АРХИТЕКТУРНИ РЕШЕНИЯ ЗА СЪВРЕМЕННИ УЕБ ПРИЛОЖЕНИЯ

SINGLE PAGE APPLICATIONS (SPA) И MULTI PAGE APPLICATIONS (MPA)

Емил симеонов
Senior Product Manager & Development Architect
Bulpros Consulting AD

ПОЗНАВАМЕ REST, HTTP(S), VERT.X И КАКВО?!

- Можем да разработим някакъв много прост, статичен уеб сайт, но едва ли си струва усилието...
  - Можем да използваме Content Management System (CMS). Например, WordPress, Umbrako и др.
  - Може би дори не ни е необходимо да използваме някакви кой знае какви технологии (HTML + уеб сървър)
- Ще построим по-сложно уеб приложение. Това са уеб сайтове, в които
  - Често разчитаме на автентикация и ауторизация
  - Се пази някакво състояние за всеки текущ потребител
  - Е наличен workflow, включващ изпълнението на AJAX заявки

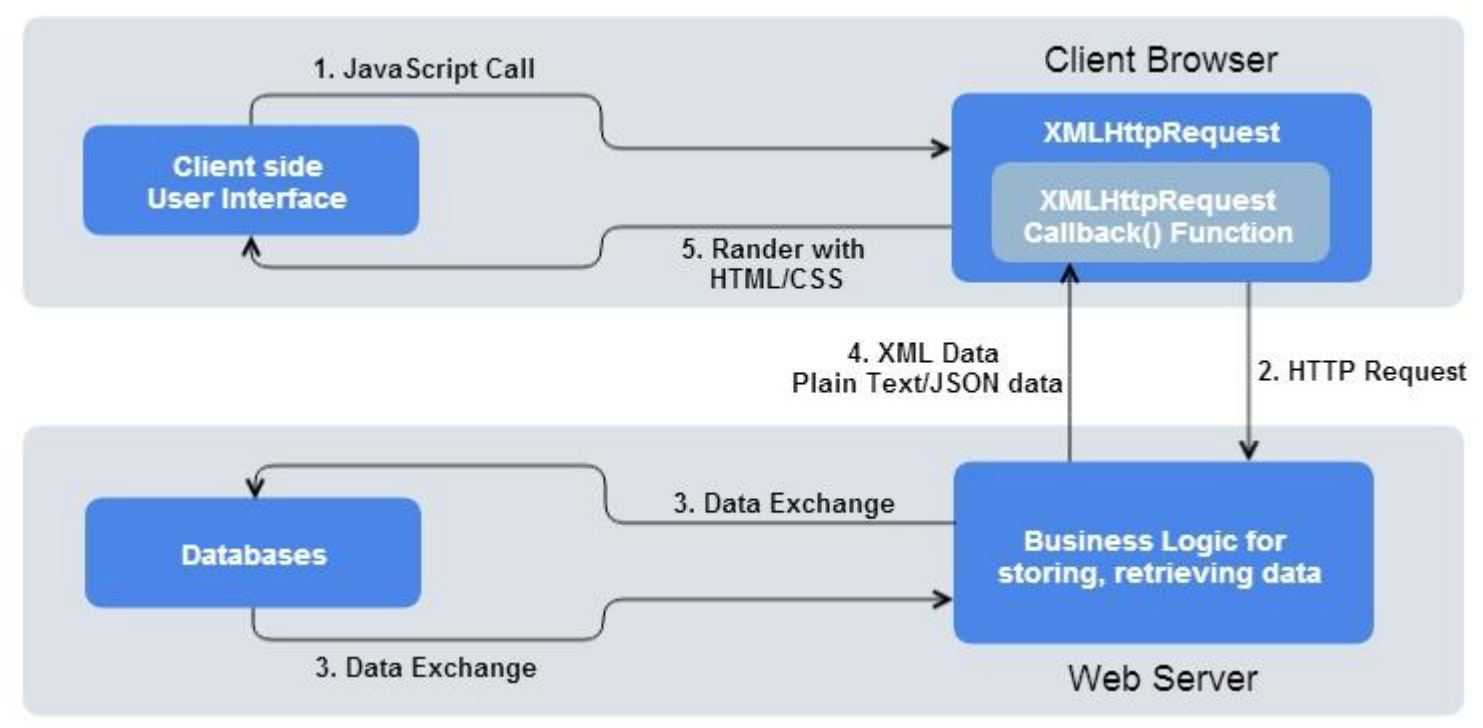
Съществуват два утрвърдени архитектурни подхода, с които следва да сме запонати:

Single Page Applications (SPA) и Multi Page Applications (MPA)

ЕВОЛЮЦИЯ НА СОФТУЕРНИТЕ АРХИТЕКТУРИ НА УЕБ-ПРИЛОЖЕНИЯ

BULPROS

AJAX + JSON = ДВИГАТЕЛИ НА ЕВОЛЮЦИЯТА

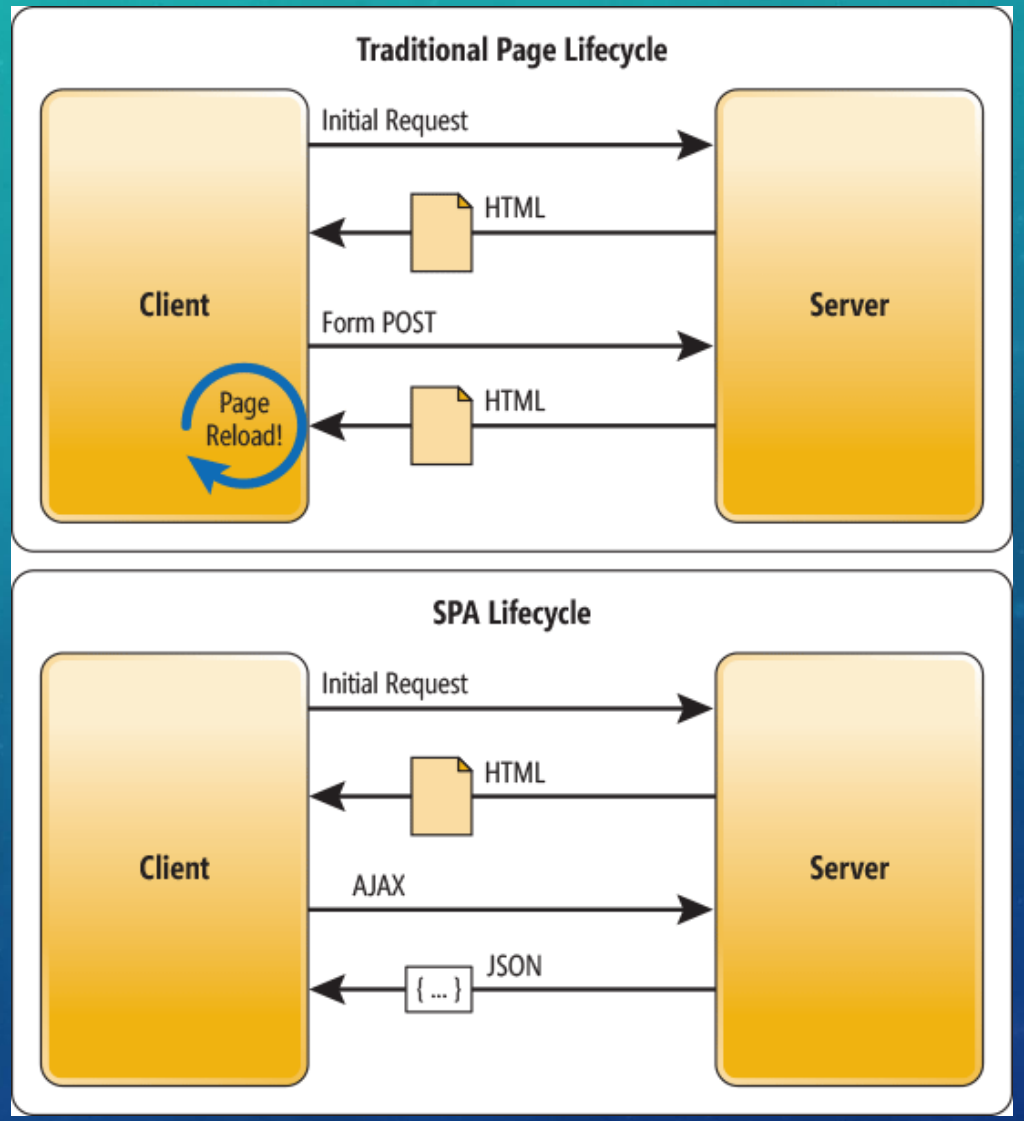


Asynchronous JavaScript and XML (AJAX) – възможност за обновяване на уеб страници посредством асинхронни HTTP заявки до уеб сървъра. Заявките и отговорите могат да пренасят всякакви данни в текстови формати, като: HTML, XML, JSON, суров текст и др.

JavaScript Object Network (JSON) е текстовият формат, в който се сериализират JavaScript обекти. Намира широко приложение при обмен на данни.

SPA И MPA – АРХИТЕКТУРНО СРАВНЕНИЕ

- Жизненият цикъл на **MPA (Traditional Page)** разчита на „генерация“ на HTML страниците в приложенията, извършена от страна на http сървъра. При навигация страниците изцяло се презареждат в уеб браузъра.
- При **SPA** се разчита на еднократно зареждане на главната страница на приложението, като се използват JavaScript рамки (frameworks) за динамична модификация на съдържанието на страницата. Сървърът се използва като хранилище за данни, което се достъпва през RESTful API.



ПРИМЕР 1: MULTI PAGE APPLICATION С VERT.X

```
private static class TemplatesRenderingVertical extends AbstractVerticle {
        private static String templatePath(String templateName) {
            return HTTPConstants.TEMPLATES_DIR + File.separator + templateName + HTTPConstants.TEMPLATE_EXT;
        }
        private static void render(HandlebarsTemplateEngine engine, RoutingContext ctx, String templateName) {
            engine.render(ctx, templatePath(templateName), res -> {
                if (res.succeeded()) {
                    ctx.response().end(res.result());
                } else {
                    ctx.fail(res.cause());
                }
            });
        }
        @Override
        public void start() throws Exception {
            HandlebarsTemplateEngine engine = HandlebarsTemplateEngine.create();
            Router router = Router.router(vertx);
            router.get().handler(ctx -> {
                render(engine, ctx, "index");
            });
            router.post("/say-hi").handler(ctx -> {
                HttpServerRequest request = ctx.request();
                request.setExpectMultipart(true).endHandler(req -> {
                    ctx.put("first-name", request.getFormAttribute("first-name"));
