**CAT-CERT 2022 Web Study #6**

**202221162 김민재**

**목차**

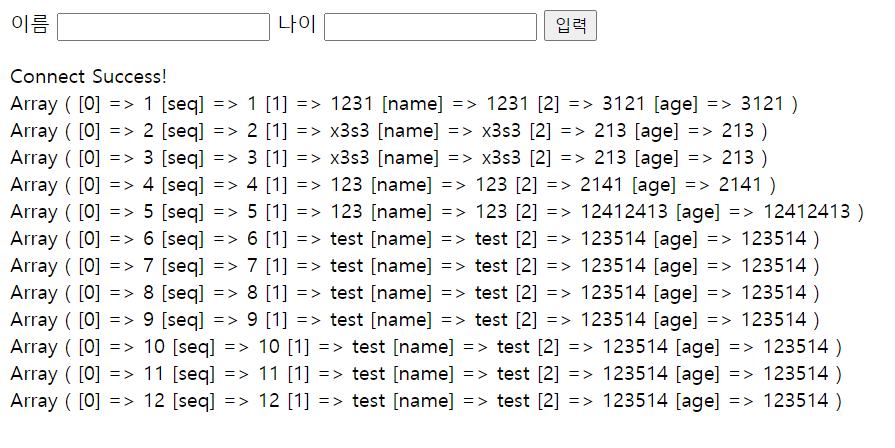
1. 5.18 스터디 실습
2. 방명록
3. 조사과제
4. **5.18 실습**

**dbinit.php**

|  |
| --- |
| <?php  $db\_Host = "localhost";  $db\_Id = "junejune0306";  $db\_Pass = "2022Wnsldj!";  $db\_Name = "junejune0306";  $conn = mysqli\_connect($db\_Host, $db\_Id, $db\_Pass, $db\_Name);  if (mysqli\_connect\_errno())  die('Connect Error : '.mysqli\_connect\_errno());  else {  printf("Connect Success!<br>");  mysqli\_query($conn, "set session character\_set\_connection=utf8;");  mysqli\_query($conn, "set session character\_set\_results=utf8;");  mysqli\_query($conn, "set session character\_set\_client=utf8;");  }  ?> |

