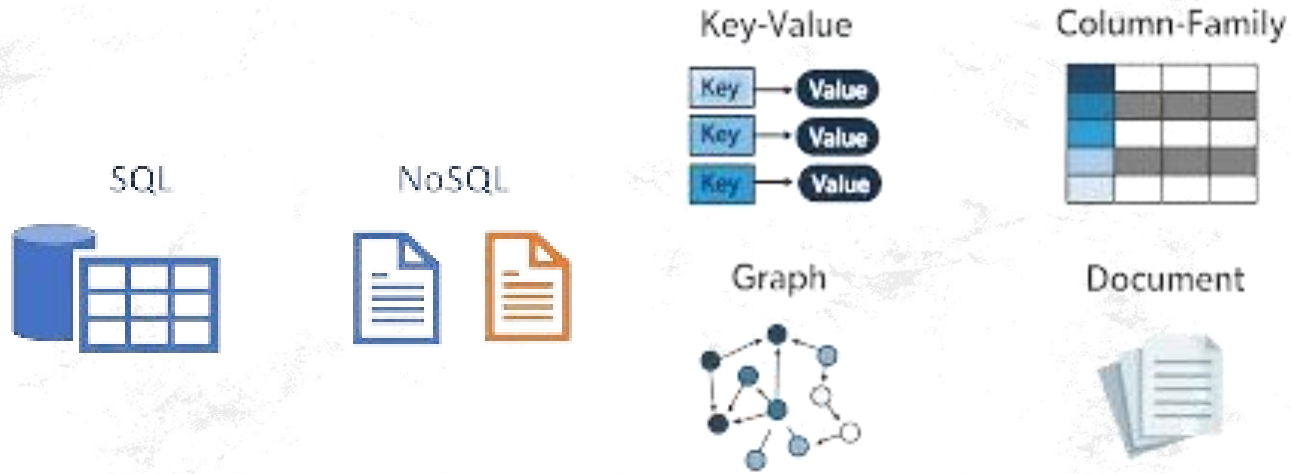# Introduction to **NoSQL**

NIELIT Chandigarh/Ropar

Agenda

- Introduction to NoSQL

- Types of NoSQL Databases

- MongoDB Overview

- MongoDB CRUD Operations

- MongoDB Aggregation Framework

- Practical Hands-on: CRUD & Aggregation in MongoDB on Glitch.com

- Creating a Todo (Notes) app using MongoDB on Glitch.com

Introduction to NoSQL

What is NoSQL?

- "NoSQL" stands for "Not Only SQL."
- A non-relational database management system designed to handle large amounts of data and diverse data models.

Why NoSQL?

- Scalability
- Flexibility in data modeling
- High performance for specific use cases
- Designed for distributed systems

Key Characteristics:

- Schema-less or flexible schema

- Horizontal scaling

- Optimized for modern data requirements (e.g., big data, real-time analytics)

Types of NoSQL Databases

1. **Document Databases**
   1. Example: MongoDB
   2. Data is stored as JSON-like documents.
   3. Best for semi-structured data.

2. **Key-Value Stores**
   1. Example: Redis, DynamoDB
   2. Data is stored as key-value pairs.
   3. Ideal for caching and session storage.

3. **Column-Oriented Databases**
   1. Example: Cassandra, HBase
   2. Data is stored in columns instead of rows.
   3. Suitable for analytical workloads.

4. **Graph Databases**
   1. Example: Neo4j, ArangoDB
   2. Data is represented as nodes and edges.
   3. Best for relationship-based queries (e.g., social networks).

Types of NoSQL Databases — Document Databases, Column-Oriented Database, Key-Value Store, Graph Database
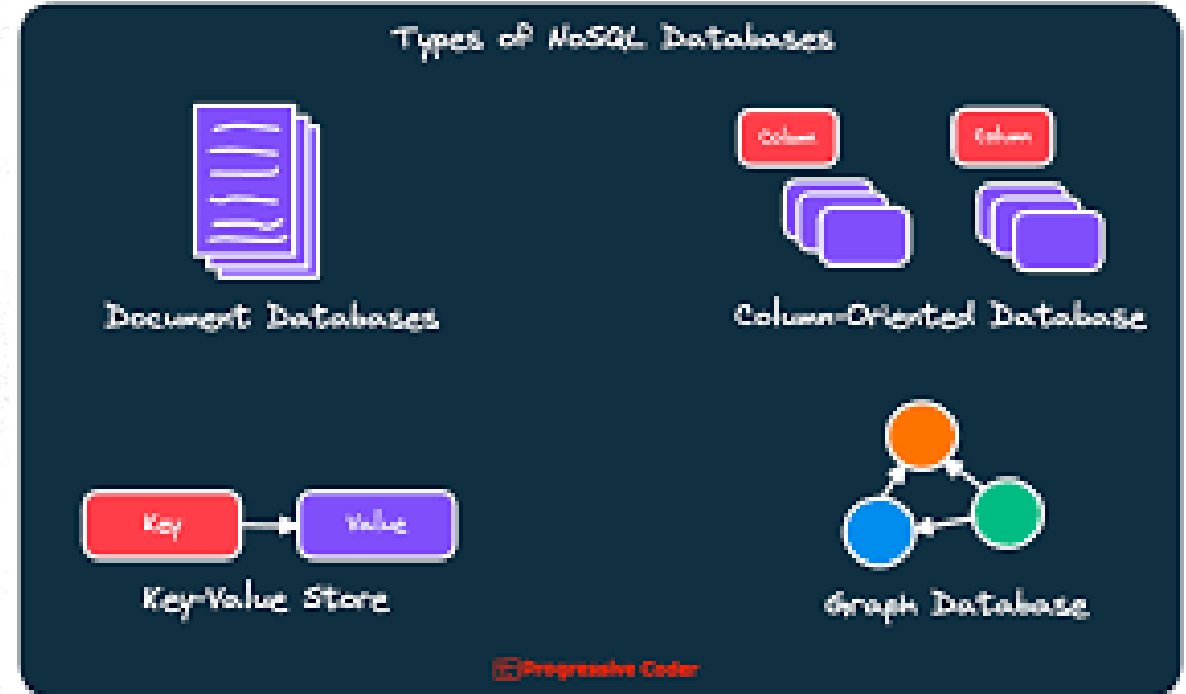
MongoDB Overview

MongoDB is a popular, open-source, document-oriented NoSQL database.

