Quality Festival 2019
ISSN 2620-2832

Hroje Puškarić [1]
Aleksandar Đorđević
Miladin Stefanović
Marija Zahar Đorđević

DEVELOPMENT OF WEB BASED APPLICATION USING SPA ARCHITECTURE

Abstract: *Creating a modern application varies considerably from the rules that have been established since the emergence of the Internet. User interfaces evolved from simple forms for data entry into a database to a complex graphical environments that are adaptable to work on any device and number of users. Of the multiple user windows, applications have evolved into single page applications where each application element renders itself on the user's side and thus relieves server work. This approach unambiguously facilitates the development of applications and increases the effectiveness of the application using standard technologies such as development frameworks and mvc models.*

Keywords: *user interfaces, GUI, Single page applications, Angular, MVC model*

1. Introduction

The user interface is part of a system that helps users to communicate with a computer, system or application. Most users of business systems interact with these systems through graphical user interfaces, although in some cases text-based terminals are still used.

The design and development of prototype user interfaces form the final part of the presentation of an application and as such require revision in accordance with all the necessary inputs and outputs.

When designing an information system, it is also imperative to consider aspects related to the design of the user interface, as well as the design of the software itself. System users often evaluate the system based on the interface, not based on its functionality, and often a good user interface design is critical to the success of the whole system. Badly designed interface can cause the user to make great mistakes, and the user interface represents a bottleneck in communication.

A weak designed interface is why many software systems have never been used because the interface is difficult to use and causes a large number of user errors.

It is necessary to create a balance between conceptual design, interaction and presentation of the user interface. The design prototype contains 60% of the overall design process. It decides on the type of data, on the functions and their usability. It determines which objects the graphical interface should have (keys, keyboard, etc.), how to set them, and what is the connection between them.

2. Designing and user interface realization

The user interface should provide the means by which the user can interact with the application to forward inputs and obtain the corresponding output data (Galitz, 2007; Garrett, 2002).

During the process of designing a user

[1] Corresponding author: Hrvoje Puškarić
Email: hrvoje@kg.ac.rs

interface it is necessary to answer certain questions (Isaac, 1995).

Interaction with the user represents 30% of the overall design, and from this aspect it is necessary to ask how the graphical interface "feels" the users? How to activate commands by pressing the button or using the menu? Does the user need to type in the data?

Presentation of the saddlebase represents 10% of the overall design and how the interface looks like? How is information presented? What colors are used? What is the size of the objects? What are the keys?

When defining the user interface, it is necessary to use algorithms (Figure 1) in order for the user to be involved in the software development process to make the user-oriented design because this approach is the design of the user interface where the needs of the user are the most important and that the user is involved in the process prototype interfaces design.

