CS2102 Database Systems

Slides adapted from Prof. Chan Chee Yong

LECTURE 07B

APPLICATION DEVELOPMENT

PREVIOUS LECTURE

Transaction, procedures, triggers

- A transaction starts with `BEGINS` ends with either `COMMIT` or `ROLLBACK`
  - We assume ACID property is maintained
- Stored function
  - `CREATE [ OR REPLACE ] FUNCTION func_name ...`
  - `SELECT func_name (...);`
- Stored procedure
  - `CREATE [ OR REPLACE ] PROCEDURE proc_name ...`
  - `CALL proc_name (...);`
- Triggers
  -
```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER | INSTEAD OF }
{ event [ OR event [ ... ] ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN cond ] EXECUTE PROCEDURE func_name();
```

- Basic lightweight development stack
  - General workflow
  - Server & libraries
  - Routing
  - Database connection
  - Template
- Securing database
  - SQL injection attack

Overview

- Basic lightweight development stack
  - General workflow
  - Server & libraries
  - Routing
  - Database connection
  - Template
- Securing database
  - SQL injection attack

Basic lightweight development stack

General workflow

Database

- Store information

Connection library

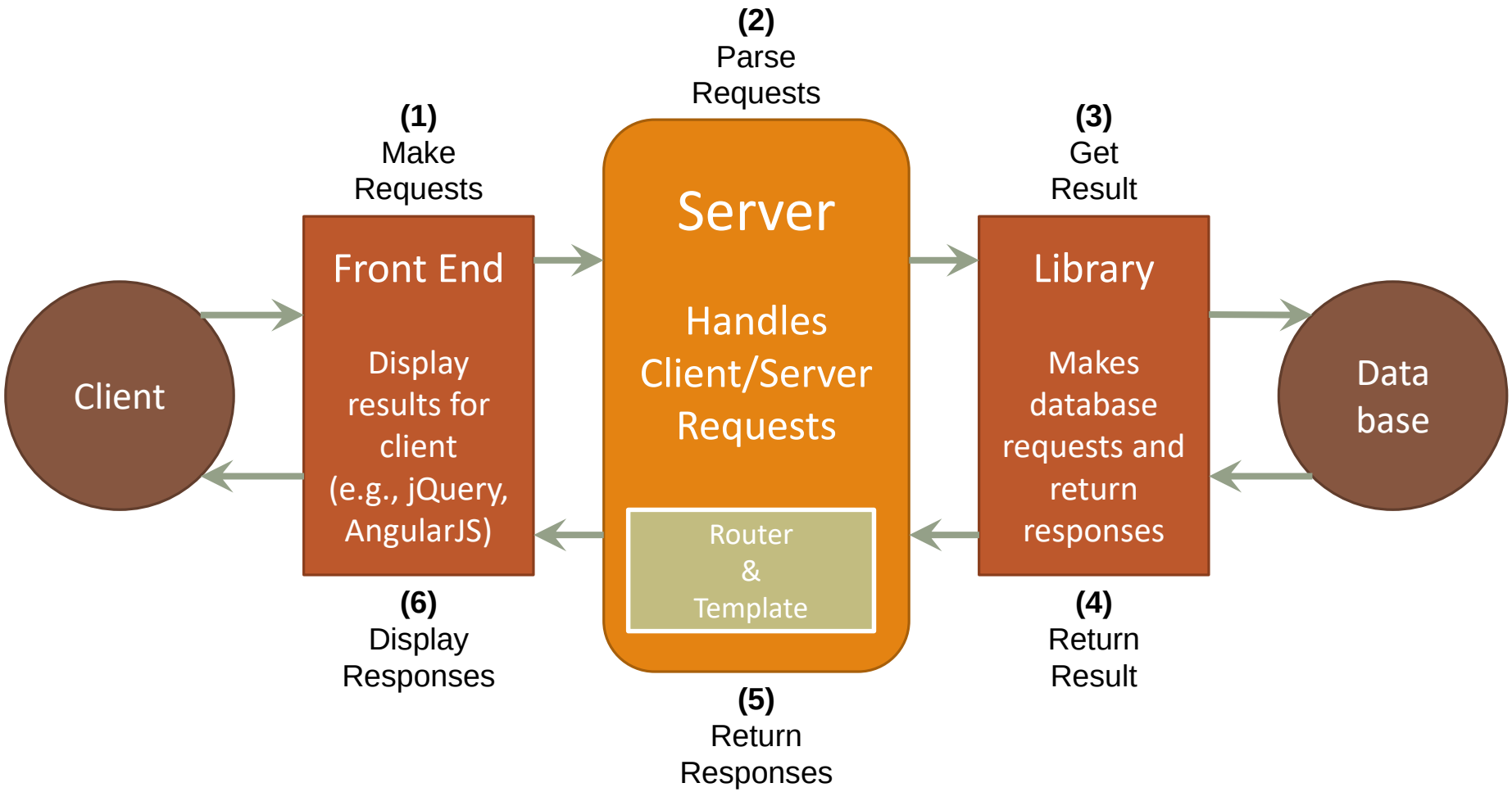
- Connect server with database

Server

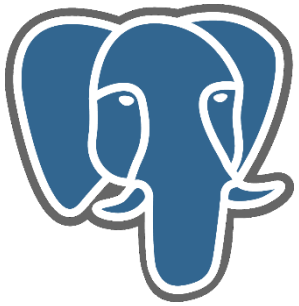
- Handle request from client
- Return answer with data from database

General workflow

Overview



General workflow



Database

- Store information
  - *PostgreSQL*

Connection library

- Connect server with database
  - *ExpressJS*

Server

- Handle request from client
- Return answer with data from database
  - *NodeJS*

Server & libraries

NodeJS and ExpressJS overlap responsibilities