- Stores data in **JSON-like documents** (BSON).
- Supports **schema-less** data models.
- Provides **horizontal scaling** and high availability through sharding.
- Used by organizations such as Uber, eBay, and Netflix for large-scale applications.

Sharding is a type of horizontal scaling, but horizontal scaling can include other methods like replication and load balancing.

Key Features:

- **Flexible Schema**: Allows for a dynamic and flexible data model.
- **High Availability**: Replica sets for data redundancy and fault tolerance.
- **Sharding**: Distributes data across multiple machines for horizontal scaling.
- **Aggregation Framework**: Powerful data transformation and analysis tool.

MongoDB Overview

Features:
- Schema flexibility
- Horizontal scaling with sharding
- Built-in replication for high availability
- Rich query language

Use Cases:
- Content management
- Real-time analytics
- IoT applications
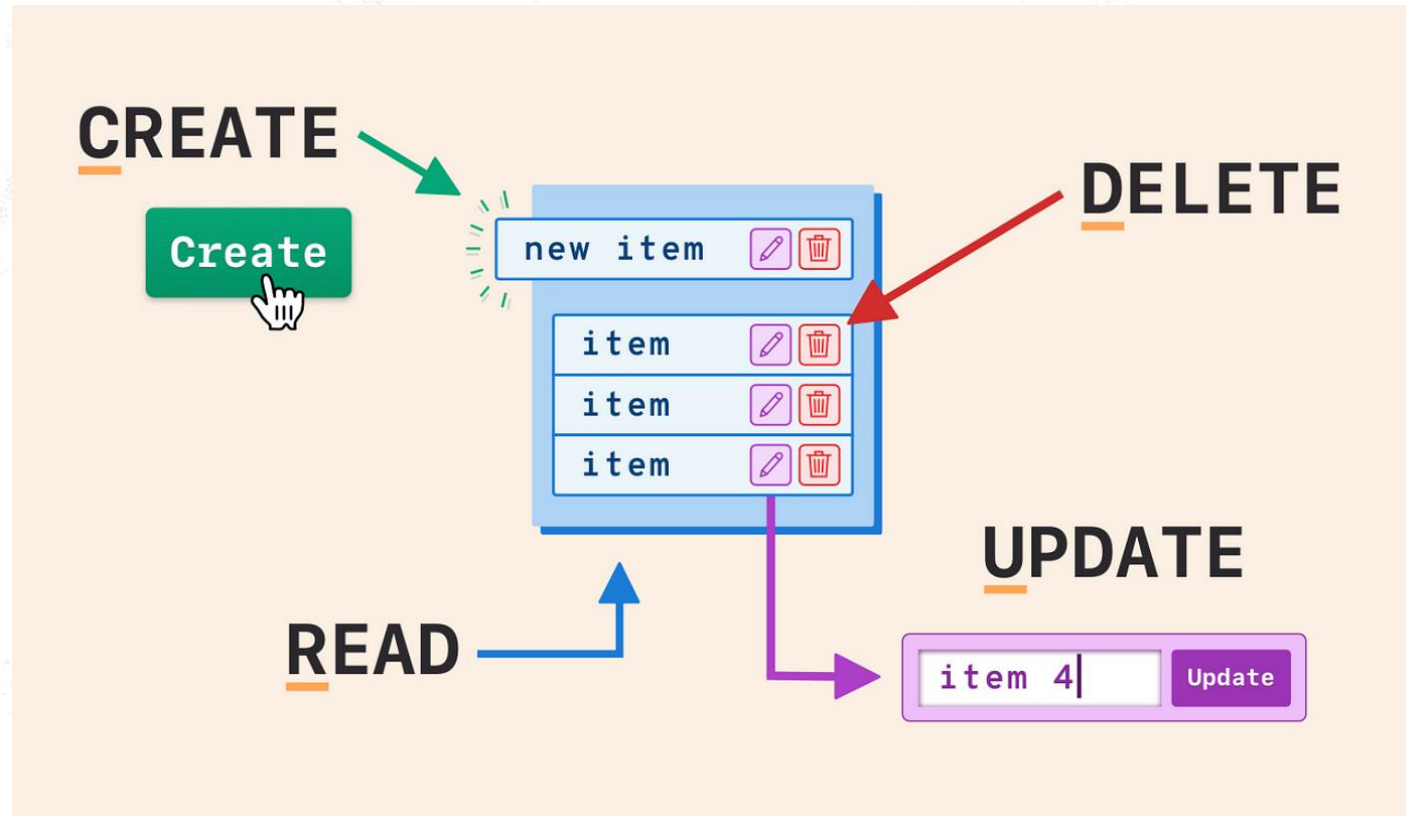- Mobile and web applications

Relational DB	MongoDB
database	database
table	collection
row	document
column	field

MongoDB CRUD Operations - Overview

- **CRUD** stands for Create, Read, Update, and Delete, the basic operations in MongoDB.

Create Operation (Insert)

- MongoDB provides methods to insert documents into collections.
  - **insertOne()**: Inserts a single document.
  - **insertMany()**: Inserts multiple documents.

1. Create:

```
db.collection.insertOne({ key: "value" });

db.collection.insertMany([{ key: "value1" }, { key: "value2" }]);
```

Example:

```
// Insert one document

db.collection.insertOne({

name: "John Doe",

age: 30,

email: "johndoe@example.com" });
```

```
// Insert documents

db.collection.insertMany([

{ name: "Alice", age: 25, email: "alice@example.com" },

{ name: "Bob", age: 35, email: "bob@example.com" }]);
```

Read Operation (Find)

- MongoDB provides the **find()** method to query the database.

```
db.users.find({ key: "value" });
db.users.findOne({ key: "value" });
```

Example:

```
// Find all documents in the collection
db.users.find({});
// Find specific document(s) based on a condition   GT – GREATER THAN
db.users.find({ age: { $gt: 30 } });
```

Projection: Specify fields to return:

```
db.users.find({}, { name: 1, email: 1 });
```

Update Operation (Update)

- MongoDB provides methods to modify existing documents.
  - **updateOne()**: Updates a single document.
  - **updateMany()**: Updates multiple documents.

```
db.users.updateOne({ key: "value" }, { $set: { key: "newValue" } });
db.users.updateMany({ key: "value" }, { $set: { key: "newValue" } });
```

Example:

```
// Update a single document
db.users.updateOne(
{ name: "John Doe" },
{ $set: { age: 31 } } );
```

```
// Update many documents
db.users.updateMany(
{ age: { $lt: 30 } },
{ $set: { status: "young" } } );
```

Delete Operation (Remove)

- MongoDB allows deletion of documents using the deleteOne() and deleteMany() methods.

```
db.users.deleteOne({ key: "value" });

db.users.deleteMany({ key: "value" });
```

Example:

```
// Delete a single document

db.users.deleteOne({ name: "John Doe" });

// Delete multiple documents

db.users.deleteMany({ age: { $lt: 30 } });
```

MongoDB Aggregation Framework

The **Aggregation Framework** is used for data transformation and analysis.

