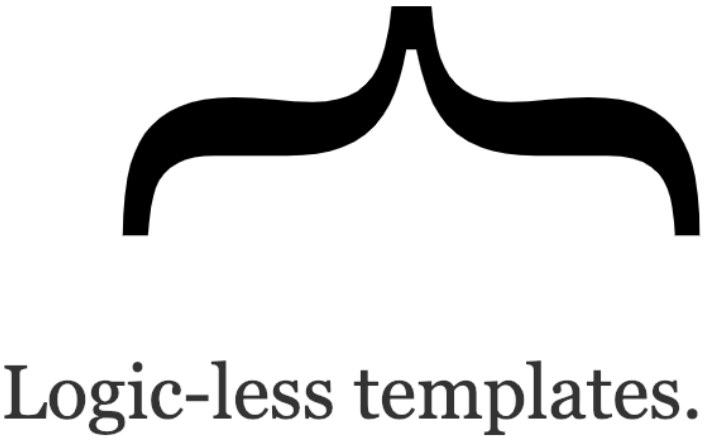
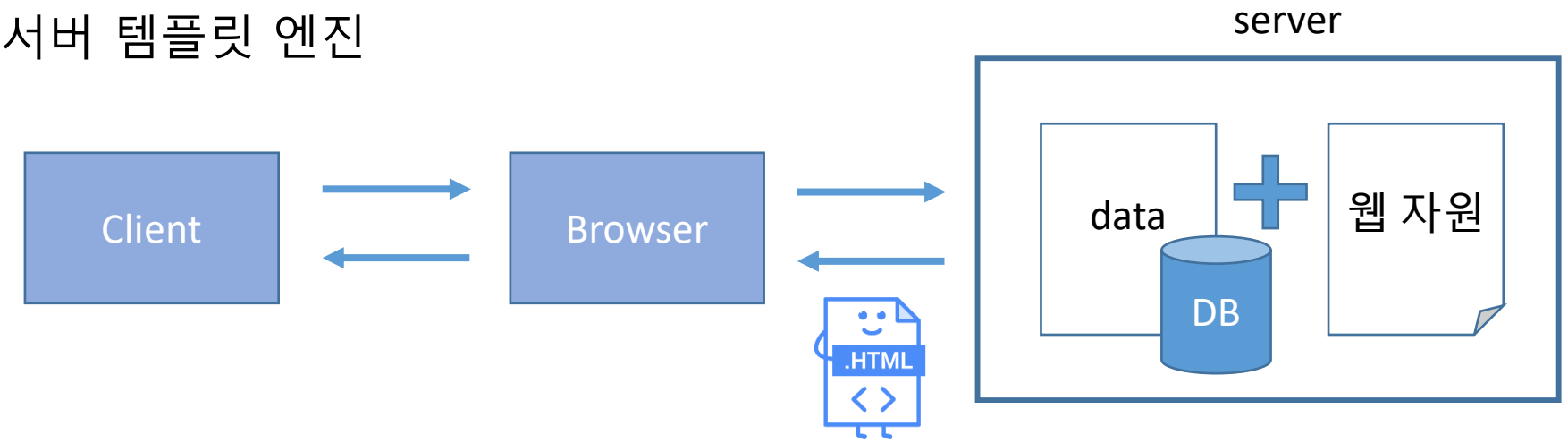
CH4 Mustache

