

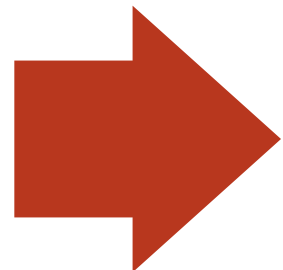
Web Programming

Server-side programming III.

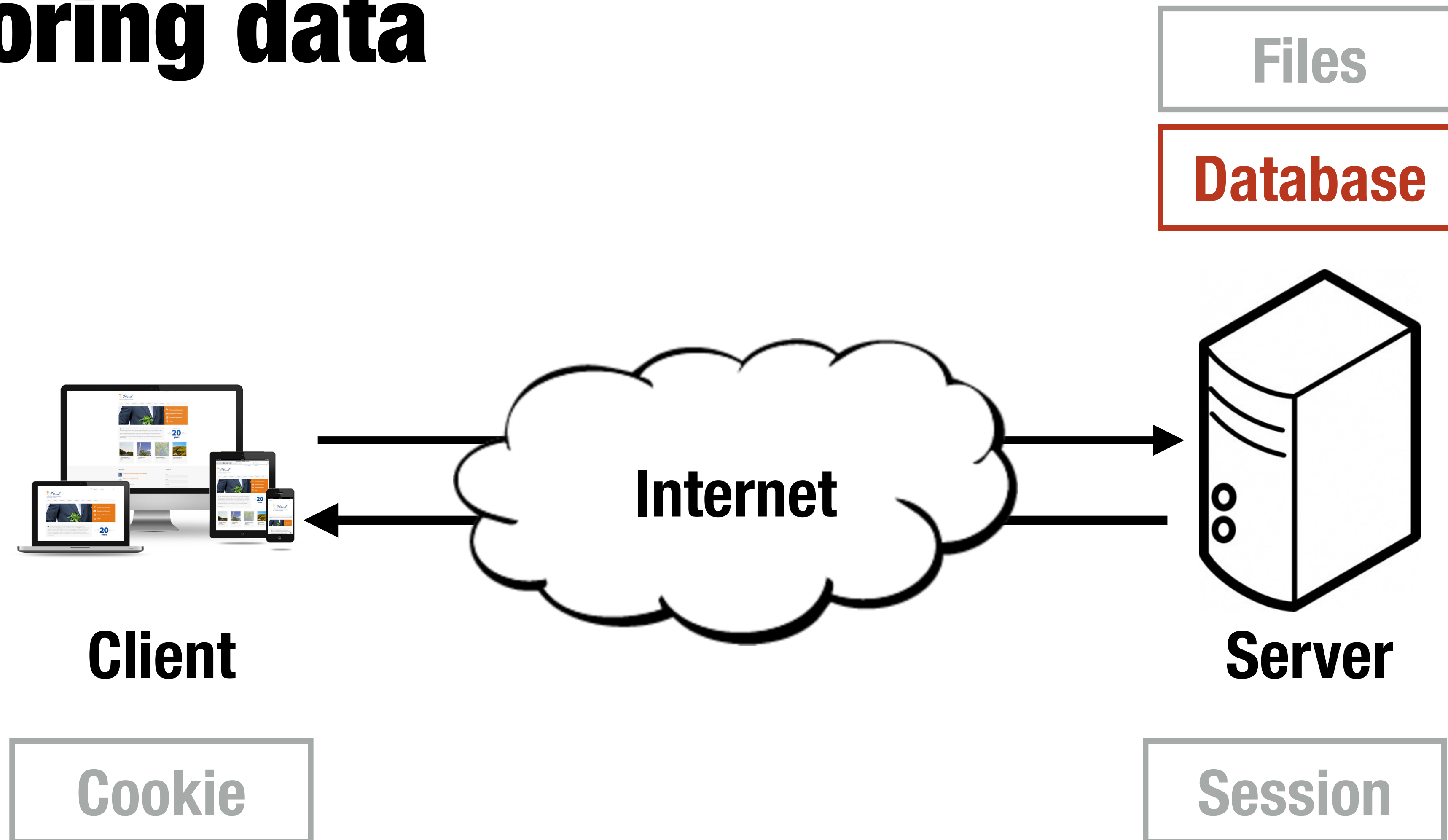
Robert Ewald | University of Stavanger