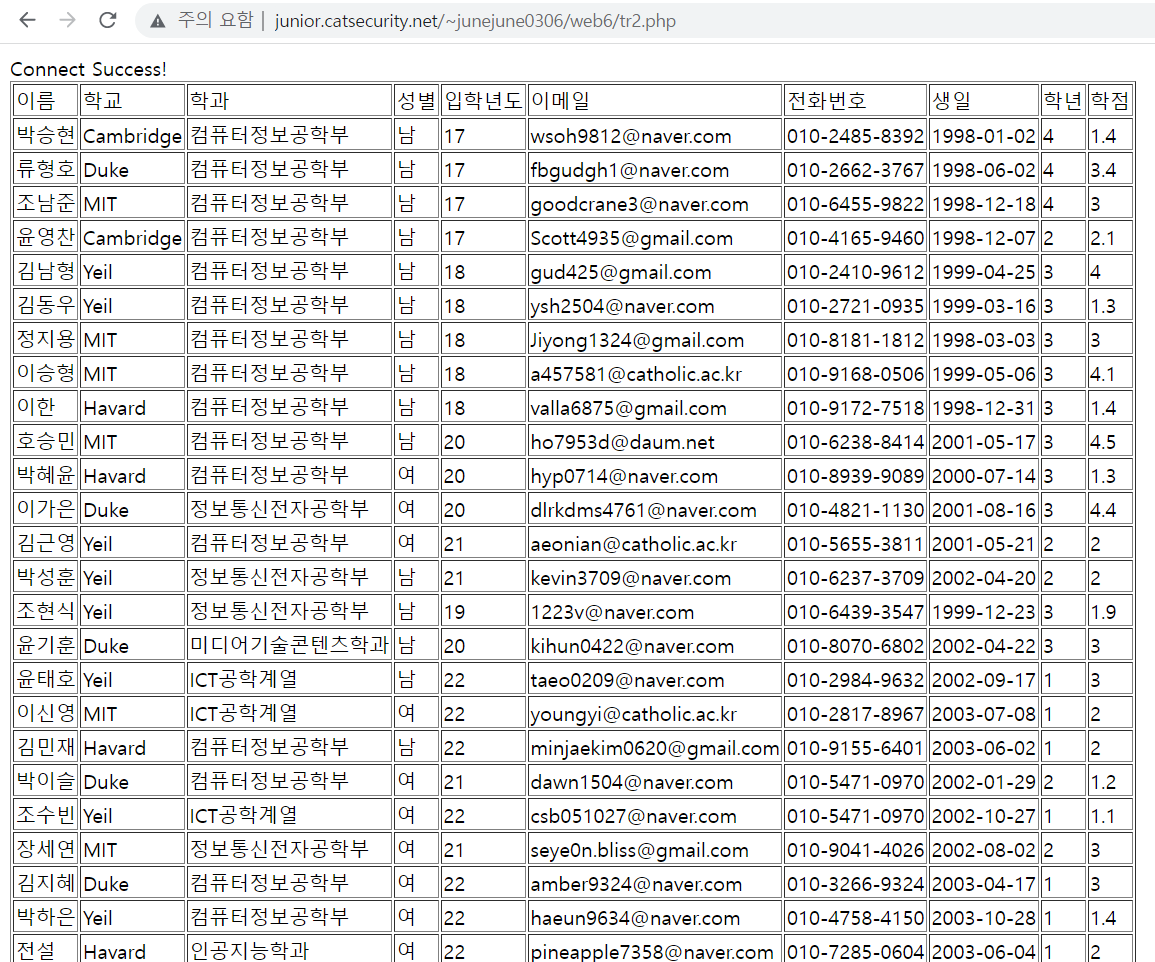
**#1**

|  |
| --- |
| <html>  <head>  <meta charset="utf-8">  <title>web6 tr1</title>  </head>  <body>  <form method="post" action="">  이름 <input type="text" name="name"/>  나이 <input type="text" name="age"/>  <input type="submit" value="입력"/>  </form>  </body>  </html>  <?php  include('./dbinit.php');  $name = $\_POST['name'];  $age = $\_POST['age'];  $query = "insert into test\_table (name, age) values ('$name', $age)";  mysqli\_query($conn, $query);  $query = "select \* from test\_table;";  if ($result = mysqli\_query($conn, $query)) {  while ($row = mysqli\_fetch\_array($result)) {  print\_r($row);  echo "<br>";  }  }  ?> |

****

**#2**

|  |
| --- |
| <html>  <head>  <meta charset="UTF-8">  <title>web6 tr1</title>  </head>  <body>  <table border="1px">  <tr>  <td>이름</td>  <td>학교</td>  <td>학과</td>  <td>성별</td>  <td>입학년도</td>  <td>이메일</td>  <td>전화번호</td>  <td>생일</td>  <td>학년</td>  <td>학점</td>  </tr><tr>  <?php  include('./dbinit.php');  $query = 'select \* from Old\_Member union select \* from New\_Member';  if ($result = mysqli\_query($conn, $query)) {  while ($row = mysqli\_fetch\_row($result)) {  unset($row[0]);  foreach($row as $cell){  echo "<td>$cell</td>";  }  echo '</tr><tr>';  }  }  ?>  </tr></table>  </body>  </html> |