- Supports operations like filtering, grouping, sorting, and reshaping data.

Stages in the aggregation pipeline:

- **$match**: Filters the data.
- **$group**: Groups data by a specific field.
- **$sort**: Sorts data.
- **$project**: Shapes the output (selects specific fields).
- **$limit**: Limits the number of documents.

Aggregation Example - Grouping and Summarizing Data

Example: Grouping users by age and calculating the average age.

```
db.users.aggregate([
 { $group: { _id: "$age", avgAge: { $avg: "$age" } } }
]);
```

Explanation:

- $group: Groups documents by the field age and calculates the average age.

Aggregation Example - Sorting and Limiting

Example: Sorting users by age and limiting the results to the top 5.

```
db.users.aggregate([
{ $sort: { age: -1 } },
{ $limit: 5 }
]);
```

Explanation:

- $sort: Sorts the users by the age field in descending order.

- $limit: Limits the result to the top 5 documents.

MongoDB Aggregation Framework

What is Aggregation?
- A powerful way to perform data processing and analysis.

Stages:
- $match - Filters documents.
- $group - Groups documents by a specified key.
- $project - Shapes the output.
- $sort - Sorts documents.
- $limit - Limits the number of documents.

```
db.collection.aggregate([

{ $match: { status: "active" } },

{ $group: { _id: "$category", total: { $sum: 1 } } },

{ $sort: { total: -1 } }

]);
```

Benefits and Challenges of NoSQL

Benefits:

- High scalability
- Flexible data models
- Fast for specific use cases
- Optimized for big data

Challenges:

- Limited Support for Small Data
- Limited support for complex queries (compared to SQL), Lack of Join-Like Functionality
- Steeper Learning Curve for SQL Users
- Possible consistency trade-offs (e.g., CAP theorem)
- Backup and Restore Complexicity

MongoDB Use Cases

- **Content Management Systems**: Storing and managing unstructured content.

- **E-commerce Platforms**: Product catalogs, user data, shopping cart.

- **Real-Time Analytics**: Storing logs and event data for real-time processing.

- **Social Networks**: Storing and managing user profiles and relationships.

- Artificial intelligence, Edge computing, Internet of things, Mobile, Payments, Serverless development, and Personalization.

- MongoDB is a good choice for teams that need to develop software applications quickly and scale data.

Here are some companies that use MongoDB

- **Walmart**: Uses MongoDB to manage its product catalog

- **Google**: Uses MongoDB Atlas in combination with Google Cloud to build intelligent applications

- **Microsoft**: Integrates MongoDB Atlas with Microsoft Azure's AI and data analytics tools

- **IBM**: Customers can use MongoDB Community Edition and MongoDB Enterprise Edition as fully managed databases in the IBM Cloud

- **Zomato**: An Indian company that uses MongoDB

- **Tata Digital**: An Indian company that uses MongoDB

- **Canara HSBC Life Insurance**: An Indian company that uses MongoDB

- **Tata AIG**: An Indian company that uses MongoDB

- **Devnagri**: An Indian company that uses MongoDB

Conclusion

- **MongoDB** is a <mark>powerful</mark>, <mark>flexible</mark>, and <mark>scalable</mark> NoSQL database.

- **CRUD operations** provide basic functionality for creating, reading, updating, and deleting data.

- **Aggregation** enables complex data processing and transformation.

- MongoDB is ideal for applications that require <mark>high scalability</mark>, <mark>flexibility</mark>, and <mark>real-time data analysis</mark>.

Next Steps:

- Set up MongoDB locally or **on a cloud service like Atlas**.

- Practice CRUD operations and aggregation queries.
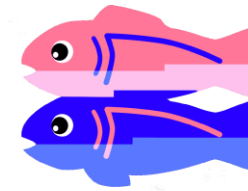
Create Live Project

NIELIT Chandigarh/Ropar

"Users love MongoDB because it offers the fastest time to value compared to any other DBMS technology".

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- **Set Up Glitch Project**:
  - Sign up or Login on Glitch.com

  - Go to Glitch.com homepage after logging in and click on **create a new project**.
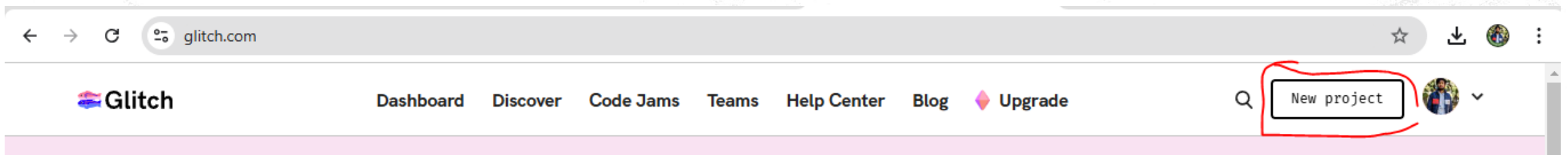
Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

