Kabakuş, Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 30(2):141-149

Erciyes University Journal of the Institute of Science and Technology
Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi
ISSN 1012-2354
Cilt (Volume): 30, Sayı (Issue): 2, Mart/March-2014
http://fbe.erciyes.edu.tr/

JavaScript için nesne yönelimli programlama yaklaşımları

A. Talha KABAKUŞ[1]

[1]*Abant İzzet Baysal Üniversitesi Bilgi İşlem Daire Başkanlığı Merkez 14280 Bolu*

Anahtar Kelimeler: JavaScript, MVC, Nesne Yönelimli Programlama, Üç Katmanlı Mimari, Web Tabanlı Mimari

ÖZET

Bu çalışmada web tabanlı uygulamalarda MVC (Model-View-Controller) mimarisinin ve nesne yönelimli programlama yaklaşımlarının kullanılmasının yazılım geliştirme süreçlerine olan pozitif etkisi ele alınmıştır. MVC mimarisi, uygulama katman ve nesnelerini birbirinden ayırarak yazılım mimarisinin sadeleştirilmesini sağlamaktadır. JavaScript web tarayıcıların doğal dili olduğundan doğrudan tarayıcılar tarafından yorumlanabilmektedir. Bundan dolayı JavaScript web tabanlı uygulamalar için en popüler ve en etkili betik dilidir. Nesne yönelimli programlama, programlama nesnelerinin gerçek dünya nesnelerine benzetilerek problemlerin çözülmesini sağlar. JavaScript, kendi yapısı itibariyle nesne yönelimli yaklaşımları ve MVC mimarisini desteklememektedir. Sencha Ext JS, nesne yönelimli programlama yaklaşımlarından ve MVC mimarisinden faydalanarak zengin web uygulamaları geliştirmeyi sağlayan bir kütüphanedir. Performans, web tabanlı uygulamalar için her zaman kritiktir. Bu sebeple web sayfalarında tanımlı olan kaynak boyutu, performans artışı sağlamak için mümkün olduğunca küçültülmelidir. Bu küçültme hem kaynak sayısının azaltılması hem de kaynakların sıkıştırılması ile yapılabilmektedir. Kalıtım, JavaScript'in doğrudan desteklemediği nesne yönelimli programlamanın önemli özelliklerinden birisidir. Gerçek dünya nesnelerinin hepsi birbirinden türediğinden kalıtım kullanılmadan gerçek dünya nesnelerini web tabanlı uygulamalarda kullanmak mümkün değildir. Dinamik kaynak yükleme, web tabanlı uygulamalar için oldukça yeni bir yaklaşım olup, kaynakların gerektikçe yüklenmesini sağlamaktadır. Çalışma boyunca elde edilen tecrübeler ve sonuçlar, nesne yönelimli yaklaşım ve MVC mimarisinin yazılım geliştirme süreçlerini kolaylaştırdığını, hem geliştirme hem de üretim aşamalarında klasik web uygulama geliştirme yaklaşımlarına göre büyük performans kazancı elde edildiğini göstermektedir. Ayrıca bu yaklaşımların kullanılmaması durumunda, özellikle büyük çaplı yazılım projelerinin bakımının ve revizyonunun çoğu zaman mümkün olmadığı gözlemlenmiştir. JavaScript yapı olarak nesne yönelimli programlama yaklaşımlarına uygun olduğundan bu yaklaşımların JavaScript'e uyarlanmasının web tabanlı uygulamalara performans ve yetenek kazandıracağı düşünülmektedir.

Object-oriented programming approaches for JavaScript

Key Words: JavaScript, MVC, OOP, Three-tier architecture, Web-based architecture

ABSTRACT

In this study, web application development with MVC (Model-View-Controller) architecture and object-oriented programming approaches is discussed in order to show how much these approaches simplify web application development. MVC simplifies application architecture by separating application domain layers and objects. Because of JavaScript is the native language of web browsers that can be directly interpreted by them, it is currently most popular and effective scripting language for web applications. Object-oriented programming (OOP) is the paradigm of using objects a representation of real-world objects to solve problems. JavaScript doesn't have a built-in (*native*) support for both OOP and MVC approaches. Sencha Ext JS is a complete RIA (Rich Internet Application) development library makes available to use most of OOP principles with MVC architecture for JavaScript. Performance is always critical for web applications, so its required resource size must be minimized in order to boost its performance. This minimization can be done through eliminating unnecessary sources and source compression. Inheritance is also another key object-oriented programming feature that JavaScript doesn't support. Because of everything in real-life extends from something, it is impossible to define real-life objects for web applications without inheritance. Results and experiments show that OOP and MVC approaches simplified development stages and offered big performance gains for both development and production stages than classic web application development paradigms. Also without these approaches, most of time it becomes impossible to maintenance and revise software projects especially big scaled ones. JavaScript has suitable structure for object-oriented programming approaches, so its approaches can be imitated for JavaScript to improve performances of web applications and extend their capabilities.

