Development of the System to Provide Cross-browser Compatibility of Web Application

Andrew Popov [1], Julia Bilokin [1], Tatiana Solianyk[1], Kateryna Vasylchenko [1]
[1] National Aerospace University KhAI, 17, Chkalova st., Kharkiv, Ukraine, t.solianyk@khai.edu, http://k302.khai.edu/

The problem of providing cross-browser compatibility of a web application have been considered. The existing methods of providing cross-browser compatibility of a web application have been analyzed, their capabilities and limitations are determined. Algorithm for providing cross-browser compatibility, based on the interaction of the proposed system with the developer's file system using node.js, was created. The architecture and structure of the system, which automates the process of providing cross-browser compatibility of the web application using the developed algorithm, was proposed. The developed system was implemented using Electron.js.

Keywords—cross-browser compatibility*; *validity; cross-browser layout; vendor prefixes; task manager

I. INTRODUCTION

With the expansion of the Internet technical capabilities, the requirements for web applications have increased. The ways of displaying data and the concept of web page design have changed; a large number of web browsers have appeared to display them.

Today, the most popular web browsers are Google Chrome, Safari, UC, Mozilla Firefox, Internet Explorer, Opera, etc. Each of them adheres to the general recommendations for page layout visualization, but at the same time, everyone processes the code according to the peculiarities of their own engine.

Despite the existing world standards for building web pages, developers are constantly introducing new technologies and new rules for their implementation in browsers. This allows them to create more dynamic and functional sites, implementing previously unattainable opportunities. Each developer seeks to ensure that his browser supports the latest technologies, new scripts and new libraries. As a result, some users are happy about the new browser, which reflects the most advanced sites well, while others cannot understand why the open web page is not displayed correctly.

The development of a web application includes several stages: the definition of the goals and objectives of the project; development of site structure; development of design layouts; html-layout; programming and quality control; launch and maintenance, as well as SEO-optimization.

After the design and graphic layout of the future web-resource are coordinated, it is time for markup.

The markup means the design of web pages by a way that is understandable for Internet browsers. To do this, HTML codes that specify the logical structure (skeleton) of the web page and cascading CSS style sheets that define its appearance are created. After the work on the site is completed, the customer is provided with technical documentation with layout codes of all pages. This information can be used to improve the web application in the future.

The quality of the layout affects not only the display of the resource in different browsers, but also its perception by search engines. We can say that the development of the site was carried out at a high level, if the designer provided [1]:

- Cross-browser compatibility, i.e. correct and convenient display of web resource pages for the user in different browsers.
- Cross-platform, i.e. adapted web application for easy viewing on any device (from computer to tablet and smartphone).
- Validity – code compliance with current international standards (HTML codes have a logical sequential structure).
- High speed of web pages loading.

Cross-browser layout is such html-code, which is correctly displayed in all existing browsers [2]. Correctness means the exact location of all the layout details and the execution of all the functionality of the web application.

Without cross-browser compatibility, it is possible to lose a huge number of site visitors whose browsers were not taken into account when the web application was created, as well as loss of customer's confidence when the web application does not conform to the approved layout.

The task of providing cross-browser compatibility is quite complicated, since all browsers display the elements of web pages in their own way, and each new version of the browser includes new features that are implemented in browsers of other developers not on the same day.

At present, the technologies for providing the cross-browser compatibility are still in progress. For example, Bootstrap 4 framework provides cross-browser compatibility for the latest versions of browsers [3].

For earlier versions of browsers, the task of providing cross-browser compatibility is relevant.

The easiest way to ensure cross-browser compatibility is to use elements that are the same in all the required browsers when you make the html code. However, this limits the functionality of the web application.

The second way is to write so-called "hacks" – sets of special selectors or rules that are understandable only for a certain browser and ignored by others. The application of this approach is irrational with the support of several dozen different browsers and their versions and may be appropriate only in exceptional cases [4].

Another way to provide cross-browser compatibility is to use vendor prefixes. Browser vendors for experimental CSS properties that are not yet accepted in the standard use such prefixes.