정승혜

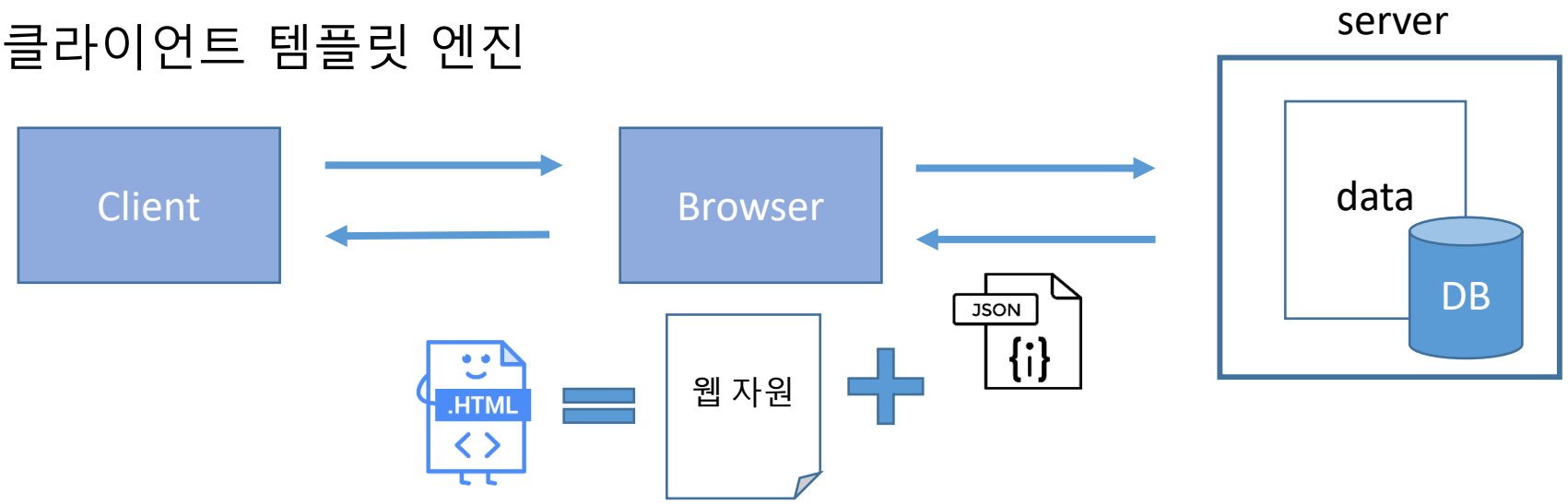


서버/클라이언트 템플릿 엔진

- 서버 템플릿 엔진



- 클라이언트 템플릿 엔진



머스테치(Mustache)

- 지정된 템플릿 양식과 데이터를 합쳐서 HTML로 출력해주는 SW
- 대부분의 언어를 지원하므로 언어에 따라 서버 템플릿 엔진 혹은 클라이언트 템플릿 엔진으로 사용 가능

- **장점**

- 문법이 다른 템플릿 엔진보다 심플
- 로직코드를 사용할 수 없어서 View 역할과 서버 역할이 명확하게 분리됨
- Mustache.js와 Mustache.java 2가지 모두 있어, 하나의 문법으로 클라이언트/서버 템플릿을 모두 사용 가능

4.2 기본 페이지

4.2 기본 페이지

- 의존성 등록

```
compile('org.springframework.boot:spring-boot-starter-mustache')
```

- Mustache 자동 로딩 경로

  = > src/main/resources/templates

- **IndexController.java 와 index.mustache 작성**

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class IndexController {

    @GetMapping("/")
    public String index(){
        return "index";
    }
}
```

머스테치 스타터에 의해서 return 문자열에 대해
앞 경로와 뒤의 .mustache 확장자가 자동 생성되어
View Resolver로 넘어가 처리

4.3 게시글 등록 화면 생성

4.3 게시글 등록 화면

- HTML 뿐만 아니라 프론트엔드 라이브러리를 사용해보자
- 사용하는 방법에는 2가지가 있음

    1. 외부 CDN 사용 – 실 서비스에서는 의존성 문제로 잘 사용 x
    2. 직접 라이브러리를 받아서 사용

라이브러리를 레이아웃 방식으로 header/footer 형태로 추가

- header.mustache , footer.mustache

```
<!DOCTYPE HTML>
<html>
<head>
    <title>스프링부트 웹서비스</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
```

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

</body>
</html>
```

4.3 게시글 등록 화면

- Css 위치와 js 위치가 다른 이유

- 페이지 로딩 속도를 높이기 위해서 화면을 그리는 css는 header에서 불러오도록 하고, js는 footer에서 불러오도록 함

- Bootstrap.js의 경우는 jquery에 의존하므로 jquery가 먼저 호출되도록 작성

```
<!DOCTYPE HTML>
<html>
<head>
    <title>스프링부트 웹서비스</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
```