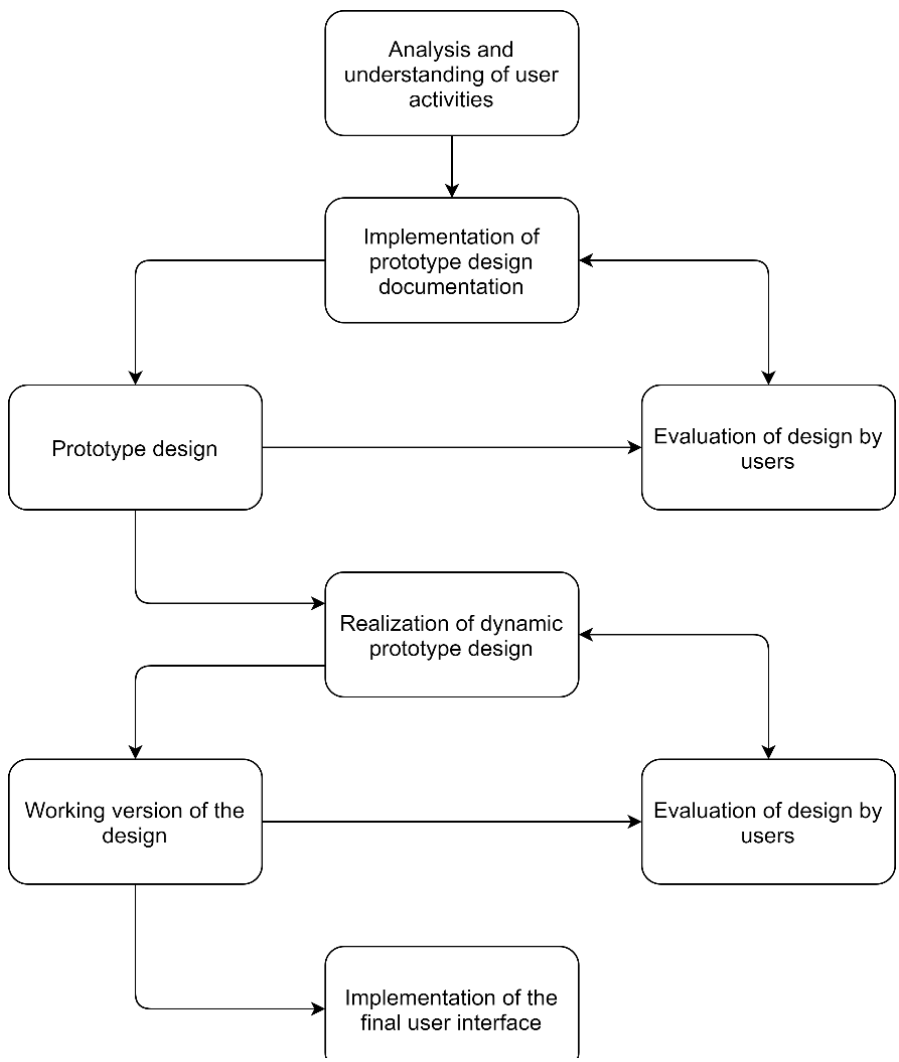

Figure 1. Algorithm for development of user interfaces

The interface should be based on user concepts and concepts rather than on computer concepts (Laurel and Moundford, 1990; Mayhew, 1992; McKay, 2013; Tidwell, 2005; Yu and Shi, 2012).

3. Event management

If certain events are defined, the question is whether if there is a management of that event. So, if there is an event management, it must be executed. Of course, if the event management is not defined, nothing will happen within the application.

Each component of the user interface can generate different types of "events" - "events" when a user uses a keyboard or mouse.

E.g:

- When a user moves the mouse cursor to a button of a user interface, the operating system will register it;
- When a user clicks on, another event will be activated (click event - (click) = "some_event");
- For each generated event, the system will first check if there is a corresponding event handler;
- It is therefore necessary to develop event management to define the activities that the system will take after the event is triggered.

In the program, it is necessary, by calling the addActionListener() method, to establish a connection between the event source and the listener who will react every time that event occurs.

A listener is an arbitrary class object that implements the ActionListener interface. When a button is clicked, an object that represents the event is automatically created. It is a type of action event object. This object is forwarded as argunemnt method actionPerforms().

Two questions should find the answers when it comes to interactive system design, which are:

- How should the information be entered by users in the system?
- How should the information be displayed to the user by the system?

4. Single page appliactions

Single page application (SPA) architecture is relatively new, but very quickly takes the mache among web developers (Fink and Flatow, 2014). Several rather mature SPA development frameworks (Angular, React etc.) have been created, with a very active community that develops add-ons and libraries to help in specific aspects of SPA (Powell & Mikovski, 2013; Klauzinski, 2016; Monteiro, 2014).

SPA is a web application that is delivered to the browser and is not loaded again during use. During that first download, applications load all the resources needed - HTML, JavaScript, CSS, data, etc. - or use them asynchronously, using AJAX calls, to load and install them into the application. Basically, SPA is a client heavy that runs in a browser, distributes over the Internet, and uses the server for permanent data retention.

4.1. Benefits of SPA approach

The roles in the SPA has changed over time. Almost all business logic, as well as the generation of HTML, have been transferred to the client. This causes a significant reduction in the load both of system resources of the server and of the network flow. The client side of the SPA architecture looks more like a standard desktop application: has its status, local data, responds to events and is highly responsive, providing a smooth and dynamic experience for the user. Since the generation of HTML documents is done on the client, the reusability is large and the transition is very

smooth; a generic header can not be generated and plotted again until it changes itself. The server is still very important, it's just in a relaxed commitment. Almost all the server side is focused on data handling: data is permanently stored (usually) in the database; authentication and user authentication is performed, so that it is clear which data may be sent to the client; Validation is performed over all data received from the client, so that only valid data is stored permanently. Although it is usually relieved of business logic, it is not entirely excluded in the SPA that the server performs some tasks that are more appropriate to it than the client, for example, generating files or executing an algorithm that the owner wants to keep secret. Additionally, generating HTML on the server is limited to generating one single page.