○ Choose the "Hello World Website" template.
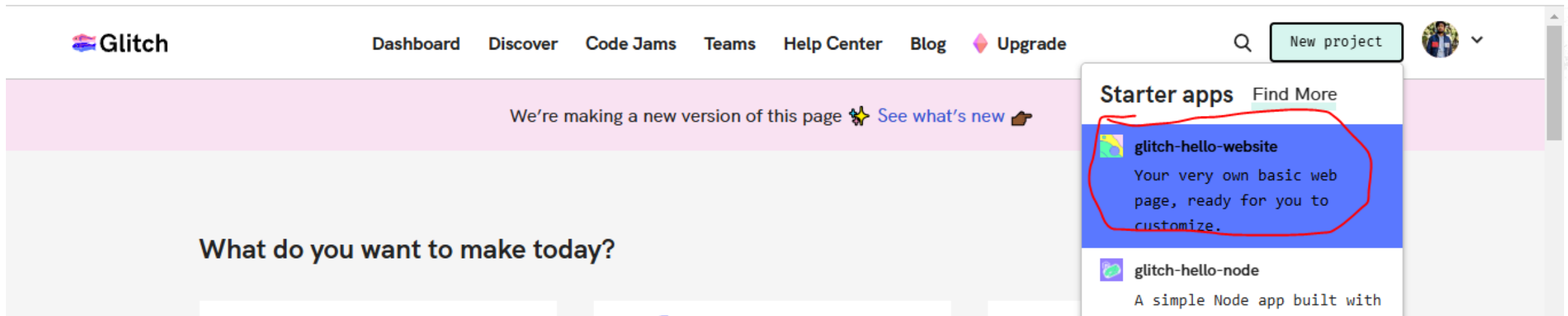
Delete all the existing files and Setup up website with Python – Flask by **creating all these 3 files listed in below screenshot** and in **start.sh** file write
`python app.py`

Create These Files:

1. `app.py`
2. `requirements.txt`
3. `start.sh`

groovy-charming-wombat «

🎌 Settings

📦 Assets

🗃 Files   +

app.py

requirements.txt

start.sh   ⋮

```
1  python app.py
```

Click on + sign to create a new file or folder

Click on these dots to **Delete** rename, duplicate a file
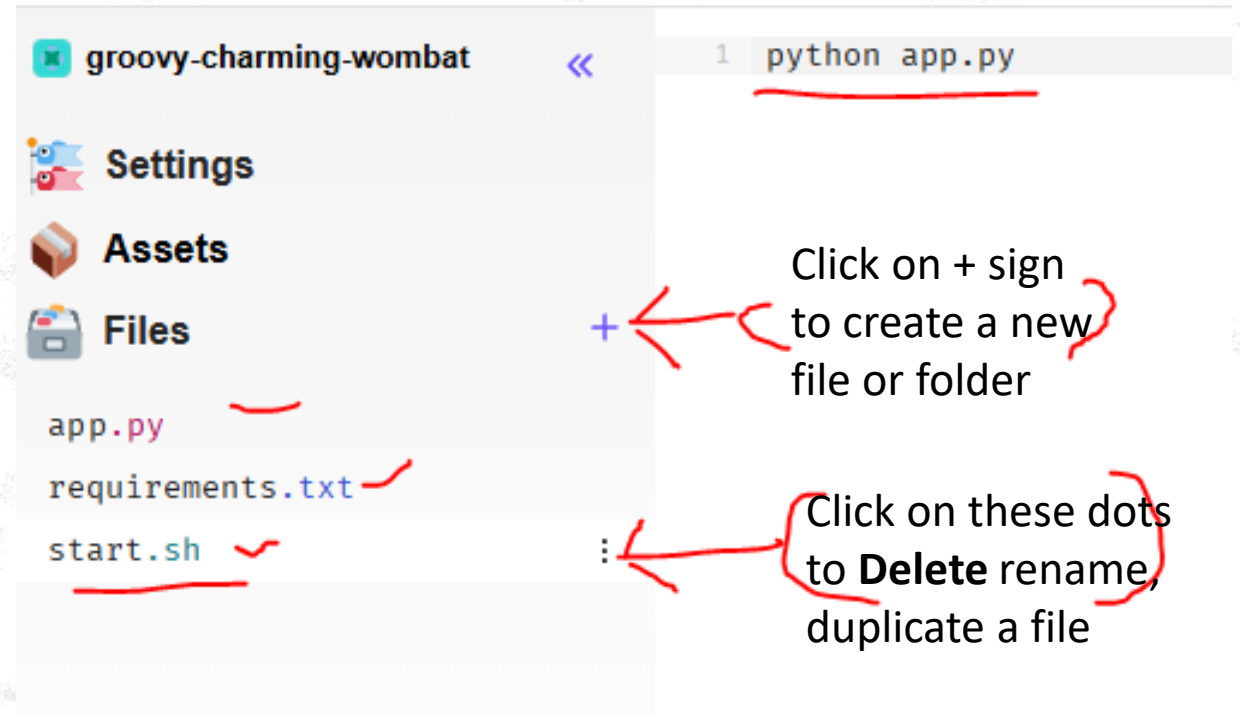
Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Click on Settings and then Edit Project Details to rename the project.
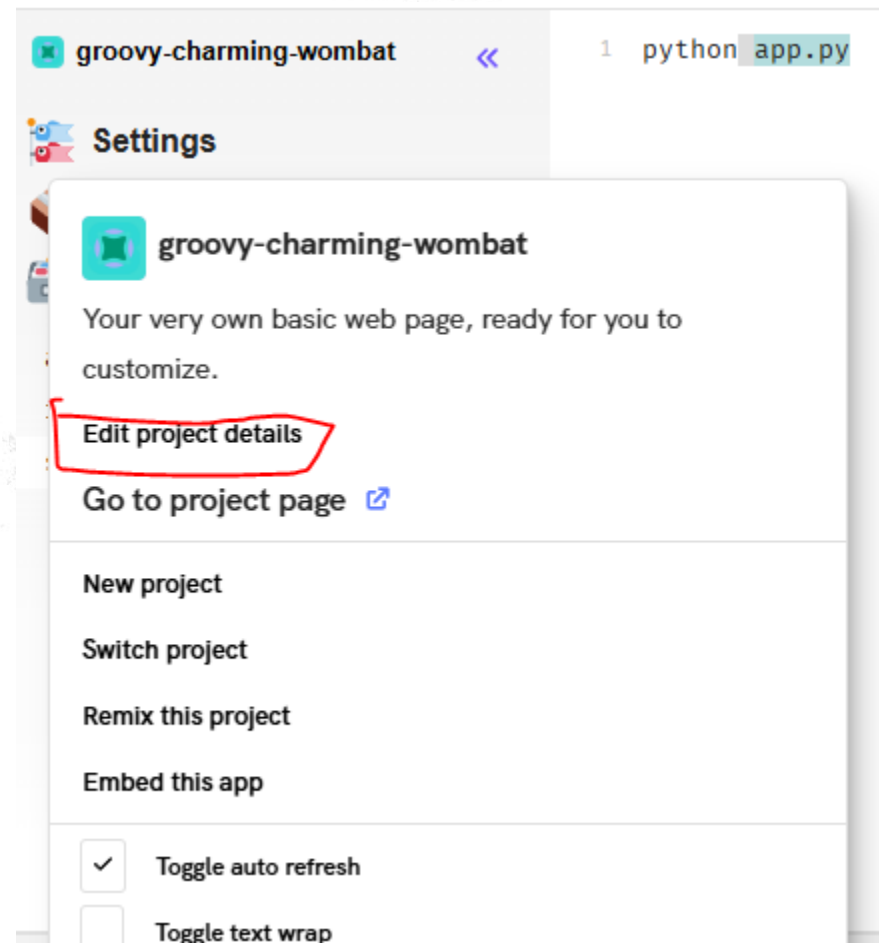
Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Add list of all required libraries in requirements.txt file.


```
1  flask
2  datetime
3  pymongo[srv]
4  Flask-PyMongo
5  pytz
```
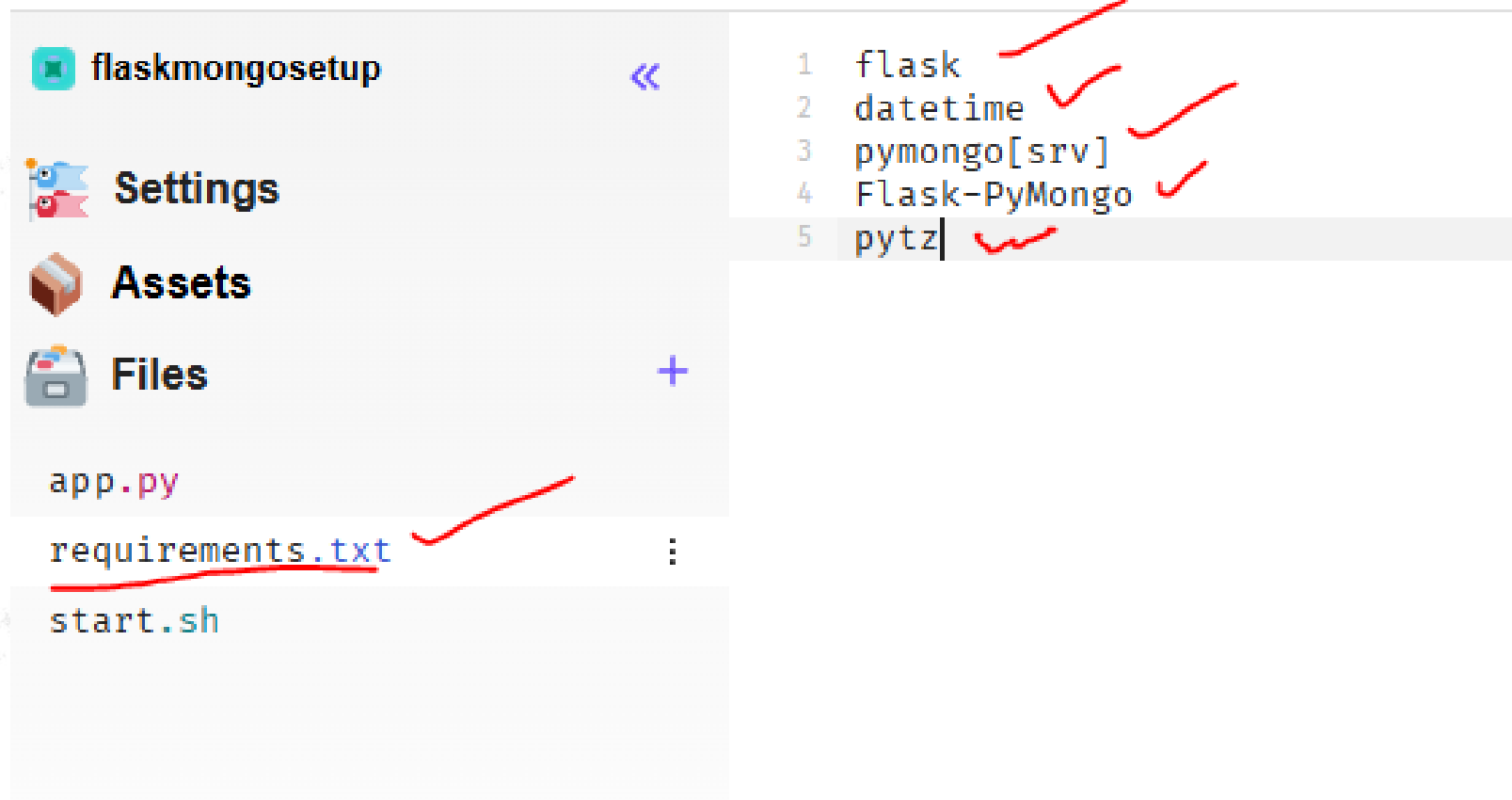
Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- To Setup Your Flask App edit app.py file import all the libraries we're going to use and then set up flask app structure:
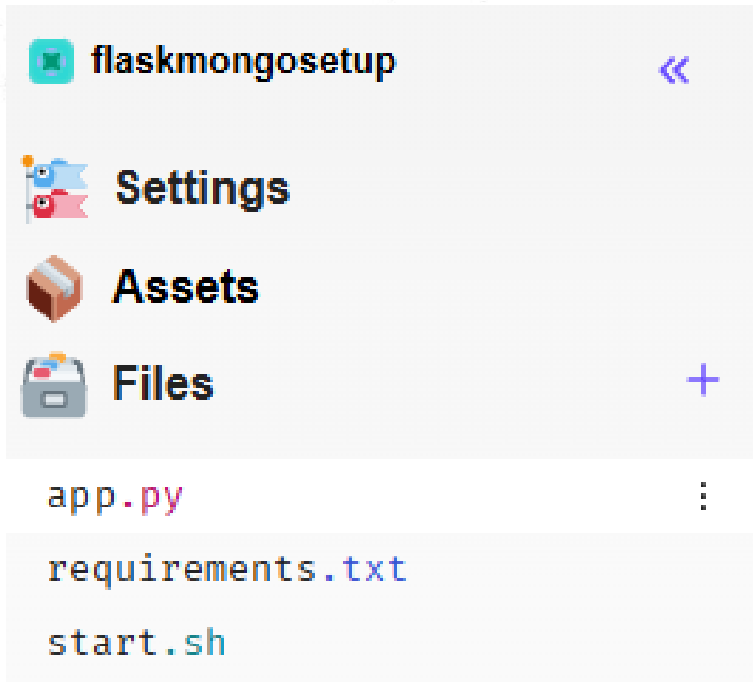
flaskmongosetup «