                    ctx.put("family-name", request.getFormAttribute("family-name"));
                    render(engine, ctx, "say-hi");
                });
            });
            vertx.createHttpServer().requestHandler(router::accept).listen(HTTPConstants.SERVER_PORT);
        }
}
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World - Handlebars and Vert.x</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>
<form action="say-hi" method="post">
    <label id="first-name-label">First Name</label>
    <input id="first-name" name="first-name">
    <label id="family-name-label">Family Name</label>
    <input id="family-name" name="family-name">
    <button id="submit">Submit</button>
</form>
</body>
</html>
```

index.hbs

```
<!DOCTYPE html>
<html>
<head>
    <title>Say "Hi"</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>
<h1>Hello {{first-name}} {{family-name}}!</h1>
</body>
</html>
```

say-hi.hbs

BULPROS

ПРИМЕР 2: SINGLE PAGE APPLICATION С VERT.X (1/3)

```
    private static class SinglePageVertical extends AbstractVerticle {
        private static void jsonResponse(RoutingContext ctx, JsonObject jsonObject) {
            ctx.response().putHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            ctx.response().end(jsonObject.encode());
        }
        private static void jsonResponse(RoutingContext ctx, JsonArray jsonArray) {
            ctx.response().putHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            ctx.response().end(jsonArray.encode());
        }
        @Override
        public void start() throws Exception {
            JsonArray names = new JsonArray();
            Router router = Router.router(vertx);
            router.route().handler(BodyHandler.create());
            router.get("/api/names").handler(ctx -> jsonResponse(ctx, names));
            router.get("/api/names/:id").handler(ctx -> {
                int id = Integer.parseInt(ctx.request().getParam("id"));
                jsonResponse(ctx, names.getJsonObject(id));
            });
            router.post("/api/names").handler(ctx -> {
                JsonObject newName = ctx.getBodyAsJson();
                names.add(newName);
                jsonResponse(ctx, newName);
            });
            router.route().handler(StaticHandler.create());
            vertx.createHttpServer().requestHandler(router::accept).listen(8080);
        }
    }
```