****

**#3**

**app.js**

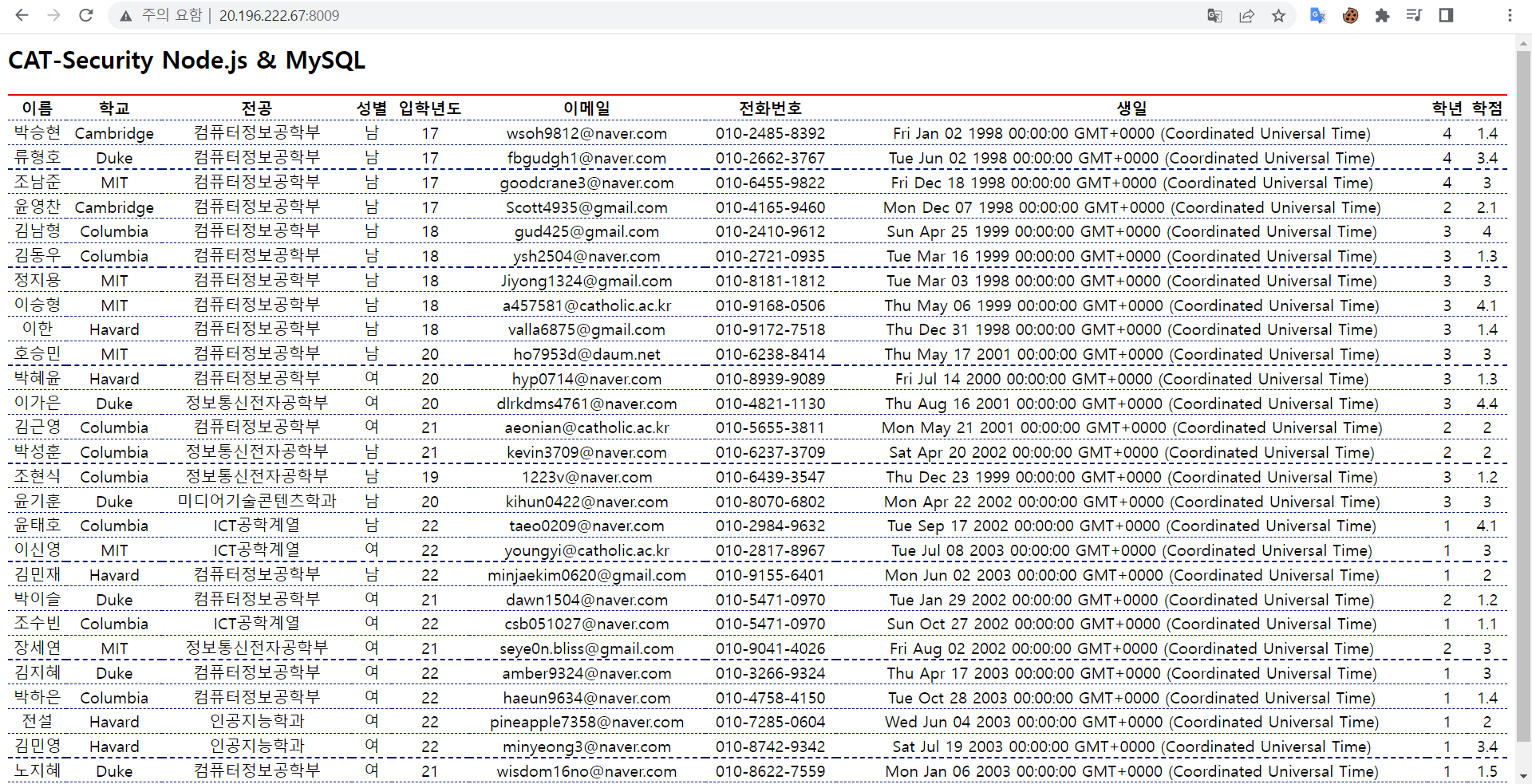
|  |
| --- |
| const express = require("express");  const ejs = require("ejs");  const path = require("path");  const app = express();  const conn = require("./db.js");  app.listen(8009, () => {  console.log("listening on port 8009");  });  app.use(express.urlencoded({ extended: false }));  app.set("views", path.join(\_\_dirname, "views"));  app.set("view engine", "ejs");  app.get("/", (req, res) => {  var sql = "SELECT \* FROM Old\_Member UNION SELECT \* FROM New\_Member;";  conn.query(sql, (err, rows) => {  if (err) console.log("query is not excuted. select fail!\n" + err);  else res.render("index.ejs", { list: rows });  });  }); |

**db.js**

|  |
| --- |
| const mysql = require( "mysql" );  const connection = mysql.createConnection({  host: 'localhost',  user: 'junejune0306',  password: '2022Wnsldj!',  port: 3306,  database: 'junejune0306'  });  connection.connect((err) => {  if (err) console.log(err);  else console.log( 'Connected!' );  });  module.exports = connection; |

