():
    if request.method == 'POST':
        name = request.form['name']
        description = request.form['description']
        new_item = Item(name=name, description=description)
        db.session.add(new_item)  # Add to the database
        db.session.commit()  # Save changes
        return redirect(url_for('list_items'))  # Redirect to the
list view
    return render_template('create.html')
```

-

Form: The `create.html` form collects data from the user.

Example in Action:

1. User fills out a form to add an item.
2. The `POST` method submits the form data.
3. Flask saves the data to the database.

5: Understanding "R" - Read

Why Read?

- To display or retrieve data stored in the database, such as showing all products or searching for a specific one.

Implementation:

Fetch all records using `Item.query.all()`:
python
Copy code

```
@app.route('/')
def list_items():
    items = Item.query.all()  # Fetch all records
    return render_template('index.html', items=items)
```

-

HTML Template: `index.html` displays data in a table format.

6: Understanding "U" - Update

Why Update?

- To allow users to modify existing data, such as correcting the name of an item or updating its description.

Implementation:

- Fetch the specific record using `Item.query.get_or_404(id)`.
- Use a form to submit the updated data.

Example:
python
Copy code

```
@app.route('/update/<int:id>', methods=['GET', 'POST'])
def update_item(id):
    item = Item.query.get_or_404(id)  # Fetch the item by ID
    if request.method == 'POST':
        item.name = request.form['name']  # Update the name
        item.description = request.form['description']  # Update the
description
        db.session.commit()  # Save changes
        return redirect(url_for('list_items'))
    return render_template('update.html', item=item)
```

Form: The `update.html` form pre-fills the current values for editing.

7: Understanding "D" - Delete

Why Delete?

- To allow users to remove unwanted data, such as deleting an item no longer needed.

Implementation:

- Fetch the specific record using `Item.query.get_or_404(id)`.
- Use the `db.session.delete()` method to remove it.

Example:
python
Copy code

```
@app.route('/delete/<int:id>', methods=['POST'])
def delete_item(id):
    item = Item.query.get_or_404(id)  # Fetch the item by ID
    db.session.delete(item)  # Delete the record
    db.session.commit()  # Save changes
    return redirect(url_for('list_items'))
```

Form: Add a delete button in `index.html` wrapped in a `POST` form for security.

8: Fetching Data

Fetching Data from the Database

- **All Records:** `Item.query.all()` retrieves all rows.
- **Specific Record:** `Item.query.get(id)` fetches a row by its ID.

Filtering Data: Use filters for custom queries, e.g.:
python
Copy code
`Item.query.filter_by(name='Laptop').all()`

-

9: Adding Frontend with HTML and CSS

Why Add Frontend?

- To make the application user-friendly and visually appealing.
- CSS styles enhance readability and layout.

Key Components:

1. **HTML Templates:** Use Jinja2 templates (`.html`) for dynamic rendering.
2. **CSS Styling:** Store styles in `static/styles.css` and link it in the HTML.

Example CSS (styles.css):
css
Copy code
```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    margin: 0;
    padding: 20px;
}
```

10: Testing and Running the Application

Steps to Run:

Run the Flask app:
bash
Copy code
```
python app.py
```

1.
2. Open the browser and go to `http://127.0.0.1:5000`.

Testing:

- Test **Create**: Add a new item.
- Test **Read**: Check the list view.
- Test **Update**: Modify an item.
- Test **Delete**: Remove an item.

Summary

CRUD operations are essential for managing data in web applications. Flask and Flask-SQLAlchemy simplify the process by providing tools for database interaction and web rendering. With this step-by-step guide, you now have a comprehensive understanding of building a CRUD application.