- Settings
- Assets
- Files +

app.py ⋮

requirements.txt

start.sh

```
1  from flask import Flask, request, redirect, url_for, render_template
2  from pymongo import MongoClient
3  from bson.objectid import ObjectId
4  from datetime import datetime
5  import pytz
6
7  app = Flask(__name__)
8
9
10
11 if __name__ == "__main__":
12     app.run(debug=True)
13
```

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- To Setup MonogoDB Altlas into your Flask App edit app.py file and add Mongodb Client and set Database name and Collection name
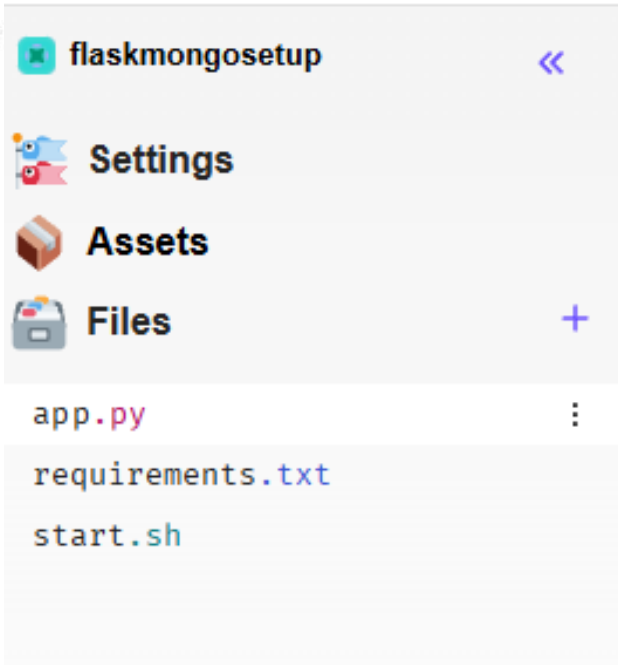
```python
from flask import Flask, request, redirect, url_for, render_template
from pymongo import MongoClient
from bson.objectid import ObjectId
from datetime import datetime
import pytz

app = Flask(__name__)

# MongoDB Connection
client = MongoClient("your-mongodb-connection-string")   # Replace with your MongoDB Atlas URI
db = client['todo_db']   # Set any Database name
tasks = db.tasks         # Set any Collection name


if __name__ == "__main__":
    app.run(debug=True)
```

Steps to get your MongoDB Atlas URL

- **1. Sign Up / Log In to MongoDB Atlas**

- Visit MongoDB Atlas.

- If you don't already have an account, sign up for a free account.

- Log in to your account.

- **2. Create a New Cluster**

- Once logged in, click **"Build a Cluster"** or **"Create"**.

- Select a **M0** free tier cluster
  - Set Cluster Name
  - Choose a cloud provider (e.g., AWS, Azure, GCP).
  - Select a region (choose one closer to your location for better performance).
  - Click **"Create Deployment"**.

Steps to get your MongoDB Atlas URL

- **Create a Database User**

- Create a username and password (e.g., admin / pass123police).

Don't add @ in your password it will cause error later on.

- Then Click on Create Database User.

- Then click on Choose connection method.

Steps to get your MongoDB Atlas URL

- **Create a Database User**

- Create a username and password (e.g., admin / pass123police).

- Then click on Choose connection method.

ℹ️ You'll need your database user's credentials in the next step. Copy the database user password.

Username

admin

Password

pass123police   HIDE   📋 Copy

Create Database User

Close   Choose a connection method

Steps to get your MongoDB Atlas URL

- **Click on Drivers:**

- **You'll See your connection string copy it and save it somewhere safe:**

Steps to get your MongoDB Atlas URL

- **Allow Network Access**

- Go to **"Network Access"** in the left sidebar.

- Click **"Add IP Address"**.

  +ADD IP ADDRESS

  - Choose **"Allow Access from Anywhere"** (recommended for development only). This adds 0.0.0.0/0 to allow access from any IP address.

    ALLOW ACCESS FROM ANYWHERE

  - Alternatively, add specific IPs if you know your development machine's public IP.

- Save changes by clicking Confirm.

Atlas   creativ

Project 0

Clusters

SERVICES
Atlas Search
Stream Processing
Triggers
Migration
Data Federation

SECURITY
Quickstart
Backup
Database Access
Network Access
Advanced

Access List Entry: 0.0.0.0/0

Comment: Optional comment describing this entry
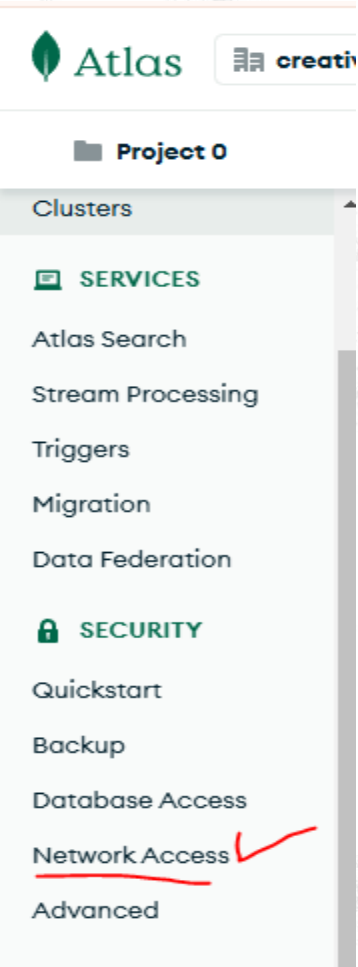
This entry is temporary and will be deleted in 6 hours