5. SPA architecture

SPA consists of two completely separate applications - client and server (Figure 2) (Lowe, 1995; Loosley, 1998). The client application implements the majority of business logic and all communication with the user. The server application mainly serves as a layer for data layer and security implementation (Berson, 1992; Berson 1996). Aside from the data, the client server also serves static files (HTML, JavaScript, CSS), which the client requests as needed. All communication between the server and the client application is done asynchronously. The client side's emphasis is on modularity. Modular code is code separated into independent entities with clearly separate responsibility. Modularity makes delivering parts of the application to the browser easier, and the code itself more comprehensible, easier to test, maintain and change. One major module, usually called a app.module, is needed, which will take care of the client application life cycle and interact with the browser. This module aligns the functioning of the work modules, auxiliary modules, and universal browser interfaces, such as URLs, history, and cookies. It is also responsible for generating HTML, managing the state of the application, and communicating between work modules.

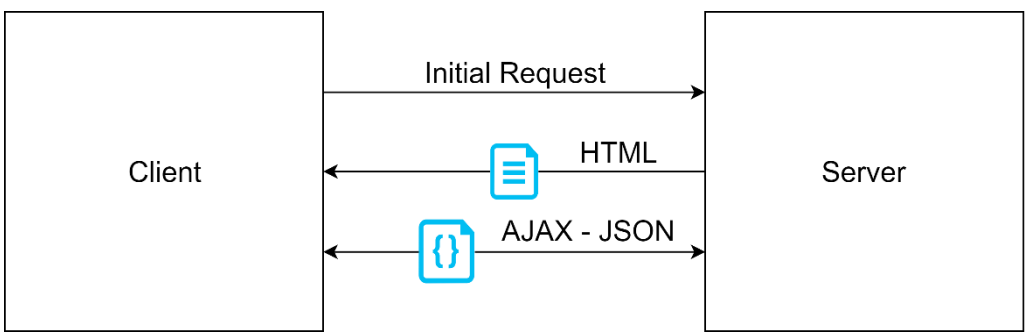

Figure 2. Client-server comunication

Working modules perform a specific business logic and work independently as much as possible. Auxiliary services store data and provide ancillary services, which are often required in several different work modules. Among them, the "context of data" on the client side is distinguished - the service for delivering data from the server, their caching on the client, and the forwarding of working modules upon request. The server application consists of static files and a layered data access system. It is usually organized as a set of API services that access the client data context.

The API call, which successfully passes security checks, accesses the context of the data on the server - the service that communicates with the database - and the client is sent a response in JSON format. The data context on the server retrieves data from the database and prepares them for sending to the client or processes and validates data from the client and places them in the database.

5.1. Angluar framework

Angular represents the platform and framework for creating the client part of the application (fron end) using HTML and TypeScript, while implementing its functionality through the set TypeScript libraries that are imported into the desired application (Freeman, 2017; Freeman, 2018; Frisbie, 2017). In essence, what has already been said about Angular is Typeface, which can not be interpreted by the browser, so it is compiled (translated) into JavaScript code.

Angular applications are modular and Anguar has its own modular system NgModules. Each application can have one or more modules, with small applications typically having one module, usually called a root module, or AppModule, which are composed of complex computers that have some logical interconnection (Kozłowski and Darwin, 2013; Lim 2017; Moisiev and Fain, 2017; Patel, 2014). All these different parts constitute the logical tree of the application.

The most important features of the module are:

- **declaration** - the class of views that belong to the module, which are components, directives, and pipes.
- **export** - a subset of the declaration available to components from other modules.
- **import** - modules whose classes are non-coherent clones that are declared in a given module.
- **providers** - a list of services that a specific module is attached to, and they are available in all parts of the application.
- **bootstrap** - the main view of the application, which is the root component.

