# REST Framework

## DeepDive Week 1 Day 2

# Материалы и Ресурсы

Сегодня мы начнем собирать ВСЕ в единый фреймворк и добавим возможность создавать собственные контроллеры на конкретные HTTP пути.

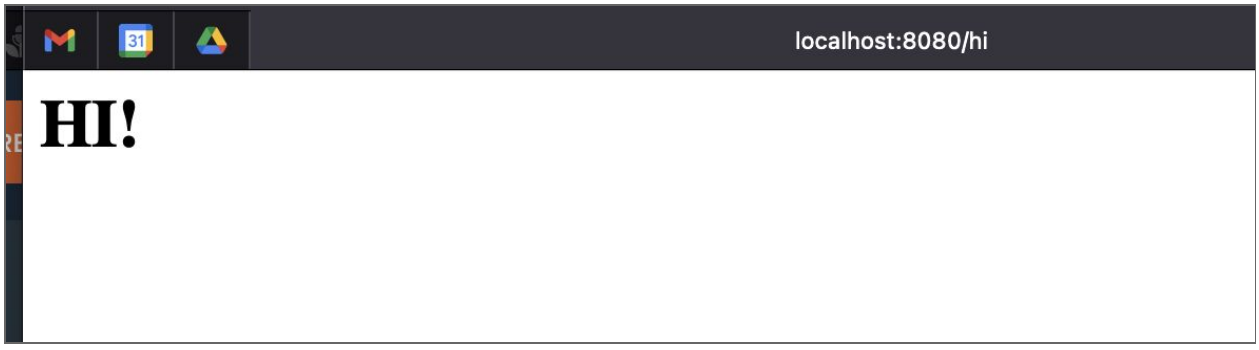
Для этого:

- создадим примитивы, которыми будем описывать HTTP запросы/ответы:
  - enum **ContentType**
  - enum **HttpMethod**
  - enum **HttpVersion**
  - record **HttpRequest**
  - record **HttpResponse**
  - interface **HttpReqeustHanlder**
- сделаем логику работы с ними:
  - **ConcurrentHttpServerWithPath** – основной сервер обработчик HTTP запросов, который мапит каждый запрос на нужный контроллер (или возвращает ошибку 404)

Давайте сначала посмотрим как будет выглядеть использование этого фреймворка со стороны стороннего разработчика:

```
var serverThread = new ConcurrentHttpServerWithPath();
serverThread.addHandler(new HttpRequestsHandler() {
    @Override
    public String path() {
        return "/hi";
    }
    @Override
    public HttpMethod method() {
        return HttpMethod.GET;
    }
    @Override
    public HttpResponse process(HttpRequest request) {
        return new HttpResponse.Builder().body("<h1>HI!</h1>").build();
    }
});
serverThread.start();
```

И соответственно в браузере:



Ну а теперь давайте создадим сам фреймворк со всеми этими классами.

## ContentType

```
package academy.kovalevskyi.javadeepdive.week1.day2;

public enum ContentType {
   TEXT_HTML, // TODO
   APPLICATION_JSON; // TODO
   // TODO
}
```

## HttpMethod

```
public enum HttpMethod {
   POST, PUT, GET
}
```

## HttpVersion

```
package academy.kovalevskyi.javadeepdive.week1.day2;

public enum HttpVersion {
   HTTP_1_1; // TODO
   // TODO
}
```

# HttpRequest

```
package academy.kovalevskyi.javadeepdive.week1.day2;
import java.util.Optional;

public record HttpRequest(String path, HttpMethod httpMethod, Optional<String>
body, ContentType contentType, HttpVersion httpVersion) {

   public static class Builder {
       // TODO
       public Builder path(String path) {
           // TODO
       }
       public Builder method(HttpMethod method) {
           // TODO
       }
       public Builder body(String body) {
           // TODO
       }
       public Builder contentType(ContentType contentType) {
           // TODO
       }
       public Builder httpVersion(HttpVersion httpVersion) {
           // TODO
       }
       public HttpRequest build() {
           // TODO
       }
   }
}
```

Важно! По умолчанию билдер должен создавать запрос с:

- Http версией 1.1
- Content type text/html
- Method GET
- path “/”

# HttpResponse

```
package academy.kovalevskyi.javadeepdive.week1.day2;

public record HttpResponse(ResponseStatus status, ContentType contentType, String body, HttpVersion
httpVersion) {

   public final static HttpResponse ERROR_404 = new Builder().status(ResponseStatus.ERROR_404).build();
   public final static HttpResponse OK_200 = new Builder().status(ResponseStatus.OK).build();
   public final static HttpResponse ERROR_500 = new Builder().status(ResponseStatus.ERROR_500).build();

   public static class Builder {
       // TODO
       public HttpResponse build() {
           // TODO
       }
       public Builder status(ResponseStatus status) {
           // TODO
       }
       public Builder contentType(ContentType contentType) {
           // TODD
       }
       public Builder body(String body) {
           // TODO
       }
       public Builder httpVersion(HttpVersion version) {
           // TODO
       }
   }

   public enum ResponseStatus {
       OK(200, "OK"), ERROR_404(404, "not found"), ERROR_500(500, "server error");
      // TODO
   }
}
```

Важно! По умолчанию билдер должен создавать ответ с:

- кодом 200
- Content type text/html
- Http версией 1.1

Теперь, имея все нужные блоки, можно перейти к созданию более высокоуровневых частей.

## HttpRequestsHandler

```
package academy.kovalevskyi.javadeepdive.week1.day2;

public interface HttpRequestsHandler {
   String path();
   HttpMethod method();
   HttpResponse process(HttpRequest request);
}
```

Этот интерфейс будет реализовывать пользователи фреймворка, которые смогут добавлять свои собственные контролы.

## ConcurrentHttpServerWithPath

```
package academy.kovalevskyi.javadeepdive.week1.day2;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ConcurrentHttpServerWithPath extends Thread {
  // TODO
  public void addHandler(HttpRequestsHandler handler) {
    // TODO
  }
  public void run() {
    // TODO
  }
  public void stopServer() {
    // TODO
  }
  public boolean isLive() {
    // TODO
  }
}
```

Важно, что все хендлеры должны быть добавлены до запуска сервера.