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

</body>
</html>
```

4.3 게시글 등록 화면

1. 글 등록 버튼 추가 – index.mustache
2. 게시글 작성(등록) 페이지 경로 매핑 – IndexController.java
3. 게시글 작성(등록) 페이지 화면 작성 – posts-save.mustache
4. 글 등록 버튼에 대한 API 호출 js 파일 작성 – index.js

4.3 게시글 등록 화면

1. 글 등록 버튼 추가 – index.mustache

```
{{>layout/header}}

    <h1>스프링 부트로 시작하는 웹 서비스 Ver.2</h1>
    <div class="col-md-12">
        <div class="row">
            <div class="col-md-6">
                <a href="/posts/save" role="button" class="btn btn-primary">글 등록</a>
            </div>
        </div>
    </div>
{{>layout/footer}}
```

2. 게시글 작성페이지 경로 매핑 – IndexController.java

```
@GetMapping("/posts/save")
public String postsSave(){
    return "posts-save";
}
```

4.3 게시글 등록 화면

3. 게시글 작성페이지 화면 작성 – posts-save.mustache

```
{{>layout/header}}

<h1>게시글 등록</h1>

<div class="col-md-12">
    <div class="col-md-4">
        <form>
            <div class="form-group">
                <label for="title">제목</label>
                <input type="text" class="form-control" id="title" placeholder="제목을 입력하세요">
            </div>
            <div class="form-group">
                <label for="author"> 작성자 </label>
                <input type="text" class="form-control" id="author" placeholder="작성자를 입력하세요">
            </div>
            <div class="form-group">
                <label for="content"> 내용 </label>
                <textarea class="form-control" id="content" placeholder="내용을 입력하세요"></textarea>
            </div>
        </form>
        <a href="/" role="button" class="btn btn-secondary">취소</a>
        <button type="button" class="btn btn-primary" id="btn-save">등록</button>
    </div>
</div>

{{>layout/footer}}
```

4.3 게시글 등록 화면

4. 글 등록 버튼에 대한 API 호출 js 파일 작성 – index.js

```
var main = {
    init : function () {
        var _this = this;
        $('#btn-save').on('click', function () {
            _this.save();
        });
    },
    save : function () {
        var data = {
            title: $('#title').val(),
            author: $('#author').val(),
            content: $('#content').val()
        };

        $.ajax({
            type: 'POST',
            url: '/api/v1/posts',
            dataType: 'json',
            contentType:'application/json; charset=utf-8',
            data: JSON.stringify(data)
        }).done(function() {
            alert('글이 등록되었습니다.');
            window.location.href = '/';
        }).fail(function (error) {
            alert(JSON.stringify(error));
        });
    }
};

main.init();
```

var main = { … } 을 통해 내부에 function을 작성

⇒ 다른 js 파일 내부의 function들과 이름이 같은 경우 브라우저의 공용 공간에서 덮어써지거나 중복되는 현상을 막기 위해 js 파일의 유효범위를 만들어 줌

⇒ 특정 객체 안에서만 function이 유효해짐

4.3 게시글 등록 화면

4. 글 등록 버튼에 대한 API 호출 js 파일 작성 – index.js

사용될 수 있도록 footer.mustache에 추가

```
<!--index.js 추가-->
<script src="/js/app/index.js"></script>
```

4.4 전체 조회 화면 추가

4.4 전체 조회 화면

1. 게시글 전체 조회 화면(목록출력) 추가 – index.mustache
2. 조회 쿼리 작성 – PostsRepository.java
3. 데이터를 받을 Dto 작성 – PostsListResponseDto.java
4. 조회된 데이터를 Dto와 연결 – PostsService.java
5. 메인 기본화면에 목록(서비스에서 넘어온) 뷰 추가 – IndexController.java

4.4 전체 조회 화면

1. 게시글 전체 조회 화면(목록출력) 추가 – index.mustache