- Accept connection
- Route if necessary
  - What to do when client request `/page`?
- Connect to database
  - How to construct SQL queries?
- Create and return response
  - What to give to client when `/page` is requested?

Server & libraries

Basic structure of an express web app

```
App/
+- bin/                # executables
|  +- www              # run with ‘npm start’
+- node_modules/
+- public/             # static files
|  +- images/          # - images
|  +- javascripts/     # - client-side js files
|  +- stylesheets/     # - css files
+- routes/             # handle /page request
+- views/              # template html
+- app.js              # main program
```

Routing

Basic

- Commonly done using RESTful API
  - `METHOD = GET, PUT, PATCH, POST, DELETE`
- Function: `app.METHOD(path, callback [, callback [...]])`
  - Example: handle request on `/page` using `GET` method
  - `app.get('/page', handler)`
    - References: https://expressjs.com/en/api.html#app.METHOD
- Basic handlers
  - **Display web page** `res.render(template, params)`
  - **Redirect to another page** `res.redirect(path)`
  - **Send something** `res.send(string | JSON | array)`
    - References: https://expressjs.com/en/api.html#res.render

Routing

Example

- GET
  - app.get(‘/register’, register_handler)
  - function register_handler(request, result, next) {
    res.render(template, params);
    }

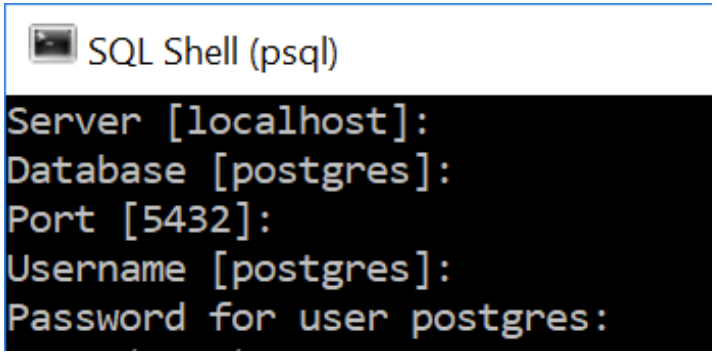
- POST
  - app.post(‘/register’, login_checker, *register_handler*)
  - function login_checker(request, result, *next*) {
    if(request.isAuthenticated()) {
    res.redirect(‘/index’);
    } else {
    next();
    }
    }