Cancel   Confirm

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- **Add your Connection URL in your flask app edit app.py file**

```
flaskmongosetup          «

Settings

Assets

Files                     +

app.py
                          ⋮
requirements.txt

start.sh
```

```python
1  from flask import Flask, request, redirect, url_for, render_template
2  from pymongo import MongoClient
3  from bson.objectid import ObjectId
4  from datetime import datetime
5  import pytz
6
7  app = Flask(__name__)
8
9  # MongoDB Connection
10 client = MongoClient("mongodb+srv://admin:pass123police@cluster0.h9cz2.mongodb.net/?retryWrites=true&w=majority")
11 db = client['todo_pp']     # Set any Database name
12 tasks = db.tasks           # Set any Collection name
13
14
15 if __name__ == "__main__":
16     app.run(debug=True)
17
```

`html`

- **Create templates folder and inside it create index.html file click on + sign and then enter `templates/index.html` it will create templates folder and inside it index.html file**

```
flaskmongosetup          «

templates/index.html

Add This File

To add a file inside a folder, include
a forward slash in the file name (e.g.
folder/my-awesome-file.js)
```

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Create Home route, Add task route and Delete task route in your app.py file

flaskmongosetup

Settings

Assets

Files

⌄ templates/
 index.html

app.py

requirements.txt

start.sh

```python
# MongoDB Connection
client = MongoClient("mongodb+srv://admin:pass123police@cluster0.h9cz2.mongodb.net/?retryWrites=true&w=majority")
db = client['todo_pp']   # Set any Database name
tasks = db.tasks          # Set any Collection name

# Home Route - Show all tasks
@app.route("/")
def home():
    ist = pytz.timezone("Asia/Kolkata")   # India Standard Time timezone
    current_time = datetime.now(ist).strftime("%Y-%m-%d %H:%M:%S")
    all_tasks = list(tasks.find())   # Fetch all tasks
    return render_template("index.html", current_time=current_time, tasks=all_tasks)   # Pass tasks and datetime to HTML

# Add a Task
@app.route("/add", methods=["POST"])
def add_task():
    task_text = request.form.get("task")   # Get form data
    if task_text:
        tasks.insert_one({"task": task_text.strip()})   # Add task to MongoDB
    return redirect(url_for("home"))

# Delete a Task
@app.route("/delete/<task_id>")
def delete_task(task_id):
    tasks.delete_one({"_id": ObjectId(task_id)})   # Delete by ObjectId
    return redirect(url_for("home"))


if __name__ == "__main__":
    app.run(debug=True)
```

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Edit index.html file add following code:


```html
<html>
  <head>
    <title>Simple To-Do App</title>
  </head>
  <body>
    <!-- Display Current Time -->
    <h1>Simple To-Do App {{ current_time }}</h1>

    <!-- Form to Add a Task -->
    <form action="/add" method="POST">
      <input type="text" name="task" placeholder="Enter a task" required />
      <button type="submit">Add Task</button>
    </form>

    <!-- Display Tasks -->
    <h2>Tasks:</h2>
    {% if tasks|length > 0 %}
    <ul>
      {% for task in tasks %}
      <li>
        {{ task.task }}
        <a href="/delete/{{ task._id }}">[Delete]</a>
      </li>
      {% endfor %}
    </ul>
    {% else %}
    <p>No tasks added.</p>
    {% endif %}
  </body>
</html>
```

Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Let's View our app Click preview open in new window:
- You can also see your app by visiting yourappname.glitch.me

Open preview pane
Preview in a new window ✓
TOOLS   PREVIEW

← → C   flaskmongosetup.glitch.me

Simple To-Do App 2025-01-08 02:43:48

Enter a task   Add Task

Tasks:

No tasks added.
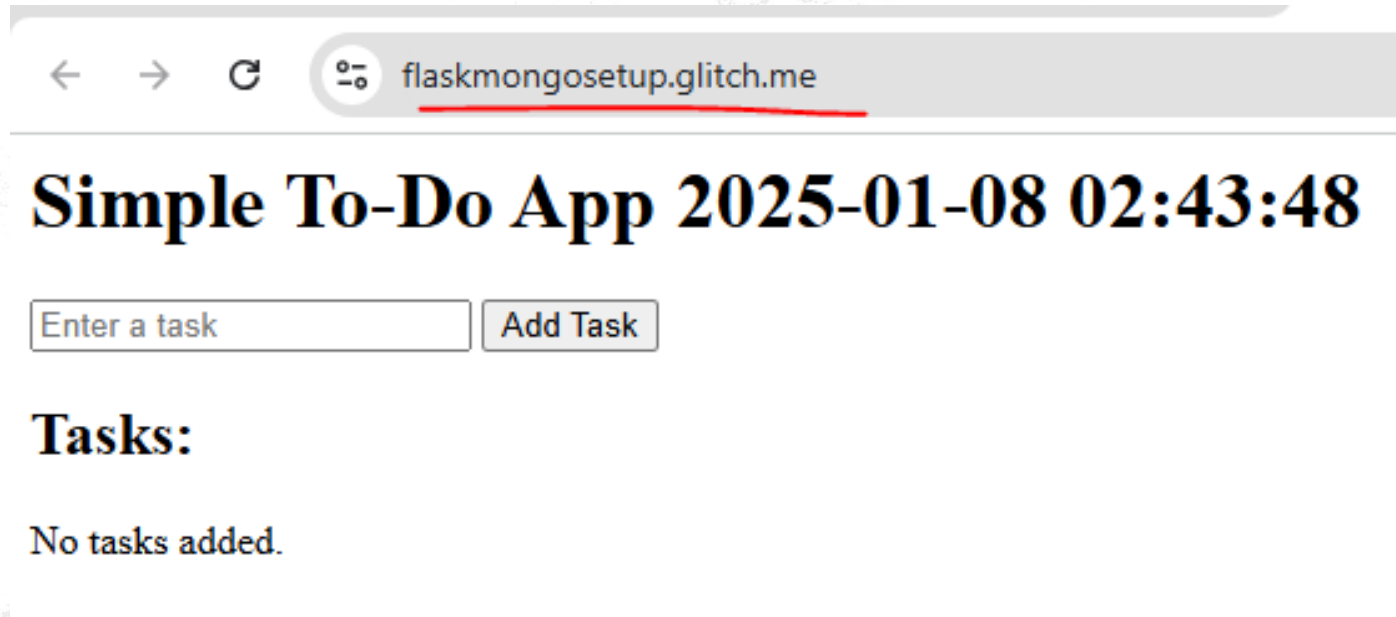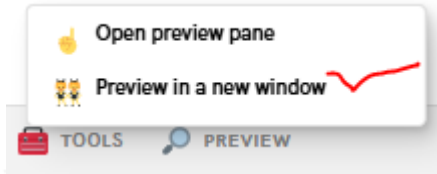
Create a ToDO App using MongoDB and Python Flask Framework On Glitch.com