**Index.ejs**

|  |
| --- |
| <html>  <head>  <title>CAT-Security Node.js & MySQL</title>  <style>  table {  width: 100%;  text-align: center;  border-collapse: collapse;  border-top: 2px solid #ed0606;  }  th, td {  border-bottom: 1px dashed #001763;  }  </style>  </head>  <body>  <h2>CAT-Security Node.js & MySQL</h2>  <div>  <table>  <thread>  <th> 이름 </th>  <th> 학교 </th>  <th> 전공 </th>  <th> 성별 </th>  <th> 입학년도 </th>  <th> 이메일 </th>  <th> 전화번호 </th>  <th> 생일 </th>  <th> 학년 </th>  <th> 학점 </th>  </thread>  <tbody> <% for (var i of list) { %> <tr>  <td> <%= i.Name %></td>  <td> <%= i.School %></td>  <td> <%= i.Major %></td>  <td> <%= i.Gender %></td>  <td> <%= i.EnterYear %></td>  <td> <%= i.Email %></td>  <td> <%= i.Phone %></td>  <td> <%= i.Birth %></td>  <td> <%= i.Grade %></td>  <td> <%= i.Score %></td>  </tr> <% } %> </tbody>  </table>  </div>  </body>  </html> |

****

1. **방명록 node.js**

**app.js**

|  |
| --- |
| const express = require("express");  const ejs = require("ejs");  const path = require("path");  const app = express();  const conn = require("./db.js");  app.listen(8009, () => {  console.log("listening on port 8009");  });  app.use(express.urlencoded({ extended: false }));  app.use(express.static(path.join(\_\_dirname, 'views')));  app.set("views", path.join(\_\_dirname, "views"));  app.set("view engine", "ejs");  app.get("/", (req, res) => {  var sql = "SELECT \* FROM guestBook ORDER BY seq DESC;";  conn.query(sql, (err, rows) => {  if (err) console.log("query is not excuted. select fail!\n" + err);  else {  for (var i of rows) {  i.time.setHours(i.time.getHours() + 9);  i.time = i.time.getFullYear()+"."+(i.time.getMonth()+1)+"."+i.time.getDate()+" "+i.time.getHours()+":"+i.time.getMinutes()+":"+i.time.getSeconds();  }  res.render("guestBook.ejs", { list: rows });  }  });  });  app.post("/", (req, res) => {  console.log(req.body);  var sql = "insert into guestBook (name, text) values (?, ?);";  params = [req.body.name, req.body.text];  if (params[0] != '' && params[1] != '') conn.query(sql, params, (err, rows) => {  if (err) console.log(err);  });  res.redirect("/");  });  app.get( "/delete/:seq", (req, res) => {  var sql = 'delete from guestBook where seq=?';  var params = [ req.params.seq ];  conn.query( sql, params, (err, rows) => {  if (err) console.log("query is not excuted. modify fail!\n" + err);  else res.redirect("/");  })  });  app.post( "/modify/:seq", (req, res) => {  var sql = 'update guestBook set text=? where seq=?';  var params = [ req.body.text, req.params.seq ];  conn.query( sql, params, (err, rows) => {  if (err) console.log("query is not excuted. modify fail!\n" + err);  else res.redirect("/");  })  }); |

**db.js**

|  |
| --- |
| const mysql = require( "mysql" );  const connection = mysql.createConnection({  host: 'localhost',  user: 'junejune0306',  password: '2022Wnsldj!',  port: 3306,  database: 'junejune0306'  });  connection.connect((err) => {  if (err) console.log(err);  else console.log( 'Connected!' );  });  module.exports = connection; |