Database connection

node-postgre

- Client vs pool
  - Connecting to database is expensive (~30ms)
  - Database can only handle limited number of clients at a time
  - Reuse pool of client instead of creating new client
- Initialization code

```
const { Pool } = require('pg')
const pool = new Pool({
  user:     username,
  host:     hostname,
  database: database,
  password: password,
  post:     post_num
);
```



Database connection

node-postgre

- SQL query
  - `pool.query(sql_query, callback);`
- Example:
  -
```
function display_games(request, result, next) {
  function render_games(error, data) {
    if(error) {
      result.render(error_template);
    } else {
      result.render(template, { res: data.rows });
    }
  }
  pool.query("SELECT * FROM Games", render_games);
}
```
- Callback?
  - Callback is only invoked when results are ready
  - But functions return immediately!

Database connection

node-postgre

- What about multiple queries?
  - Attempt #1
    - `pool.query(sql_query1, callback_1);`
    - `pool.query(sql_query2, callback_2);`
  - Attempt #2
    -
```
pool.query(sql_query1, function(err, data) {
  if(err) { /* error */ }
  else    { pool.query(sql_query2, callback); }
});
```
  - Attempt #3
    - `pool.query(sql_transaction, callback);`

Template

Basic

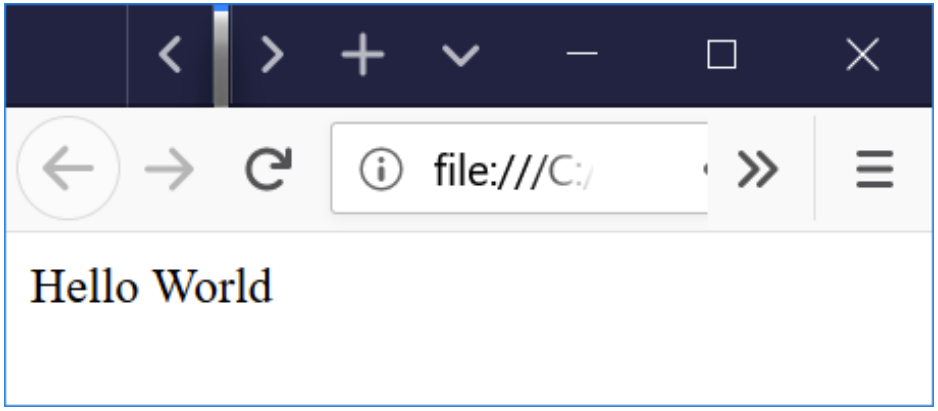
- So far, we have deferred discussion about rendering
  - `result.render(template, params)`
    - `result.render(error_template);`
    - `result.render(template, { res: data.rows });`
- What is a template?
  - A prototype HTML page
  - Embedded JavaScript Template (EJS)
    - HTML with placeholder for variable values
    - Can import common elements
    - Augmented with JavaScript for
      - Repetition
      - Selection

Template

No template

- At its most basic, EJS is equivalent to HTML
  - `<html><body><p>Hello World</p></body></html>`