Server-side programming

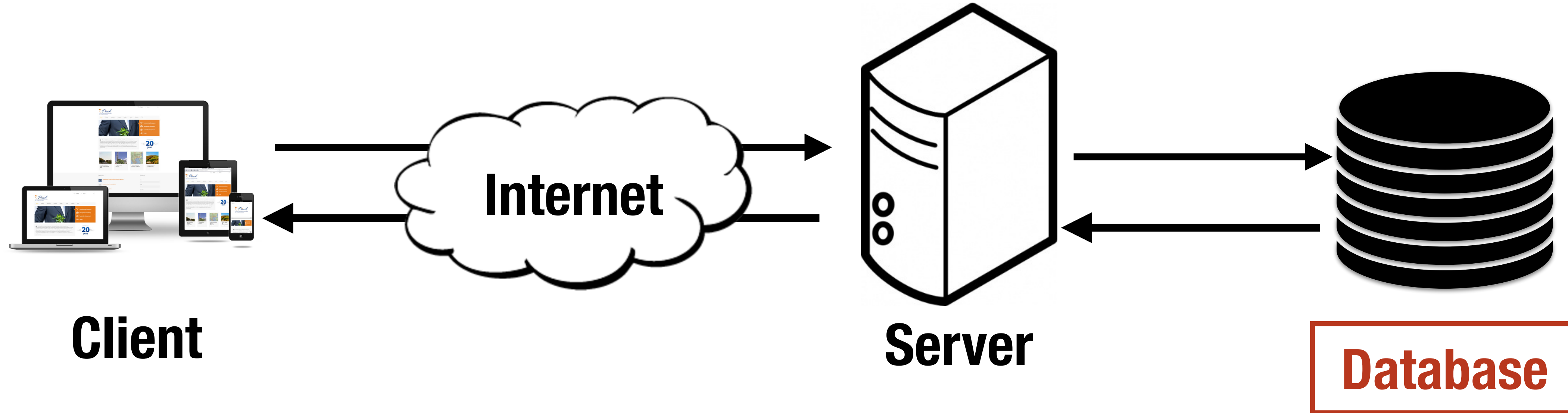
- Part I. handling requests
- Part II. templating
- Part III. SQLite
- Part IV. cookies and sessions