For different browser engines there are different vendor prefixes. For example, "-o-" for Opera, "-moz-" for Firefox, "-ms-" for Internet Explorer, "-webkit-" for Safari, Google Chrome and UC for Android, etc. [5].

In the process of layout, developers perform a large number of the same operations, which allows applying to the process of various means of its automation.

II. Automation of Layout for Providing Cross-Browser Compatibility

At the stage of web page layout, special programs – task managers – are used to automatically perform repetitive routine processes. These processes include mining, CSS processing, concatenation of files, addition of vendor prefixes, and so on. [6].

There are exist more than 10 task managers, which can be used to solve this problem [7]. Grunt and Gulp are the most popular task managers, which are implemented in the scripting language Javascript. They are used on various platforms and function in Node.js [8].

Grunt is a command-line tool for running predefined repetitive tasks. These tasks are defined declaratively with configuration objects, which are handled by enormous numbers of plugins to perform various tasks at the same time. Because of this, the complexity of Grunt applying increases as well as the project configuration grows. Comparing Grunt vs Gulp, the core difference is that Gulp defines tasks as JavaScript functions. It builds on top of the piping concepts (sources, filters, sinks). Despite of the fact that it provides faster performance due to the stream usage and in-memory operations and requires less amount of code, comparing to Grunt, initially streams are hard to understand and the developer has to be proficient in Node.js.

The choice mainly depends on your team's skill sets and working experience, but usage of existing task managers is associated with a number of difficulties:

- It is necessary to integrate the task managers into every web application that is being developed. This is done using the npm console. This requires the presence of Node.js, as well as the addition of necessary plug-ins with npm and package.json;
- It is need to create certain tasks for each project, but there is no visual interface to this process. Cross-browser layout is carried out by changing a special file in the code editor.

This article is devoted to the development of an alternative tool for automating the process of providing cross-browser compatibility at the stage of layout in order to improve its efficiency.

The proposed system allows:

- Verify the correctness and validity of the written code;
- Minify and concatenate files;
- Compile CSS;
- Add vendor prefixes;
- Track changes in files in real time.

A distinctive feature of this system is:

- One-time installation of the application by activating the .exe file;
- The developer does not need to write certain tasks for each project;
- The developer does not need to know the language of JavaScript;
- The developer only needs to select the project folder specify the files to be compiled and minify, and specify the path to the compiled file.

III. Development of the System for Providing Cross-Browser Compatibility of the Web Application

A. System architecture for providing cross-browser compatibility of the web application

The developed system is a cross-platform desktop application written using Electron.js.

Electron is the runtime for desktop applications in the extended JavaScript language. This application provides interaction with various APIs to ensure the completeness of the functionality.

The entry point for Electron.js is the main file (called main.js), which is defined in the "main" property of the configuration file (package.json). This file is executed when the application is launched. In this file, windows are created that render and display web pages with the additional ability to interact with the native GUI of the operating system.

The generalized architecture of the developed system is shown in Figure 1.

The main process runs the main script, is responsible for interacting with the native GUI of the operating system and creates a graphical interface for the web application (application windows).