The implementation logic is described in Figure 3 and follows from the first called file, which is index.html. Within the main.ts file, call or import app.module.ts with the following import code {AppModule} from './app/app.module' is performed in addition to other imported libraries. The app.module.ts contains the imported root component AppComponentimport {AppComponent} from './app.component'; and within the @NgModulebootstrap: [AppComponent], invoked by the files that make up the root component app.component (app.component.css, app.component.html, app.component.ts, app.component.spec.ts). The basic building unit of the fresh Angular Project is a component. Each component consists of input properties and output events. Each component within the Angularr application appears in the TypeScript class, where the encapsulated logic of the given component is also encapsulated. Within each component, it is also possible to nest in another component. All components are located at their respective locations in the project tree or application. In principle, all components must be declarants inside the module in which they are located. In the case of creating a component, the logic itself is located in the ts file, whereby the convention is followed to call the components in the format ime.component.

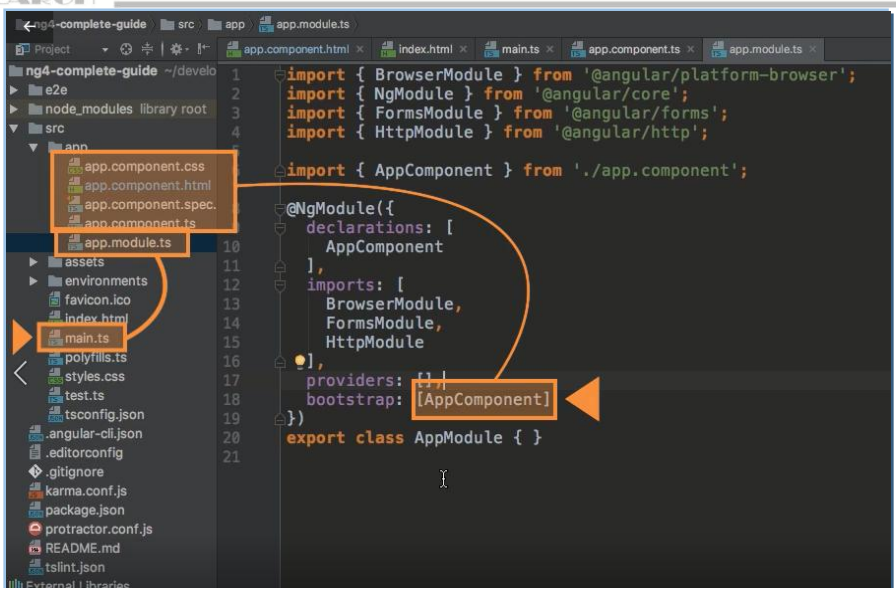

Figure 3. Logic of linking and realization of Angular project

5.2. Communication with a server

As already stated using the Angular framework, SP applications are created, so that once the application is loaded, there should be no new reloading due to sending a request to the server. However, from time to time, it is reasonable to pass on a certain request to the server, primarily to accept or store specific data in the database or to perform certain background calculations. Therefore, connecting to the server is important. A possible option is that the Angular SPA sends a request to the server, so the server at some time following sends an outburst, but in the event the reloading of the Angular application reappears. This is not a good option because in this way all the previously rendered Angular application components and the data contained in them are lost due to reset.

Having said that, it is necessary to forward the request to the server from the application, but so the response will be forwarded to the application, without restarting. This request is sent using Ajax, i.e. Http requests in the background, which uses the XML HttpRequest object.

5.3. MVC model

Model-view-controller allows a flexible structure in which the interface is independent and indirectly associated with application functions, so GUI can be easily customized (Pop and Altar, 2014; Wojciechowski et al., 2004). This allows users to choose or design an interface at will, thus facilitating the designer's work on changing the interface with the development of user needs:

- **Model** - represents a state of the system that can change the model's operations. The model does not have to know who the view and controller are.
- **Controller** - listens and receives a request from the client to perform the operation. Then calls the operation defined in the model, and if the model changes the state, it informs the view of the change of state.
- **View** - provides the user with an interface with which the user enters the data and calls for the appropriate operations to be performed over the model. View displays the state of the model.

6. Validation

Validation is the process of data quality control, ensuring that only precise and reasonable data is permanently stored. Validation helps to prevent mistakes and their spread to other systems.

It is important to apply validation on both the client and the server side: on the client side as a way to quickly notify the user of errors, and on the server side before the permanent storage, because it can never be believed that the data arrived from the client valid. A different problem can cause the server to get error data:

- A program error can disable client validation;
- Some other client application may not contain validation - Web server applications often access multiple different client applications;
- The validity at the time of sending does not have to be valid at the time of arrival at the server;
- A malicious user can try to break the security system or disable the application by dropping the invalid data into the system by inserting SQL query (SQLinjection) or by buffer overflow (Buffer overflow).