Storing data

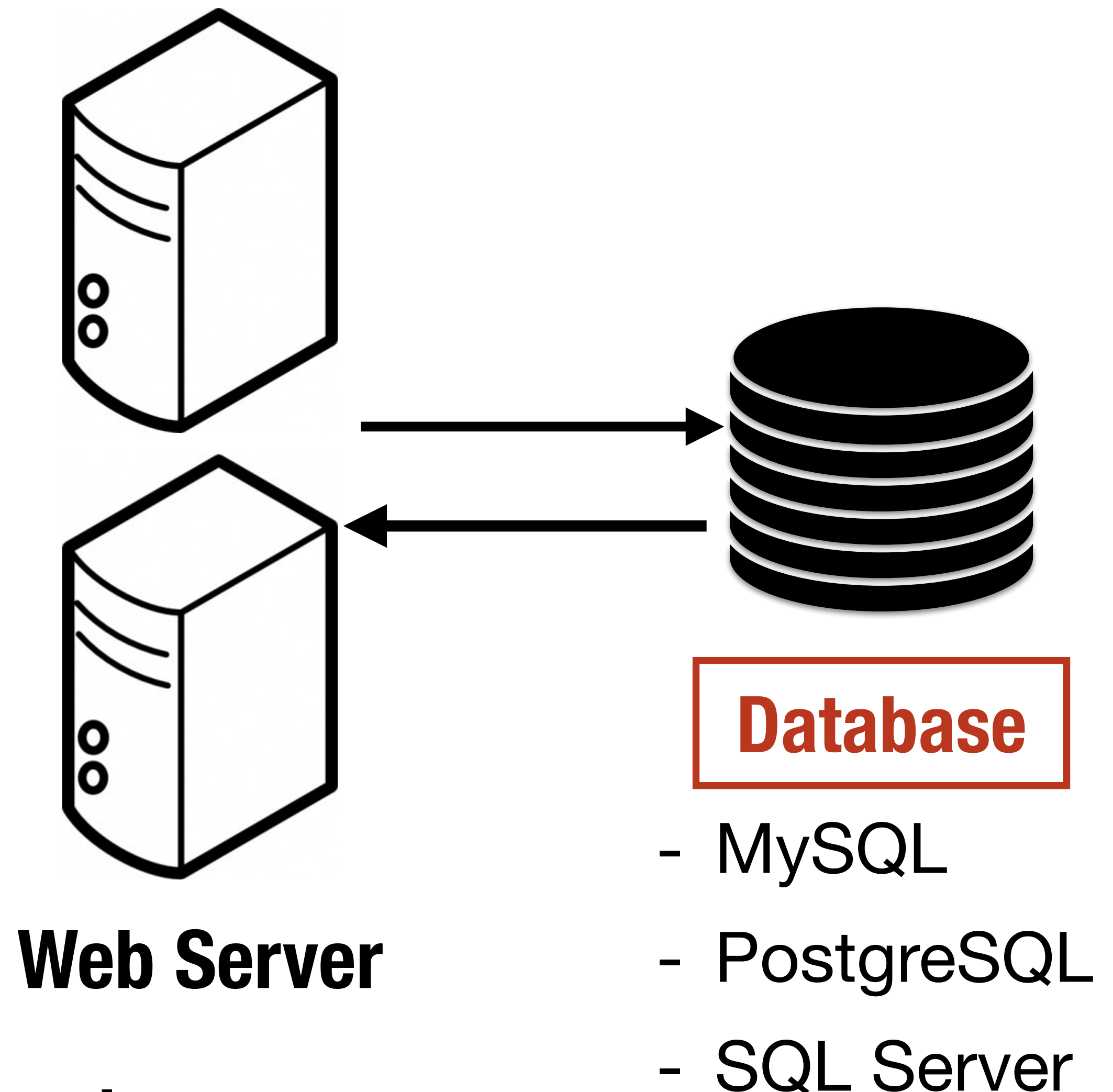


Architecture



Architecture

- Database server:
 - maintains state
 - stays consistent
- Web servers:
 - process client request
 - access state from database servers
 - may cache state but otherwise stateless



SQLite

- Lightweight database:
 - Store database in a file
 - Good for prototyping and examples
 - Only for single webserver

Using SQLite from Python

Connectors

- Low level connectors vs. Object-relational mapping (ORM)
- Many packages for low level connection
 - Most of them are compliant with the Python Database API Specification (PEP 249) <https://www.python.org/dev/peps/pep-0249/>
- We will be using **PySQLite Connector/Python**
 - Included in the standard library
 - Similar interface to database servers
 - Tutorial: <https://www.sqlitetutorial.net/sqlite-python/>

Python Database API Specification

- Two main objects
 - Connection
 - Cursor
- Connection methods
 - **cursor()** returns a new Cursor
 - **close()** closes connection to DB
 - **commit()** commits any pending transactions
 - **rollback()** rolls back to the start of any pending transaction (optional)

Connecting to a DB

```
import sqlite3

conn = sqlite3.connect("database_file.db")

# do some stuff

conn.close()
```

- The **connect()** constructor creates a connection to the SQLite database and returns a **Connection** object
- **connect()** takes the path to a database file (absolute or relative). If the file does not exist a new database is created.

Error Handling

📄 examples/python/sqlite/sqlite1.py

```
from sqlite3 import Error

try:
    conn = sqlite3.connect("database_file.db")
except Error as err:
    print(err)
else:
    # do some stuff
    conn.close()
```

All database statements should be done
inside **try: except:**

Python Database API Specification

- Cursor methods/attributes
 - **execute()** executes a database operation or query
 - **rowcount** read-only attribute, number of rows that the last execute command produced (SELECT) or affected (UPDATE, INSERT, DELETE)
 - **close()** closes the cursor
 - **fetchone()** fetches the next row of a query result set
 - **fetchmany()** fetches the next set of rows of a query result
 - **fetchall()** fetches all (remaining) rows of a query result
 - **arraysize** read/write attribute, specifying the number of rows to fetch at a time with **fetchmany()** (default is 1)

Creating a Table

📄 examples/python/sqlite/sqlite1.py

```
cur = conn.cursor()
try:
    sql = ("CREATE TABLE postcodes ("
           "postcode VARCHAR(4), "
           "location VARCHAR(20), "
           "PRIMARY KEY(postcode))")
    cur.execute(sql)
except Error as err:
    print("Error: {}".format(err))
else:
    print("Table created.")
finally:
    cur.close()
```