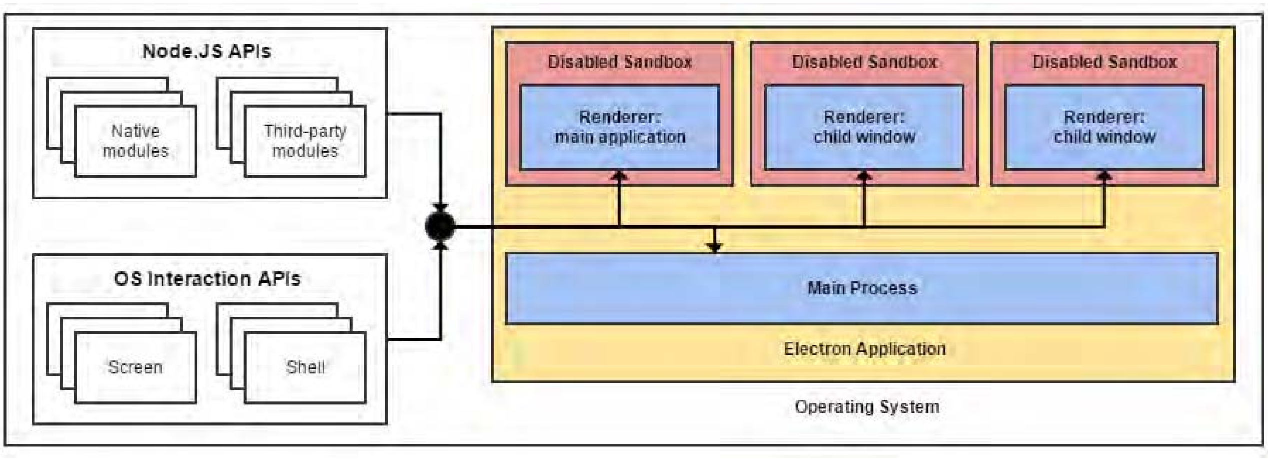

Figure 1. The architecture of the system for providing cross-browser compatibility of the web application

Initially, application windows are not provided to developers. The main process in the main file using the BrowserWindow module creates them.

Electron uses Chromium to display web pages and provide a multiprocessor architecture. Each web page in Electron runs in its own renderer process. This rendering process displays a web page in the application window.

In a regular browser, web pages are launched in a closed environment (so-called "sandbox") and do not have access to native resources. Application Electron allows using the Node.js API on web pages, providing access to interaction with the operating system at a low level [9].

Directly in the context of web pages and Angular.Js, the application is based on the MVC architecture [10].

B. Description of system functionality

A class diagram that describes the logical model of a system for cross-browser compatibility of web applications at the stage of layout is presented in Figure 2.

This diagram describes the classes and shows the relationships between them. Six classes are involved:

- "Project" class (attributes: Name, Path, Date);
- "File" class (attributes: Name, Path, Type);
  ProjectTree class (attributes: Project, operations: ShowFileTree);

