

# Task R6

Il task R6 richiedeva l'implementazione di una versione in lingua inglese dell'applicazione.

L'approccio scelto è stato quello di modificare le rotte esistenti che restituivano pagine HTML, premettendo a tale percorso un ulteriore livello, che identifica la lingua. Ad esempio:

- `https://localhost:80/it/modificaClasse` per la pagina in italiano;
- `https://localhost:80/en/modificaClasse` per la pagina in inglese.

A tale struttura dovrà chiaramente corrispondere una gerarchia analoga della directory in cui le pagine HTML sono contenute, distinguendo una cartella "it" per le pagine in italiano ed una cartella "en" per le pagine in inglese.

L'approccio si rivela in realtà fortemente scalabile, in quanto al netto delle modifiche effettuate in questa iterazione e di una minima aggiunta al file di configurazione del reverse proxy, permette di aggiungere altre cartelle contenenti le pagine HTML tradotte in qualsiasi altra lingua (ad esempio, "es" per lo spagnolo, "fr" per il francese).


## Installazione di Maven

Lavorando con l'IDE Visual Studio Code, è stato necessario installare Maven, ovvero uno strumento di gestione di progetti software ampiamente utilizzato nell'ecosistema Java per automatizzare il processo di build di un progetto, la gestione delle dipendenze, la distribuzione e la documentazione del software.

1. Download del Binary zip archive.

Maven – Download Apache Maven

Apache Maven 3.9.6 is the latest release: it is the recommended version for all users.

 <https://maven.apache.org/download.cgi>

2. Aggiunta del percorso dell'archivio alla variabile d'ambiente `PATH`.
3. Posizionarsi con la shell nella directory `{project_path}/T1-G11/applicazione/manvsclass`.
4. Eseguire il comando `mvn clean install` per rieseguire il processo di build.
5. Rieseguire lo script `installer.bat`.

## Templates directories

Path: `T1-G11/applicazione/manvsclass/src/main/resources/templates`

```
templates \
|
| --> it \
|   --> home.html
|   --> home_admin.html
|   --> login_admin.html
|   --> modificaClasse.html
```

```

| --> registraAdmin.html
| --> reportClasse.html
| --> Reports.html
| --> uploadClasse.html
| --> en \
| --> home.html
| --> home_admin.html
| --> login_admin.html
| --> modificaClasse.html
| --> registraAdmin.html
| --> reportClasse.html
| --> Reports.html
| --> uploadClasse.html

```

Path: `T5-G2/t5/src/main/resources/templates`

```

templates \
|
| --> it \
| --> change_password.html
| --> editor.html
| --> login.html
| --> main.html
| --> report.html
| --> en \
| --> change_password.html
| --> editor.html
| --> login.html
| --> main.html
| --> report.html

```

Path: `T23-G1/src/main/resources/templates`

```

templates \
|
| --> it \
| --> login.html
| --> mail_register.html
| --> password_change.html
| --> password_reset.html
| --> register.html
| --> en \
| --> login.html
| --> mail_register.html
| --> password_change.html
| --> password_reset.html
| --> register.html

```

## HomeController.java

Il framework Spring, il quale implementa il pattern Model-View-Controller (MVC) permette di definire all'interno del Controller delle rotte che restituiscono pagine HTML dinamicamente (Views). La sintassi è simile alla

seguente:

```
@Controller
public class WelcomeController {

    @GetMapping("/welcome")
    public String showWelcomePage() {
        // Ritorna il nome del file HTML (senza estensione) da visualizzare
        return "welcome";
    }
}
```

Nel nostro caso, è stato necessario anteporre un parametro (chiamato coerentemente "language") che viene letto come variabile e utilizzato per costruire il percorso del file HTML all'interno della directory, divisa ora in una cartella `it/` e una cartella `en/`.

Path: `T1-G11/applicazione/manvsclass/src/main/java/com/groom/manvsclass/controller`

```
@GetMapping("{language}/home_admin")
public String showHomeAdmin(@PathVariable String language) {
    return language + "/home_admin";
}

@GetMapping("{language}/loginAdmin")
public String showLoginAdmin(@PathVariable String language) {
    return language + "/login_admin";
}

@GetMapping("{language}/registraAdmin")
public String showRegistraAdmin(@PathVariable String language) {
    return language + "/registraAdmin";
}

@GetMapping("{language}/modificaClasse")
public String showModificaClasse(@PathVariable String language) {
    return language + "/modificaClasse";
}

@GetMapping("{language}/uploadClasse")
public String showUploadClasse(@PathVariable String language) {
    return language + "/uploadClasse";
}

@GetMapping("{language}/reportClasse")
public String showReportClasse(@PathVariable String language) {
    return language + "/reportClasse";
}

@GetMapping("{language}/Reports")
public String showReports(@PathVariable String language) {
    return language + "/Reports";
}
```