Dropping a Table

📄 [examples/python/sqlite/sqlite1.py](#)

```
cur = conn.cursor()
try:
    sql = "DROP TABLE postcodes"
    cur.execute(sql)
except Error as err:
    print("Error: {}".format(err))
else:
    print("Table dropped.")
finally:
    cur.close()
```

Inserting Data

📄 examples/python/sqlite/sqlite1.py

```
sql = "INSERT INTO postcodes (postcode, location) VALUES (?, ?)"
try:
    cur.execute(sql, (k, v)) # data is provided as a tuple
    conn.commit() # commit after each row
except Error as err:
    print("Error: {}".format(err))
```

- Add placeholder **?** to sql statement
- Data is provided as a tuple (list of values)
- DELETE and UPDATE work the same way
- You must **commit** the data after these statements

Querying Data

📄 examples/python/sqlite/sqlite1.py

```
cur = conn.cursor()
try:
    sql = ("SELECT postcode, location FROM postcodes "
           "WHERE postcode BETWEEN ? AND ?")
    cur.execute(sql, ("4000", "5000"))
    for (postcode, location) in cur:
        print("{}: {}".format(postcode, location))
except Error as err:
    print("Error: {}".format(err))
finally:
    cur.close()
```

- Use **cur.fetchall()** to get list of row values

Object-Relational Mapping

- For Object-Relational Mapping (ORM), see SQLAlchemy
 - <https://www.sqlalchemy.org/>
 - Flask extension: <http://flask.pocoo.org/docs/0.12/patterns/sqlalchemy/>

```
users = Table('users', metadata,
              Column('user_id', Integer, primary_key=True),
              Column('name', String(40)),
              Column('age', Integer),
              Column('password', String),
              )
users.create()

i = users.insert()
i.execute(name='Mary', age=30, password='secret')

s = users.select(users.c.age < 40)
rs = s.execute()
```

Using SQLite from Flask

Flask Contexts

- Flask provides two contexts
- **request** variable is associated with the current request

```
from flask import request
```

- **g** is associated with the "global" application context

```
from flask import g
```

- typically used to cache resources that need to be created on a per-request case, e.g., DB connections
- resource allocation: **get_X()** creates resource X if it does not exist yet, otherwise returns the same resource
- resource deallocation: **teardown_X()** is a tear down handler

Example

📄 examples/python/flask/5_sqlite/app.py

```
def get_db():  
    if not hasattr(g, "_database"):  
        g._database = sqlite3.connect("database.db")  
    return g._database
```

```
@app.teardown_appcontext  
def teardown_db(error):  
    db = getattr(g, '_database', None)  
    if db is not None:  
        db.close()
```