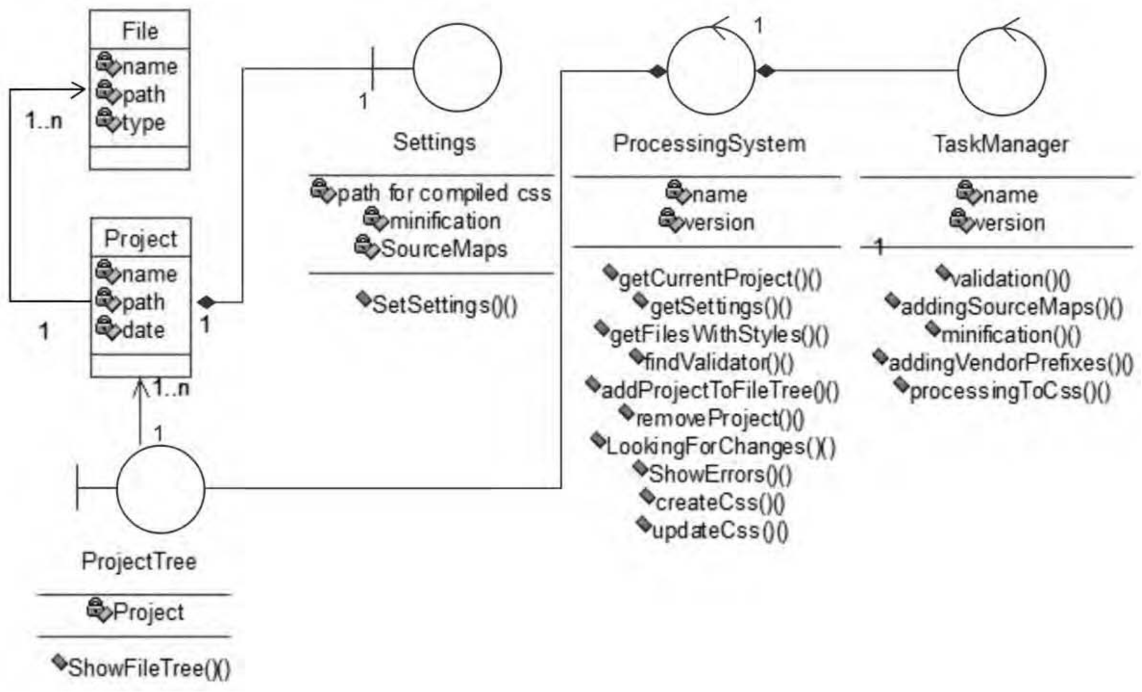

Figure 2. Class diagram

- "Settings" class (attributes: Path for compiled css, minification, SourceMaps; operations: SetSettings);
- TaskManager class (attributes: Name, Version; operations: Validation, AddingSourceMaps, Minification, AddingVendorPrefixes, ProcessingToCss);

"ProcessingSystem" class (attributes: Name, Version; operations: GetCurrentProject, GetSettings, FindValidator, AddProjectToFileTree, RemoveProject, LookingForChanges, GetFilesWithStyles, ShowErrors, CreateCss, UpdateCss).The behavior of the system can be described at the level of individual objects that exchange messages among themselves in order to reach the desired goal or implement some service.

Using the cooperation diagram, we can describe the full context of interactions as a kind of a temporary "slice" of a set of objects interacting with each other to perform a specific task or business goal of the software system.

The collaborative diagram of the system for cross-browser compatibility of web applications at the stage of layout is presented in Figure 3.

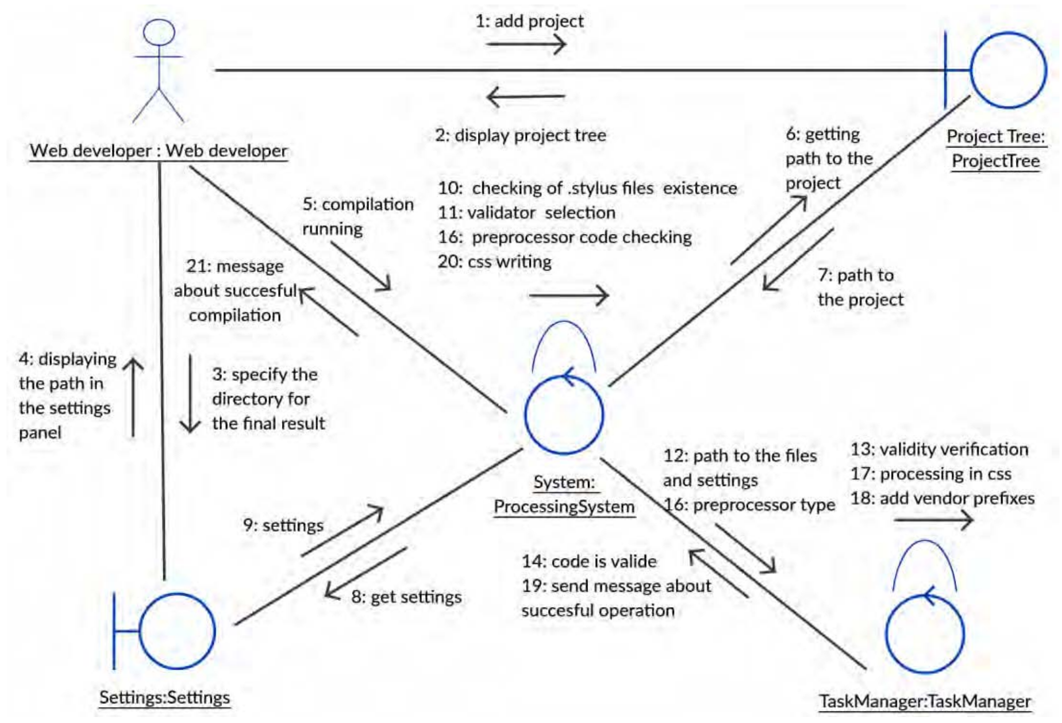
1: add project
2: display project tree
Web developer : Web developer
Project Tree: ProjectTree
6: getting path to the project
7: path to the project
5: compilation running
10: checking of .stylus files existence
11: validator selection
16: preprocessor code checking
20: css writing
21: message about succesful compilation
4: displaying the path in the settings panel
3: specify the directory for the final result
System: ProcessingSystem
12: path to the files and settings
16: preprocessor type
13: validity verification
17: processing in css
18: add vendor prefixes
9: settings
8: get settings
14: code is valide
19: send message about succesful operation
Settings:Settings
TaskManager:TaskManager

Figure 3. Collaborative diagram

Objects: actor – web developer, separator classes – ProjectTree, Settings, controller classes – ProcessingSystem, TaskManager.