**./views/**

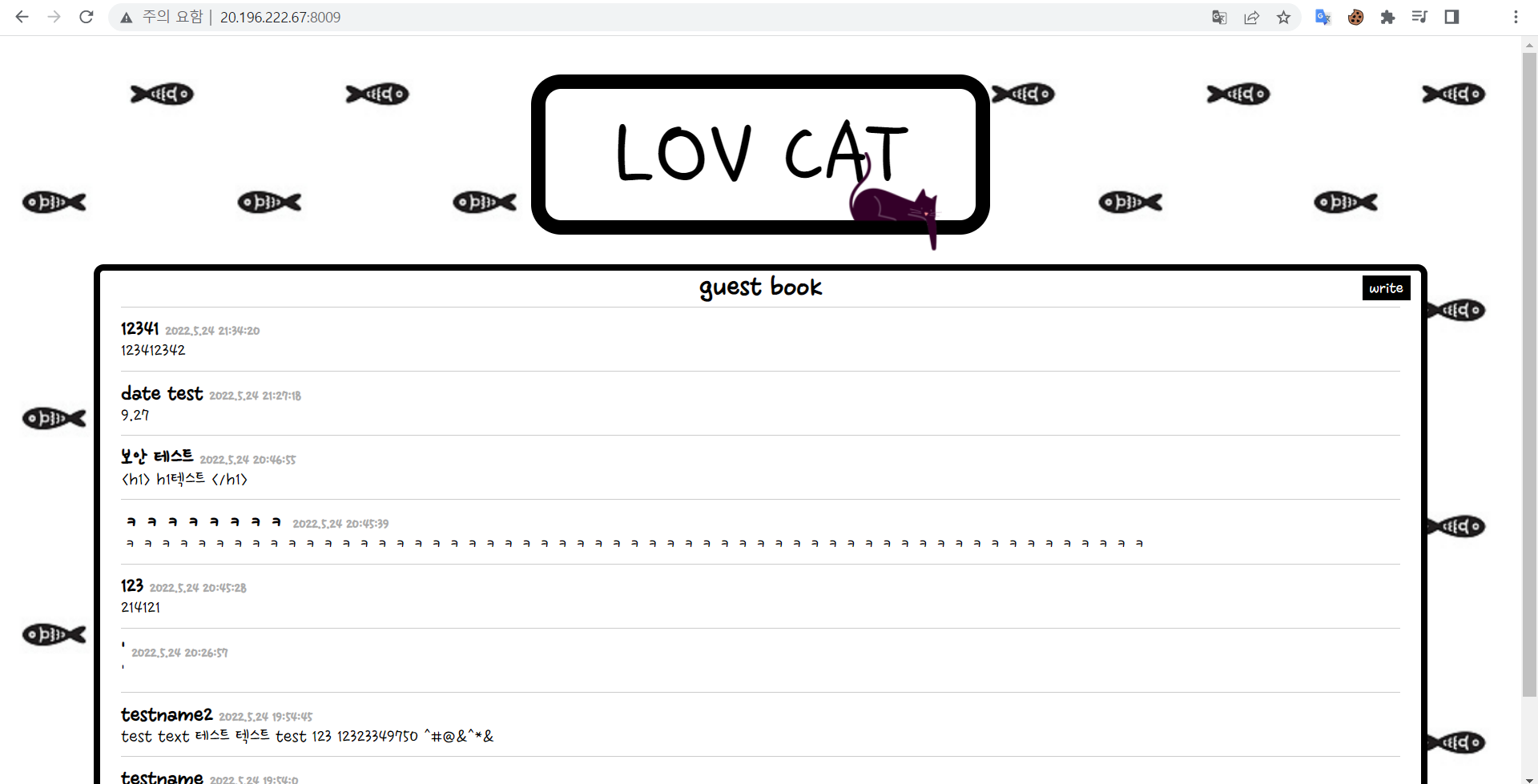
****

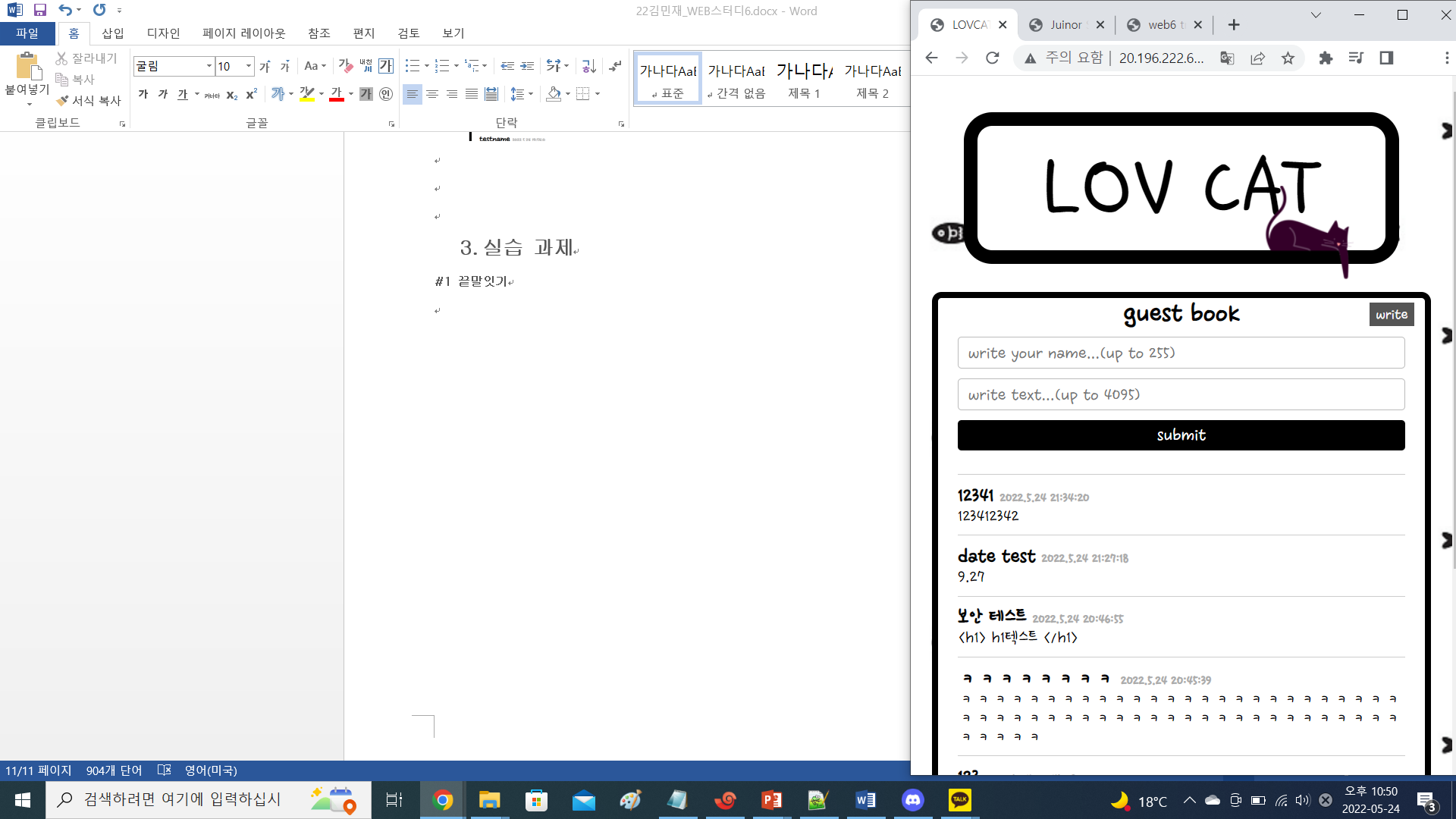
**guestbook.ejs**

|  |
| --- |
| <html>  <head>  <meta charset="utf-8">  <title>LOVCAT - guest book</title>  <link rel="stylesheet" href="guestBook.css">  </head>  <body>  <div id="headline">LOV CAT  <img id="catonboard" src="catonboard.png">  </div><br>  <div id="book"><div id="bookt">guest book</div>  <button id="addbtn" class="btn" onclick="document.getElementById('addBook').style.display = 'inline-block';">write</button>  <div id="addBook">  <form method="post" action="" onkeydown="return event.key != 'Enter';">  <input type="text" name="name" placeholder="write your name...(up to 255)" autocomplete="off"/>  <input type="text" name="text" placeholder="write text...(up to 4095)" autocomplete="off"/>  <input type="submit" class="btn" value="submit"/>  </form>  </div>  <% for (var i of list) { %><div class="page">  <div class="head"><%= i.name %> <span> <%= i.time %></span></div><div class="text"><%= i.text %>  </div><div class="config"><span onclick="this.parentNode.nextSibling.style.display='block';this.parentNode.previousSibling.innerHTML='';console.log(this.parentNode.previousSibling)">수정</span> <span onclick="location.href='/delete/<%= i.seq %>'">삭제</span>  </div><form method="post" action="/modify/<%= i.seq %>">  <input type="text" value="<%= i.text %>" name="text" placeholder="write text...(up to 4095)" autocomplete="off"/>  <input type="submit" class="btn" value="modify"/>  </form>  </div><% } %>  </div>  </body>  </html> |