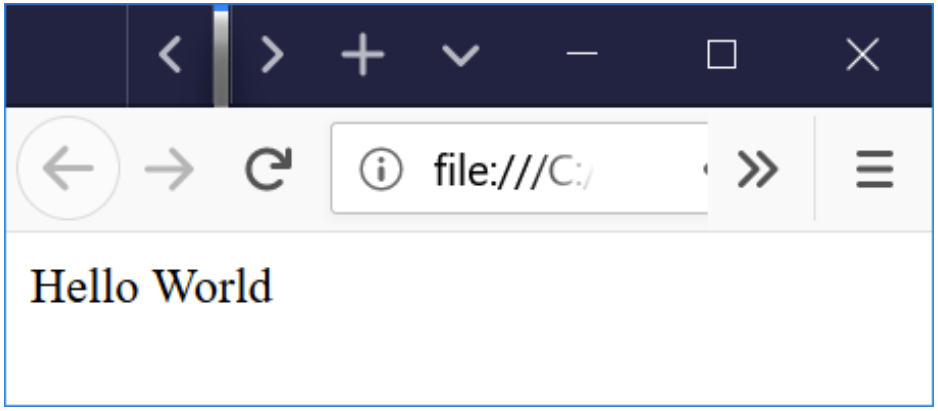
  - *HTML Tutorial* https://www.w3schools.com/html/
  - *CSS Tutorial* https://www.w3schools.com/css/



Template

Parameter

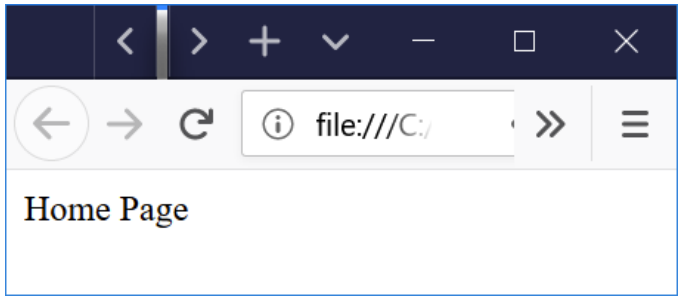
- Parameters can be passed to template
  - `result.render(template, {title: ‘Hello World’});`
- Arguments can be used in template
  - `<html><body><p><%= title %></p></body></html>`

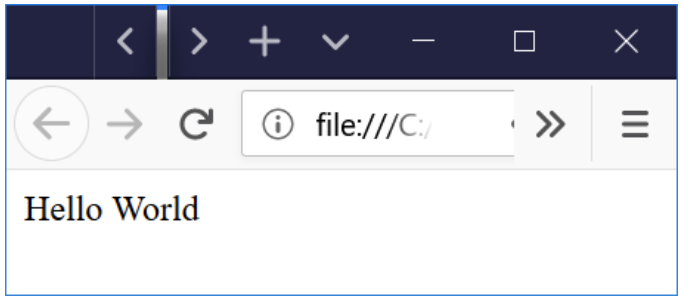
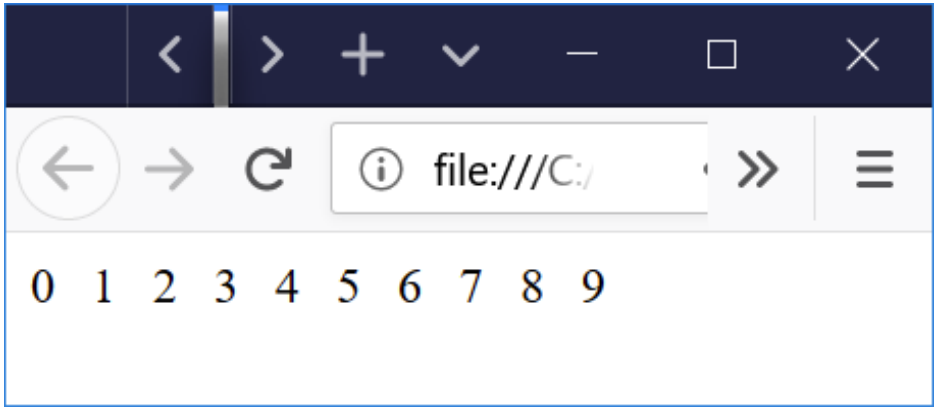