```
<!-- 목록 출력 영역 -->
<table class="table table-horizontal table-bordered">
    <thead class="thead-strong">
    <tr>
        <th>게시글번호</th>
        <th>제목</th>
        <th>작성자</th>
        <th>최종수정일</th>
    </tr>
    </thead>
    <tbody id="tbody">
    {{#posts}}
        <tr>
            <td>{{id}}</td>
            <td><a href="/posts/update/{{id}}">{{title}}</a></td>
            <td>{{author}}</td>
            <td>{{modifiedDate}}</td>
        </tr>
    {{/posts}}
    </tbody>
</table>
```

{{#posts}} : list 순회 (for)

{{id}} 등의 {{변수명}} :
list에서 뽑아낸 객체 필드

4.4 전체 조회 화면

2. 조회 쿼리 작성 – PostsRepository.java

```
@Query("SELECT p FROM Posts p ORDER BY p.id DESC")
List<Posts> findAllDesc();
```

Repostiory에 query 추가시

1. @Query 어노테이션 사용
2. SpringDataJpa가 제공하는 기본 메소드 사용

- 위와 같은 조회 메서드는 프로젝트 규모가 커질 경우 DB 조건들때문에 Entity 클래스로 처리하기 어려워서 조회용 프레임워크를 추가로 사용

- querydsl , jooq, MyBatis

4.4 전체 조회 화면

3. 데이터를 받을 Dto 작성 – PostsListResponseDto.java

```
@Getter
public class PostsListResponseDto {

    private Long id;
    private String title;
    private String author;
    private LocalDateTime modifiedDate;

    public PostsListResponseDto(Posts entity){
        this.id = entity.getId();
        this.title = entity.getTitle();
        this.author = entity.getAuthor();
        this.modifiedDate = entity.getModifiedDate();
    }
}
```

4. 조회된 데이터를 Dto와 연결 – PostsService.java

```
@Transactional(readOnly = true)
public List<PostsListResponseDto> findAllDesc() {
    return postsRepository.findAllDesc().stream()
            .map(PostsListResponseDto::new)
            .collect(Collectors.toList());
}
```

4.4 전체 조회 화면

5. 메인 기본화면에 목록 뷰 추가 – IndexController.java

```
@GetMapping("/")
public String index(Model model){
    model.addAttribute("posts",postsService.findAllDesc());
    return "index";
}
```

4.5 게시글 수정/삭제

4.5 게시글 수정/삭제 화면

1. 게시글 수정/삭제 화면 작성 – posts-update.mustache
2. 수정/삭제 API 호출 위한 js 파일 업데이트 – index.js
3. API 확인 – PostsService.java
4. 메인 페이지에서 수정 페이지로 이동가능하게 수정 – index.mustache
5. 게시글 수정 페이지 경로 매핑 – IndexController.java

DB

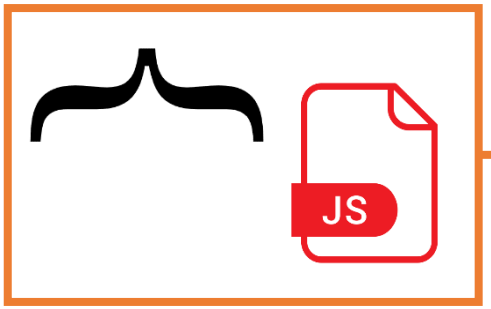
Repository
- 인터페이스 형태
- Entity 클래스와 맞물려있음
- 쿼리
- SpringDataJpa 기본 메소드

Entity - domain
- DB와 연결된
- 신규 데이터 업데이트

Dto
- 내부 데이터 교환에 사용됨

Service
- Repository를 변수로 지님
- Dto를 활용해서 repository로 데이터를 받아 web에 전송하거나
- 웹에서 넘어온 dto형태의 새로운 데이터를 저장 혹은 업데이트

Controller
- Service를 변수로 지님
- 화면 경로 매핑 컨트롤러
- API 컨트롤러



Mustache의 이벤트에 따라 js가 API 호출