# Databases

# Databases

- Software that stores data on disk
- Runs as a server and is communicated with via TCP sockets
- Provides an API to store/retrieve data
  - The software handles the low-level file IO
  - Allows us to think about our data, not how to store it
- Provides many optimizations

# Databases

- We'll look at 2 different databases
- Both are pieces of software that must be downloaded, installed, ran, then connected to via TCP

- MongoDB
  - An unstructured server based on document stores

- mySQL
  - A server implementing SQL (Structured Query Language)

# MongoDB

- Runs on port 27017 (By default)
- A document-based database
- Stores data in a structure very similar to JSON
- In python/JS
  - Insert dictionaries/objects directly
- Each object is stored in a collection

# MongoDB - Connection

- Download a connection library and use to establish a connection with MongoDB
- MongoDB is separated into several layers
  - Databases - Named by Strings; Contains collections
  - Collections - Where the data is stored; similar to a SQL table
- Access your collections to insert/retrieve/update/delete data

```
from pymongo import MongoClient

mongo_client = MongoClient("localhost")
db = mongo_client["cse312"]
chat_collection = db["chat"]
```

# MongoDB - Insert Data

- Insert dictionaries/objects directly
- For languages without a data structure comparable to dictionaries/objects
  - More work to do to prepare your data for Mongo

```
chat_collection.insert_one({"username": "hartloff", "message": "hello"})
```

# MongoDB - Security

- No Mongo injection attacks
- Mongo does not rely on parsing statements as strings
- Any injected code would be treated as values

```
chat_collection.insert_one({"username": "hartloff", "message": "hello"})
```

# MongoDB - Retrieve Data

- Retrieve documents using find
- Find takes a key-value store and returns all documents with those values stored at the given keys
  - Ex. {"username": "hartloff"} returns all documents with a username of "hartloff"
- To retrieve all documents, use an empty key-value store {}

```
my_data = chat_collection.find({"username": "hartloff"})
all_data = chat_collection.find({})
```

# MySQL

- Listens for TCP connections on port 3306 (By default)
- Install a library for your language that will connect to the MySQL server
- SQL is based on tables with rows and column
  - Similar in structure to CSV except the values have types other than string

# MySQL - Insert Data

- If using inputs from the user, always use prepared statements

```
val statement = connection.prepareStatement("INSERT INTO players VALUE (?, ?)")

statement.setString(1, "mario")
statement.setInt(2, 10)

statement.execute()
```

# MySQL - Security

- Not using prepared statements?
  - Vulnerable to SQL injection attacks
- If you concatenate user inputs directly into your SQL statements
  - Attacker chooses a username of "';DROP TABLE players;"
  - You lose all your data
  - Even worse, they find a way to access the entire database and steal other users' data
  - SQL Injection is the most common successful attack on servers

# MongoDB vs. SQL

- MongoDB is unstructured
  - Can add objects in any format to a collection
  - Can mix formats in a single collection
    - Ie. In a single collection the documents can have different attributes
- SQL is structured (That's what the S stands for)
  - Table columns must be pre-defined
    - All rows have the same attributes
    - Adding a column can be difficult
  - Fast!

# MongoDB vs. SQL

- Hot Take
  - MongoDB is best for prototyping when the structure of your data is constantly changing
    - Take advantage of the flexibility
  - SQL is best once your data has a defined structure
    - Take advantage of the efficiency

# MongoDB vs. SQL

- For the HW
  - MongoDB is required

- For the project
  - Use any DB you choose

# Running MongoDB

- Choose 1:

- Download and install MongoDB and run it on your device

- Run MongoDB using Docker (recommended)

# Running MongoDB

- Download and install MongoDB and run it on your device

- Go to Mongo’s website and follow the instructions
- Mongo will run locally on your machine and listen for connections on port 27017
- Connect using the host “localhost”

# Running MongoDB

- Run MongoDB using Docker (recommended)

- Install docker
- Run the command:
  - docker run -d -p 27017:27017 mongo:latest
- This uses docker to run mongo in a container

# Running MongoDB

- Run MongoDB using Docker (recommended)

- docker run -d -p 27017:27017 mongo:latest
  - Run a container using the mongo:latest image
  - Map localhost port 27017 to port 27017 inside the container
- Now you can connect to the DB on “localhost”

# Running MongoDB

- Run MongoDB using Docker compose

- Alternatively, use the provided docker compose file from the HW handout
- docker compose -f docker-compose.db-only.yml up -d
- Same effect as the previous docker command

# Running MongoDB

- Run MongoDB using Docker compose

- Last option: run the full app using docker compose using the docker-compose file from the HW handout
- docker compose up —build —force-recreate
  - Runs the database and your app
  - Rebuilds your app if you made changes
  - Connects to the database on host “mongo”

# Running MongoDB

- However you choose to run your database:
  - Use the setup in database.py of the HW handout to connect to your DB
  - Dynamically connects to either “localhost” of “mongo” depending on how it was started
  - Let’s you not have to think about the DB host that much

# Running Your App

- If you use any “localhost” method to run your DB:
  - Run your app by running server.py

# HTML Templates

# HTML Templates

- Instead of writing complete HTML files
  - Write HTML templates
- An HTML template is an "incomplete" HTML file that is used to generate complete pages
- Use additional markup to add placeholders in the HTML
- Replace the placeholders with data at runtime

# HTML Templates

- Example template with 3 placeholders
- The title, description, and image_filename will be replaced later
  - Provide values for these 3 placeholders to serve a response

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>This page was generated from a template</title>
</head>
<body>

<h1>{{title}}</h1>

<p>{{description}}</p>

<img src="{{image_filename}}"/>

</body>
</html>
```

# HTML Templates

- To substitute the placeholders
  - Use any string manipulation that gets the job done
  - Find/replace is the simplest solution

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>This page was generated from a template</title>
</head>
<body>

<h1>{{title}}</h1>

<p>{{description}}</p>

<img src="{{image_filename}}"/>

</body>
</html>
```

# HTML Templates

- On the HW:
  - layout.html contains all the HTML that is shown on every page (navigation, structur, etc)
  - layout.html has one placeholder - {{content}} - which is where you'll insert all the HTML for the specific page that was requested
  - To render a page, eg. index.html, replace {{content}} with everything read from index.html
  - Send the resulting rendered page to the client

# Common Template Features

- Loops
- To add loops to your templates
  - Choose syntax for the start and end of the loop

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>This page was generated from a template</title>
</head>
<body>

{{loop}}
<h6>{{content_from_data_structure}}</h6>
{{end_loop}}

</body>
</html>
```

# Common Template Features

- Conditionals
- Can use similar approach as loops
- Choose syntax for the start and end of each block in the conditional

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>This page was generated from a template</title>
</head>
<body>

{{if cookie_set}}
<h6>Welcome back!</h6>
{{else}}
<h6>Welcome!</h6>
{{end_if}}

</body>
</html>
```