- Let's Add Some Tasks to test our newly created app.
- Enter Task and Click on Add Task.
- Wow as we can see your app is working Fine.
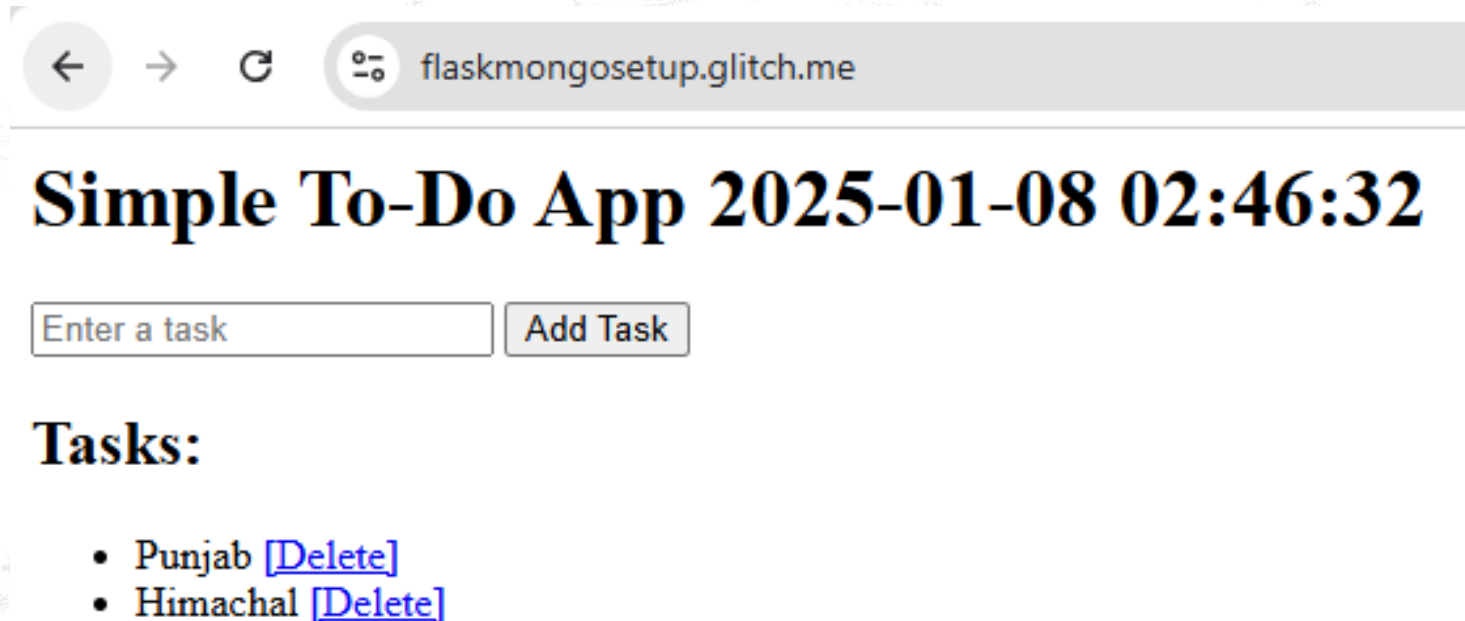
See Collection on MongoDB Atlas

To See your Collection on MongoDB Atlas

- Click on Clusters.

- Then Click on Your Cluster Name.

- Then Click on Collections.

See Collection on MongoDB Atlas

Click on your DB name and then your Collection Name to view all your Collections.

Enjoy your own Personal ToDo App

Simple To-Do App 2025-01-08 02:46:32

Enter a task | Add Task

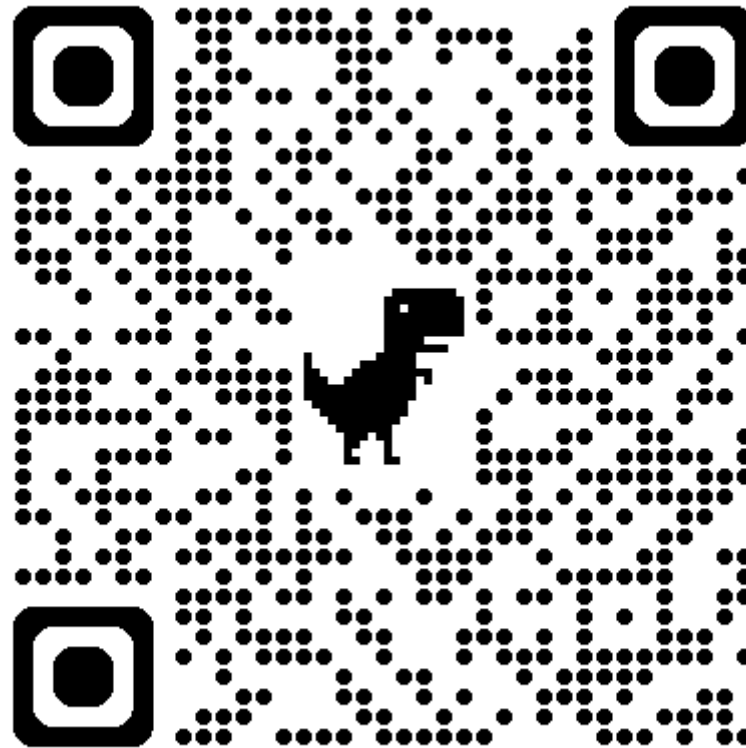**Tasks:**

- Punjab [Delete]
- Himachal [Delete]

What's Next:

- Add CSS Styling to improve App appearance.

- Add BootStarp to make it Mobile Fiendly.

- Add Advance Features like aggregation and make your app like https://flaskmongo.glitch.me/ this improved version of app with better Styling (UI) and MongoDB aggregation.

- Keep Learning Try Creating Similar apps using MongoDB and Flask on Glitch.

Live Demo

https://flaskmongosetup.glitch.me/