Template

Selections

- Choices can be made
  - ```
    <html><body>
    <% if(cond) { %>
      Home Page
    <% } else { %>
      Hello World
    <% } %>
    </body></html>
    ```

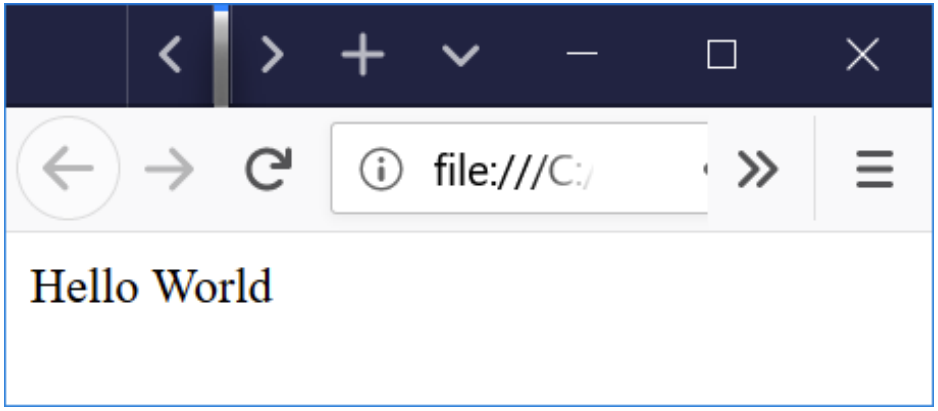



    ```

Template

Repetition

- Elements can be repeated multiple times
  - ```
    <html><body>
    <% for(var i=0; i<10; i++) { %>
      <%= i %>  
    <% } %>
    </body></html>
    ```


    ```

Template

Import

- Other templates can be included
  - header.ejs
    - `<p>Hello World</p>`
  - page.ejs
    -
```
<html><body>
<%- include header.ejs %>
</body></html>
```



- Basic lightweight development stack
  - General workflow
  - Server & libraries
  - Routing
  - Database connection
  - Template
- Securing database
  - SQL injection attack

Securing database

SELF-READ

Introduction

SQL injection attack

- Injection occurs when user-supplied data is sent to an interpreter as part of command or query
- Attackers trick the interpreter into executing unintended command via supplying specially crafted data
- Injection flaws allow attackers to create, read, update, or delete any arbitrary data available to the application
- In the worst case scenario, these flaws allow an attacker to completely compromise the application and the underlying systems, even bypassing deeply nested firewalled environments

- **TL;DR** very very very bad...
- Source: https://www.owasp.org/index.php/Top_10_2007-Injection_Flaws

SQL injection attack

Example

- Username-password retrieval
  - `pool.query(sql_query1, callback_1);`
  - `sql_query1 = “SELECT (username,password)`
    `FROM  userpass`
    `WHERE  username=‘” + input + “’”;`
- Attack
  - `input = “0’ OR ‘1’ = ‘1”;`
- Injected code
  - `SELECT (username,password)`
    `FROM  userpass`
    `WHERE  username=‘0’ OR ‘1’ = ‘1’;`

SQL injection attack

Placeholder

- Username-password retrieval
  - `pool.query(sql_query1, [input], callback_1);`
  -
```
sql_query1 = “SELECT (username,password)
               FROM   userpass
               WHERE  username=$1”;
```
- Attack
  - `input = “0’ OR ‘1’ = ‘1”;`
- Sanitized code
  -
```
SELECT (username,password)
FROM   userpass
WHERE  username=‘0\’ OR \‘1\’ = \‘1’;
```
- Reference: https://www.owasp.org/index.php/SQL_Injection

Summary

- Server & libraries
  - Accept connection
  - Route if necessary
    - What to do when client request `/page`?
  - Connect to database
    - How to construct SQL queries?
  - Create and return response
    - What to give to client when `/page` is requested?
- Security
  - SQL injection attack
  - Placeholder for sanitization