The actor adds a project, after which "ProjectTree" class displays the project tree. The actor sets the path for the result; "Settings" displays this path in the settings panel. The actor starts the compilation; "ProcessingSystem" class gets from the "ProjectTree" path to the project, as well as settings from the "Settings". Next, in "ProcessingSystem", the presence of files with styles is checked; the selection of the appropriate validator is implemented. "ProcessingSystem" passes to the "TaskManager" the path to the files and entered settings (path to the result). "TaskManager" checks the validity of the code, reports the results in the "ProcessingSystem". "ProcessingSystem" searches for the preprocessor code in the project, defines the corresponding type of preprocessor and passes it to "TaskManager". "TaskManager" performs processing in .css, adding vendor prefixes and reports progress in "ProcessingSystem". "ProcessingSystem" queries the .css and sends a message about the successful recording to the actor.

C. Structure of the system for providing cross-browser compatibility of the web application

The structure of the developed system consists of several software modules that interact with each other (Figure 4). Among them:

- Module for building a project tree;
- scss module for compiling .scss files;
- less module for compiling .less files;

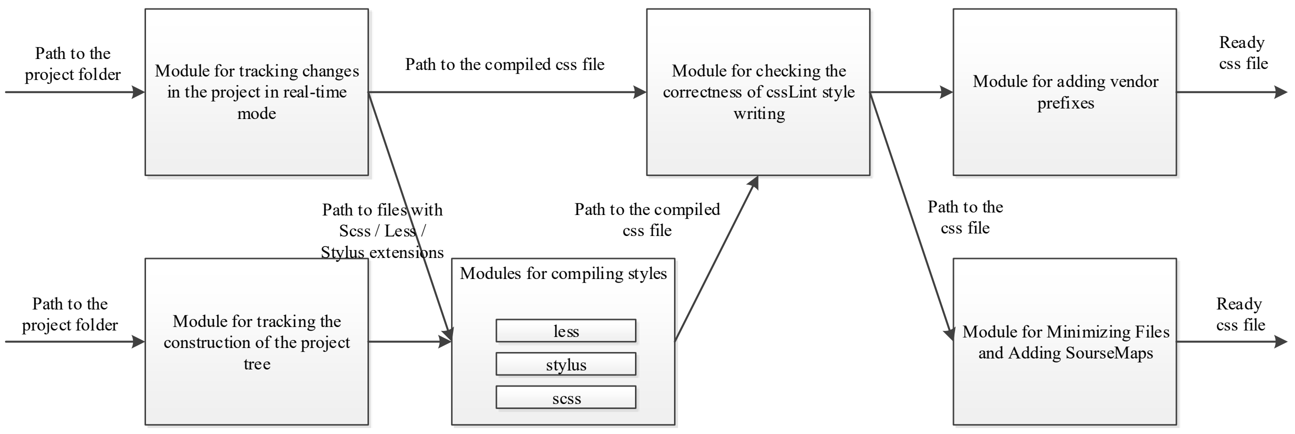
Path to the project folder
Module for tracking changes in the project in real-time mode
Path to the compiled css file
Module for checking the correctness of cssLint style writing
Module for adding vendor prefixes
Ready css file
Path to files with Scss / Less / Stylus extensions
Path to the compiled css file
Path to the css file
Path to the project folder
Module for tracking the construction of the project tree
Modules for compiling styles
less
stylus
scss
Module for Minimizing Files and Adding SourseMaps
Ready css file

Figure 4. The structure of the system for providing cross-browser compatibility of the web application

- stylus module for compiling .stylus files;
- Module cssLint to verify the correctness of writing styles;
- W3C-validator of html-code;
- Module for adding vendor prefixes;
- Module for minimizing files and adding SourceMaps;
- Module for tracking changes in the project in real-time mode.