7. Conclusion

Just like desktop applications, the SPA client monitors its status, contains local data, responds to events, and interacts with the user. Then when a desktop application searches data from a database, a SPA client searches them out of the server. Because of the speed of loading, a SPA client is usually constructed so that when loading an application from a server to a client, it is loaded exactly how much is needed for the initial functionality of the application: html, basic work and auxiliary modules and data, and the rest is loaded from the server as needed. However, if this particular application requires, a SPA client can be constructed so that the application is readily loaded at the first opening and does not require further communication with the server. This is how the application, once loaded, will be able to work without a connection to the Internet, acting as a desktop application.

Acknowledgment: Research presented in this paper was supported by Ministry of Science and Technological Development of Republic of Serbia: Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.

References:

Berson, A. (1992). *Client/server architecture*. McGraw-Hill.

Berson, A. (1996). *Client server architecture*. New York: McGraw-Hill.

Fink, G., & Flatow, I. (2014). *Pro Single Page Application Development*. Dordrecht: Springer.

Freeman, A. (2017). *Pro Angular*. Apress.

Freeman, A. (2018). *Pro Angular 6*. Apress.

Frisbie, M. (2017). *Angular 2 Cookbook*. Packt Publishing Ltd.

Galitz, W. (2007). *The essential guide to user interface design*. Indianapolis, IN: Wiley Pub.

Garrett, J.J. (2002). *The Elements of User Experience: User-centerd Design for the Web*. Pearson Education.

Isaac, C. (1995). User Interface Design & User Interface Evaluation. *The Computer Journal*, *38*(3), 265-265. doi: 10.1093/comjnl/38.3.265-b

Klauzinski, P. (2016). *Mastering JavaScript Single Page Application Development*. Packt Publishing Ltd.

Kozłowski, P., & Darwin, P. (2013). *Mastering web application development with AngularJS*. Birmingham, UK: Packt Pub.

Laurel, B., & Mountford, S. J. (1990). *The art of human-computer interface design*. Addison-Wesley Longman Publishing Co., Inc.

Lim, G. (2017). *Beginning Angular 2 with TypeScript*. CreateSpace Independent Publishing Platform.

Loosley, C. (1998). *High-Performance client/server*. Wiley.

Lowe, D. (1995). *Client/Server computing for Dummies*. IDG Books Worldwide.

Mayhew, D.J. (1992). *Principles and guidelines in software user interface design*. Englewood Cliffs, NJ: Prentice Hall.

McKay, E. (2013). *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*. Elsevier Inc.

Moiseev, A. & Fain, Y. (2017). Angular 2 Development with TypeScript. Manning Publications Company.

Monteiro, F. (2014). Learning Single-page Web Apploication Development. Packt Publishing Ltd.

Patel, S. (2014). *Responsive web design with AngularJS*. Birmingham, UK: Packt Publishing.

Pop, D. P., & Altar, A. (2014). Designing an MVC model for rapid web application development. *Procedia Engineering*, *69*, 1172-1179.

Powell, J. & Mikowski, M. (2013). *Single Page Web Applications: JavaScript End-to-end*. Manning

Tidwell, J. (2005). *Designing Interfaces*. O'Reilly Media, Inc.

Wojciechowski, J., Sakowicz, B., Dura, K., & Napieralski, A. (2004, February). MVC model, struts framework and file upload issues in web applications based on J2EE platform. In *Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004.* (pp. 342-345). IEEE.

Yu, C., & Shi, Y. (2012). Optimizing Input Efficiency for Graphical User Interface Using Adaptive Cursors. *Journal Of Software*, *23*(9), 2522-2532. doi: 10.3724/sp.j.1001.2012.04239

Hrvoje Puškarić
University of Kragujevac,
Faculty of Engineering
Kraguevac
Serbia
hrvoje@kg.ac.rs

Aleksandar Đorđević
University of Kragujevac,
Faculty of Engineering
Kraguevac
Serbia
coadjordjevic@yahoo.com

Miladin Stefanović
University of Kragujevac,
Faculty of Engineering
Kraguevac
Serbia
miladin@kg.ac.rs

Marija Zahar Đorđević
University of Kragujevac,
Faculty of Engineering
Kraguevac
Serbia
maja_199@yahoo.com