**guestbook.css**

|  |
| --- |
| @font-face {  src: url("./whitecatfont.ttf");  font-family: "cat";  }  \* {  font-family: "cat";  color: black;  }  body {  background-image: url("background\_0.png");  text-align: center;  }  form {  margin: 0;  }  .btn {  background-color: #000000;  border: none;  color: white;  cursor: pointer;  }  .btn:hover {  background-color: #555555;  }  .btn:active {  background-color: #000000;  }  #headline {  display: inline-block;  position: relative;  margin: 30px 0 0 0;  padding: 15px 65px;  border: 15px solid black;  border-radius: 30px;  background-color: white;  font-size: 100px;  }  #catonboard {  position: absolute;  height: 100px;  width: 100px;  bottom: -33px;  right: 33px;  }  #book {  display: inline-block;  position: relative;  margin: 30px 0;  padding: 15px 20px;  border: 7px solid black;  border-radius: 10px;  background-color: white;  width: 85%;  font-size: 20px;  }  #bookt {  font-weight: bold;  font-size: 30px;  line-height: 0px;  margin: 0 0 20px 0;  }  #addbtn {  position: absolute;  top: 4px;  right: 10px;  padding: 4px 7px;  font-size: 17px;  }  #addbook {  display: none;  width: 100%;  }  input {  display: inline-block;  padding: 6px 10px;  width: 100%;  margin: 5px 0;  border: 1px solid #bbb;  border-radius: 4px;  font-size: 20px;  }  .page {  position: relative;  padding: 10px 0;  border-top: 1px solid #ccc;  text-align: left;  font-size: 20px;  }  .page .head {  font-weight: bold;  font-size: 23px;  }  .page span {  font-weight: light;  font-size: 15px;  color: #aaa;  }  .page .config {  position: absolute;  top: 10px;  right: 10px;  cursor: default;  }  .page .config span {  cursor: pointer;  }  .page form {  display: none;  } |





1. **조사 과제**

**#REST API (Representational State Transfer API)**

**- REST 의 구성**

1. 자원(Resource) : HTTP URI

2. 자원에 대한 행위(Verb) : HTTP Method

3. 자원에 대한 행위의 내용 (Representations) : HTTP Message Pay Load

**- REST 의 특징**

1. Uniform (유니폼 인터페이스)

URI로 지정한 리소스에 대한 조작을 통일되고 한정적인 인터페이스로 수행하는 아키텍처 스타일.

2. Stateless (무상태성)

작업을 위한 상태정보를 따로 저장하고 관리하지 않는다. 세션 정보나 쿠키정보를 별도로 저장하고 관리하지 않기 때문에 API 서버는 들어오는 요청만을 단순히 처리하면 되므로, 서비스의 자유도가 높아지고 서버에서 불필요한 정보를 관리하지 않음으로써 구현이 단순해진다.

3. Cacheable (캐시 가능)

HTTP라는 기존 웹표준을 그대로 사용하기 때문에, 웹에서 사용하는 기존 인프라를 그대로 활용이 가능하고, 따라서 HTTP가 가진 캐싱 기능이 적용 가능하다. HTTP 프로토콜 표준에서 사용하는 Last-Modified태그나 E-Tag를 이용한다.

4. Self-descriptiveness (자체 표현 구조)

REST API 메시지만 보고도 이를 쉽게 이해 할 수 있는 자체 표현 구조로 되어 있다.

5. Client - Server 구조

REST 서버는 API 제공, 클라이언트는 사용자 인증이나 컨텍스트(세션, 로그인 정보)등을 직접 관리하는 구조로 각각의 역할이 확실히 구분되기 때문에, 클라이언트와 서버에서 개발해야 할 내용이 명확해지고 서로간 의존성이 줄어든다.

6. 계층형 구조

REST 서버는 다중 계층으로 구성될 수 있으며 보안, 로드 밸런싱, 암호화 계층을 추가해 구조상의 유연성을 둘 수 있고 PROXY, 게이트웨이 같은 네트워크 기반의 중간매체를 사용할 수 있다.

**- REST 의 장단점**

**\* 장점**

HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구출할 필요가 없다.

HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해 준다.

HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.

Hypermedia API의 기본을 충실히 지키면서 범용성을 보장한다.

REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있다.

여러 가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.

서버와 클라이언트의 역할을 명확하게 분리한다.

**\* 단점**

표준이 자체가 존재하지 않아 정의가 필요하다.

사용할 수 있는 메소드가 4가지밖에 없다.

HTTP Method 형태가 제한적이다.

브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.

구형 브라우저에서 호환이 되지 않아 지원해주지 못하는 동작이 많다. ex) Explorer

**- REST API**

REST의 원리를 따르는 API.

**- RESTful**

REST API의 설계 규칙을 올바르게 지킨 시스템을 일컫는 말.