D. *Algorithm of the system for providing cross-browser compatibility of the web application*

The algorithm of the system consists of the following steps (Figure 5).

Step 1. Select the folder in which the project is located.

Step 2. Add the project to the list of projects. It contains all the projects previously downloaded by the developer, until they are removed from the system.

The last project the developer worked on is by default active until the new project is added or another project is selected from the list.

For the active project, the project tree is displayed, which contains all project folders and files with additional settings.

Step 3. When you select a file (folder) for compilation, its type is checked. If the selected file is a file with the styles (.css, .scss, .less, .stylus), the code validator is selected, which corresponds to the given file type, then the validity of the code is checked. If the code contains errors, an error message is displayed. If the code is valid, then a message is displayed about the successful verification, and, in the case of the preprocessor code (.scss, .less, .stylus), it is processed in .css. If the preprocessor code is not found, it proceeds to step 4.

For the active project, the project tree is displayed,

Step 4. For the css-file, the vendor prefixes are added.

Step 5. Additional settings are read. If the appropriate options have been selected, the file is minified and Source Maps added.

Step 6. The css-file is created or overwritten (if this file exists in the project folder) in the directory specified in the advanced settings.

Step 7. The system tracks changes in the selected file (folder). If changes are detected, then the algorithm is repeated from step 3.

IV. CONCLUSIONS

At present, the technologies for providing the cross-browser compatibility are still in progress.

Bootstrap 4 framework provides cross-browser compatibility for the latest versions of browsers. For earlier versions of browsers, the task of providing cross-browser compatibility is relevant.

There is no the only right or wrong answer when it comes to the selection of build tools for next projects. The choice will mainly depend on the size of a project, developers' team skill set and whether you are more interested in coding (Gulp task runner) or configuring (Grunt build tool).

The proposed system is a desktop application that requires a one-time installation. To implement the program, the Electron runtime was chosen, which allows creating cross-platform applications based on native javascript.

The basic logic of the system is written in AngularJS – JavaScript framework with open source. Directly in the context of web pages and AngularJs, the application is based on the MVC architecture. The main purpose of applying this concept is to separate the business logic (model) from its visualization (presentation, view). This division increases the possibility of reuse. To create the user interface of the application, the HTML hypertext markup language and the CSS-based metalanguage SASS are used, which is designed to increase the level of abstraction of CSS code and simplify the files of