HTTPServer.java

ПРИМЕР 2: SINGLE PAGE APPLICATION С VERT.X (2/3)

```
<div data-bind="with: name">
    <label id="first-name-label">First Name</label>
    <input id="first-name" name="first-name" data-bind="value: firstName">
    <label id="family-name-label">Family Name</label>
    <input id="family-name" name="family-name" data-bind="value: familyName">
    <label id="full-name-label">Full Name</label>
    <span id="full-name" name="full-name" data-bind="text: fullName"></span>
</div>
<button id="submit" data-bind="click: addName">Add name</button>
<div data-bind="foreach: names">
    <span data-bind="text: fullName"></span>
    <br>
</div>
```

HTML body section

```
function NameModel(firstName, familyName) {
            var self = this;
            self.firstName = ko.observable(firstName || "");
            self.familyName = ko.observable(familyName || "");
            self.fullName = ko.computed(function() {
                return self.firstName() + " " +
self.familyName();
            });
        }
```

Knockout.js Domain Model

ПРИМЕР 2: SINGLE PAGE APPLICATION С VERT.X (3/3)

```
function NameVM() {
            var self = this;
            var executor = new NamesExecutor();
            self.name = new NameModel();
            self.names = ko.observableArray();
            var getNamesOnSuccess = function(names) {
                console.log(names);
                self.names(ko.utils.arrayMap(names, function(nameItem) {
                    return new NameModel(nameItem.firstName, nameItem.familyName);
                }));
            }
            var getNames = function() {
                return executor.getNames(getNamesOnSuccess);
            }
            var addNameOnSuccess = function(addedName) {
                getNames();
            }
            self.addName = function() {
                executor.addName(self.name, addNameOnSuccess);
            };
            // Load all of the names initially...
            getNames();
        }
        $(document).ready(function() {
            ko.applyBindings(new NameVM());
        });
```

Knockout.js View Model

ПРЕДИМСТВА НА SPA СПРЯМО MPA

- По-добри времена за зареждане на страниците (обикновено се зареждат веднъж и после малки фрагменти)
- Подобрено потребителско изживяване, тъй като обикновено зареждането на данните е асинхронно за потребителя
- Добро разделяне на сървърната логика от тази на клиента – UI може да се пренапише без да се модифицира сървъра
- По-прост модел за разработка на мобилни приложения – може да се преизползва REST API за целите на мобилните приложения
- По-лесни за имплементиране и внедряване на responsive концепции

НЕДОСТАТЪЦИ НА SPA СПРЯМО MPA

- SPA може да изискват тежки клиентски JavaScript библиотеки, чието време за зареждане в браузър е съществено
- Изходният код на UI е видим за всеки, което го прави труден за защита, както и по-труден за дебъгиране
- Search Engine Optimization (SEO) е трудна тема, когато става дума за SPA
  - Search Crawlers разчитат на HTML, който се обхожда и индексира
  - Крайният вариант на SPA HTML се генерира в браузъра -> налага се използване на reverse HTTP proxy и headless browser, който да върне очаквания HTML код, така както би го направил и всеки MPA

ЗАКЛЮЧЕНИЕ

- SPA предоставя възможности за responsive, богато и интерактивно потребителско изживяване на цената на
  - Допълнителна сложност
  - Възможни затруднения с подсигуряване на добра производителност на приложението
  - Затруднения при подсигуряване на SEO
- MPA имат познат модел на разработка, който е утвърден и по-безпроблемен, но този подход води до
  - По-бедно потребителско изживяване
  - По-високо време за достъп до приложението

Макар че SPA е бъдещето, MPA също намира своето широко приложение.

Който и похват да изберете, не ги смесвайте, за да не вземете най-лошото и от двата свята.

Благодаря Ви!

За контакти:

Емил Симеонов, emil.simeonov@bulpros.com

Senior Product Manager & Development Architect