Path: T5-G2/t5/src/main/java/com/g2/t5

```
@GetMapping("{language}/main")
public String GUIController( @PathVariable String language,
                             Model model,
                             @CookieValue(name = "jwt", required = false) String jwt) {

    MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
    formData.add("jwt", jwt);

    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class);

    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";

    // ...

    return language + "/main";
}

// -----

@GetMapping("{language}/report")
public String reportPage( @PathVariable String language,
                          Model model,
                          @CookieValue(name = "jwt", required = false) String jwt) {
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
    formData.add("jwt", jwt);

    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class);

    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";

    return language + "/report";
}

// -----

@GetMapping("{language}/editor")
public String editorPage( @PathVariable String language,
                          Model model,
                          @CookieValue(name = "jwt", required = false) String jwt) {
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<String, String>();
    formData.add("jwt", jwt);

    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class);

    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";

    return language + "/editor";
}
```

Path: T23-G1/src/main/java/com/example/db\_setup

```
@GetMapping("{language}/register")
public ModelAndView showRegistrationForm

    (@PathVariable String language,
     HttpServletRequest request,
     @CookieValue(name = "jwt", required = false) String jwt) {

    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language + "/main");

    return new ModelAndView(language + "/register");
}

// -----

@GetMapping("{language}/login")
public ModelAndView showLoginForm

    (@PathVariable String language,
     HttpServletRequest request,
     @CookieValue(name="jwt", required=false) String jwt) {

    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language + "/main");

    return new ModelAndView(language + "/login");
}

// -----

@GetMapping("{language}/password_reset")
public ModelAndView showResetForm

    (@PathVariable String language,
     HttpServletRequest request,
     @CookieValue(name="jwt", required=false) String jwt) {

    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language + "/main");

    return new ModelAndView(language + "/password_reset");
}

// -----

@GetMapping("{language}/password_change")
public ModelAndView showChangeFor

    (@PathVariable String language,
     HttpServletRequest request,
     @CookieValue(name="jwt", required=false) String jwt) {

    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language + "/main");

    return new ModelAndView(language + "/password_change");
}

// -----
```

```

@GetMapping("/{language}/mail_register")
public ModelAndView showMailForm

    (@PathVariable String language,
    HttpServletRequest request,
    @CookieValue(name="jwt", required=false) String jwt) {

    if(!isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language + "/main");

    return new ModelAndView(language + "/mail_register");
}

```

### Altre risorse modificate

Il refactoring ha richiesto un aggiornamento di tutti i file, sia frontend (HTML e JavaScript) che backend (le stesse classi Java in cui avviene un reindirizzamento) i quali avevano un riferimento ad una delle rotte modificate, ovvero tutte quelle che restituiscono un file HTML.

### application.properties

Path: `T1-G11/applicazione/manvsclass/src/main/resources`

Il file `application.properties` in un progetto Java Spring è utilizzato per configurare le proprietà dell'applicazione. Spring Boot, in particolare, utilizza questo file per definire vari parametri di configurazione dell'applicazione, come impostazioni del database, configurazioni di sicurezza, configurazioni del server e molte altre opzioni.

Nel nostro caso, abbiamo aggiunto due voci per garantire che Thymeleaf, ovvero il motore di template utilizzato da Spring, cercasse i file nella directory giusta e con l'estensione `.html`.

```

spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html

```

### default.conf

Path: `ui_gateway`

Il file `default.conf` è una configurazione di Nginx, un server web e reverse proxy. Questa configurazione specifica come gestire le richieste in arrivo sulla porta 80 del server.

Anche in questo caso, è stato necessario anteporre alle rotte il path `/it` e `/en` per permettere al web server di risolvere i nomi con cui ora vengono identificate le pagine.

```

server {
    listen 80;
    server_name localhost;
    proxy_read_timeout 600s;
}

```

```

# Reindirizza la radice a /login
location = / {
    return 301 /login;
}

# Gestisce le route che corrispondono a /loginAdmin, /registraAdmin, /orderbydate, /Dfilterby..., ...
location ~ ^/(loginAdmin|registraAdmin|orderbydate|Dfilterby.+|orderbyname|delete|getLikes|uploadFile|t1) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://manvsclass-controller-1:8080;
}

# Gestisce le route che corrispondono a /it/home, /en/home, /it/home_adm, /en/home_adm, ...
location ~ ^/(it|en)/(home|home_adm|loginAdmin|registraAdmin|modificaClasse|Reports|uploadClasse|reportClasse) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://manvsclass-controller-1:8080;
}

# Gestisce le route che corrispondono a /logout, /t23
location ~ ^/(logout|t23) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t23-g1-app-1:8080;
}

# Gestisce le route che corrispondono a /it/login, /en/login, /it/register, /en/register, ...
location ~ ^/(it|en)/(login|register|mail_register|password_change|password_reset) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t23-g1-app-1:8080;
}

# Gestisce la route che corrisponde a /t5
location ~ ^/(t5) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
}

# Gestisce le route che corrispondono a /it/main, /en/main, /it/editor, /en/editor, /it/report, /en/report, ...
location ~ ^/(it|en)/(main|editor|report) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
}

# Gestisce le richieste che iniziano con /api
location ^~ /api {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://api_gateway-gateway-1:8090;
}

# Disabilita i log di accesso
access_log off;
# Specifica il percorso per il log degli errori
error_log /var/log/nginx/error.log error;
}

```



Per l'aggiunta di ulteriori lingue, è sufficiente aggiungere il carattere **pipe** “|” seguito dalle due lettere identificative della nuova lingua (ad esempio, `|es|fr` per aggiungere lo spagnolo e il francese).