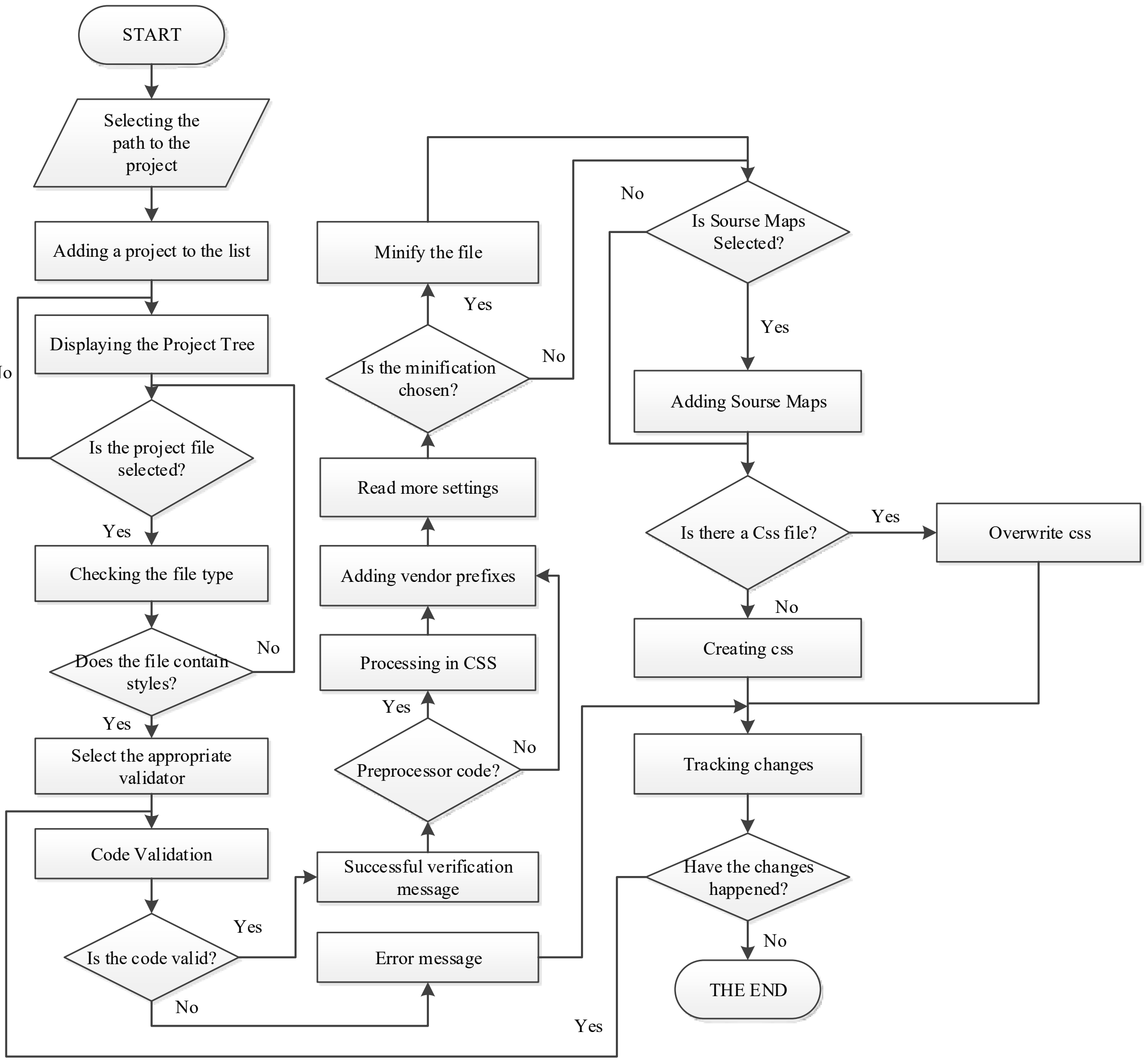

Figure 3. Block diagram of the system algorithm

cascading style tables. To store the data of projects added to the system and information about them, the JSON format is used, which serves to represent objects in the form of a string. After the environment and technology of the software product development are chosen, the structure of the system is determined, which reflects the interaction of the program modules with each other.

A distinctive features of this system are: one-time installation of the application by activating the .exe file; the developer does not need to write certain tasks for each project; the developer does not need to know the language of JavaScript; the developer only needs to select the project folder specify the files to be compiled and minify, and specify the path to the compiled file.

References

[1] R. Curedale, *Web Design Briefing Checklist*. Design Community College, 2016.

[2] T. Felke-Morris, *Web Development and Design Foundations with HTML5*, 9th ed, Pearson Education, 2018.

[3] J. Kramer, *Which Responsive Design Framework Is Best? Of Course, It Depends*, https://www.smashingmagazine.com /2017/03/which-responsive-design-framework-is-best/, March 20, 2017.

[4] A. Boehm and Z. Ruvalcaba, *Murach's HTML5 and CSS3*, 4th ed, Mike Murach & Associates Inc, 2018.

[5] B. Jobsen and A. Meyghani, *Less Web Development Cookbook*, Packt Publishing, 2015.

[6] B. Dayley, B. Dayley and C. Dayley. *Node.js, MongoDB and Angular Web Development: The definitive guide to using the MEAN stack to build web applications*, Pearson Education, 2017.

[7] What are the best Node.js build systems/task runners?, https://www.slant.co/topics/1276/~node-js-build-systems-task-runners, April 26, 2018.

[8] J. Cryer. *Pro Grunt.js.* Berkley, US: aPress, 2015.

[9] J. Muhammed. *Building Cross-Platform Desktop Applications with Electron*, Packt Publishing, 2017.

[10] J. Wilken, D. Aden and J. Aden. *Angular in Action.* Manning Publications, 2018.