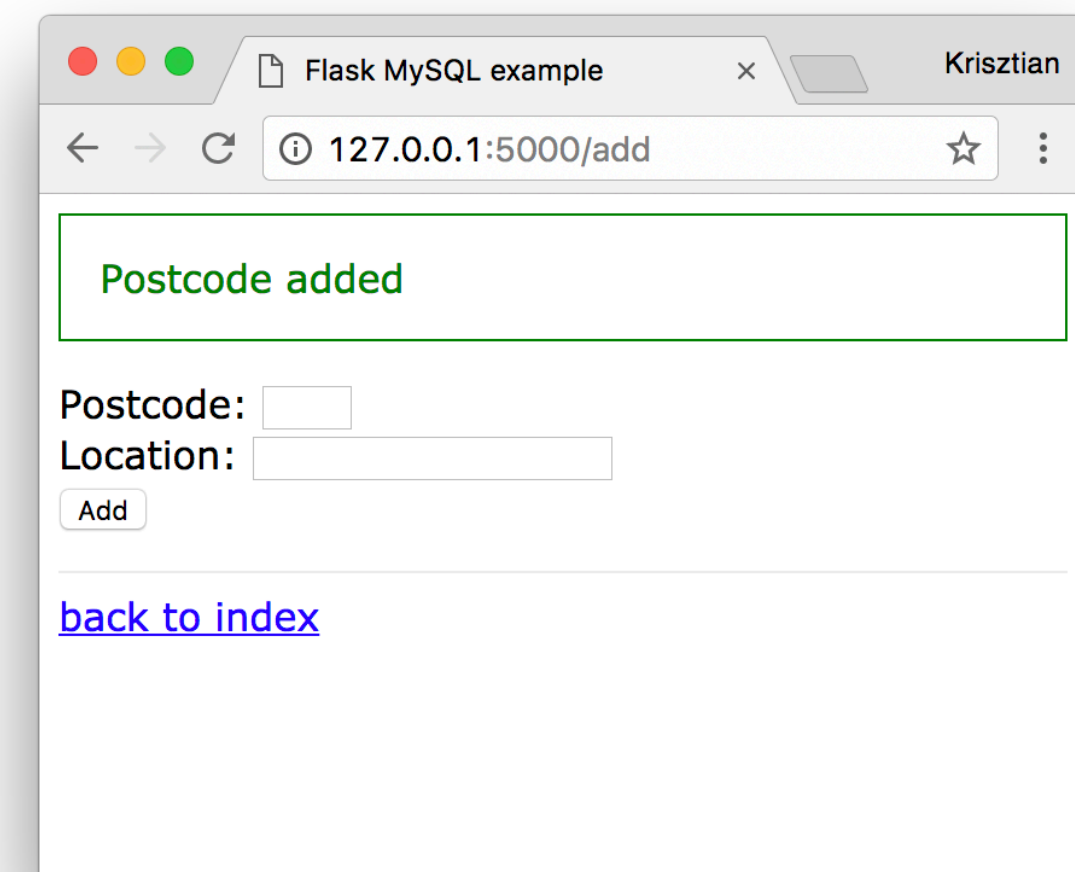
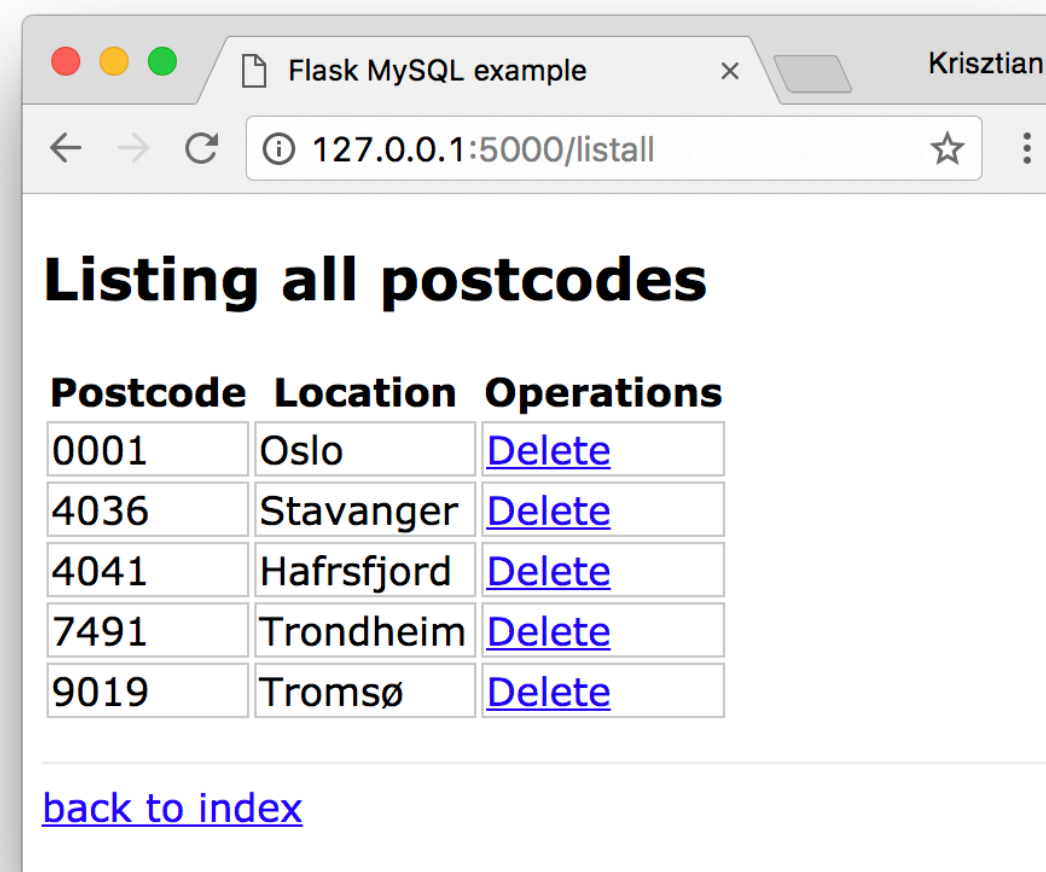
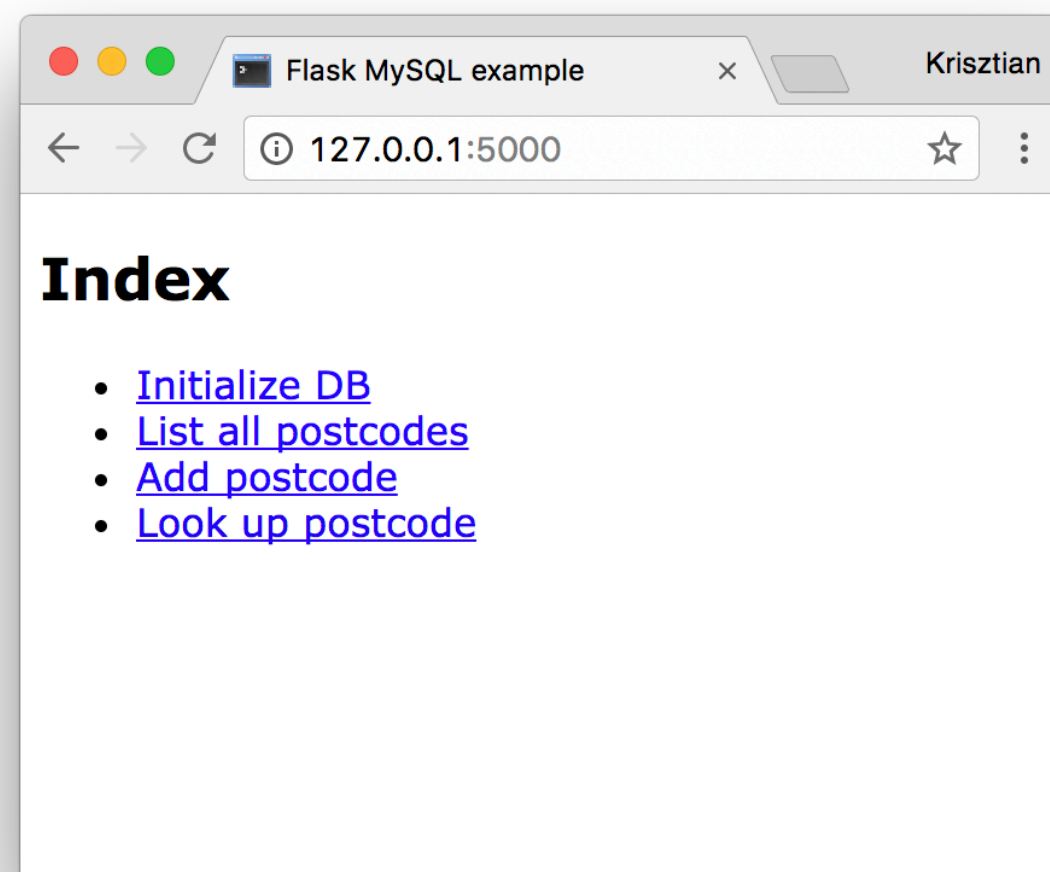
```
@app.route("/listall")  
def list_all():  
    """List all postcodes."""  
    db = get_db()  
    cur = db.cursor()
```

The first time **get_db()** is called the connection will be established

Example

🔗 [examples/python/flask/5_sqlite/app.py](#)

- Contains examples of CREATE TABLE, INSERT, SELECT (single/multiple records), DELETE
- Uses flashing for success messages



Exercises #1, #2



[github.com/dat310-2022/info/tree/main/](https://github.com/dat310-2022/info/tree/main/exercises/python/flask3)
exercises/python/flask3

Resources

- Python Database API Specification
<https://www.python.org/dev/peps/pep-0249/>
- SQLite3 Connector/Python
<https://docs.python.org/3/library/sqlite3.html>
- Flask SQLite
<https://flask.palletsprojects.com/en/1.1.x/patterns/sqlite3/>
- SQLite CLI
<https://sqlite.org/cli.html>