Sorumlu yazar (Corresponding outhor) e-mail:

1. Introduction

Web has become more and more important for last 10 years because of its nature. There are reasonable technical and social reasons for this huge growing up. First of all, web is a public area for people all over the world without any requirements except than a web browser. Web has changed the communication way people does, opened information access and share to all over the world. With development of software technologies, web applications architecture is also developed a lot. Rich Internet Application (RIA) defines web applications which has quite similar user interface (ui) like desktop applications that offers powerful and modern components on web. Google trends about RIA technologies shows (as November 2013) that plugin based frameworks are in the process of being replaced by HTML5 and JavaScript based alternatives. Model–View–Controller architecture is a design approach of dividing the software into Model, View and Controller component to better control the software quality with respect to processing, and interface design (Hasan & Isaac 2011). MVC follows the most common approach of layering. Layering is nothing but a logical split up of our code in to functions in different classes (Uyun & Rifqi 2010). Major benefits of MVC architecture can be listed as enhanced maintainability and extensibility of the system, potential multiple views of the same model, pluggable views and controllers, synchronized views and re-usability (Varma 2009).

JavaScript is directly interpreted by web browser. JavaScript's natural association with browsers makes it one of the most popular programming languages in the world (Crockford 2008). JavaScript has some lacks of ability that is available for object-oriented programming languages. These lacks can be overcome by imitating object-oriented approaches.

Usability is always very critical for software applications and it becomes even more critical if the software is a web application. In order to reach as many users as possible and satisfy user contentment, web applications must offer a user-friendly interface and be responsive. Performance is always the most critical issue for thin client applications through its architecture. There are many metrics for performance measurement including response time, render speed, resource load management. Web applications consist two main layers which are presentation and business (*application*) layer. Presentation layer is responsible for views and interactions with users at clients. This layer is also known as “front-end”. The other layer is the layer of users interactions are handled and responded to server. This layer is also called as “backend”.

This paper is organized as follows: Section 2 presents each MVC layer with details. Section 3 presents performance improvements and features enabled by OOP approach. Section 4 presents key findings of experiments and conclusions.

2. Materials and Methods

Web applications run on web browsers and performance is a big issue because of its nature. Web browsers interpret languages like JavaScript, CSS (Cascading Style Sheets) and
Best-practices recommend separating client-side and server-side responsibilities to gain performance. Also it's strongly recommended doing validations at client-side in order to prevent server from unnecessary calls and performance gain. Sencha Ext JS[1] is an open-source rich internet application (RIA) library that offers rich, powerful and modern user-interface widgets with cross platform browser compatibility with rich validation controls. With its latest release *(Ext JS 4)*, Sencha Ext JS supports MVC architecture which allows defining models, views and controllers separately like commonly used approach for backend (server-side) implementations as it is shown in Figure 1. This brand-new architecture for JavaScript offers easily organize, maintain the source code with decreasing the amount of code. One of the benefits of using the MVC architecture is re-use of code (Loiane Groner 2012).

One of the biggest advantages of using Ext JS is loading resources dynamically (*on demand*). This means that Ext JS automatically loads sources (classes) when they are necessary which provides big performance gains through loading only required sources and when they are required. Ext.require is the keyword of required class definition when defining new classes. Ext.Loader class is responsible for loading sources through dependencies. As it is shown in Table 1, dynamic loading decreases page loading time by 525%.

Table 1. Required source sizes per each page loading approach

Approach	File Size (KB)
Classic way (Directly importing all sources)	1500
Dynamic loading	240

The main new concept in Ext JS 4 is trying to imitate object-oriented programming (OOP) concepts for JavaScript as much as possible which isn't available with pure JavaScript. Inheritance is the ability of creating new classes by extending existing class(es) to reuse code of existing class without defining again. Inheritance is one of the new OOP features which are available with Ext JS 4. This is also brand-new concept for JavaScript which the language itself doesn't provide. In Ext JS 4, everything is inherited from Ext.Base class. Ext.define is the keyword of creating new classes with inheritance. Inheritance is done through “extends” configuration while defining new classes. Inheritance reduces total lines of code (LoC) and offers central management for extended classes through their bases classes. JavaScript currently only support inheritance through use of prototype chaining. The basic idea behind prototype chaining is to use the concept of prototypes to inherit properties and methods between two reference types (Zakas 2012).

This approach doesn't offer multiple inheritance which can be done through Ext JS 4. Also at the same time, an extended class can have mixins to include previously defined behaviors and configurations on other classes. JavaScript MVC[2] is another JavaScript library based on MVC architecture. Its inheritance model is based on strings. So it doesn't offer a real package based inheritance. In Ext JS 4, dependencies can be added one by one like Ext.grid.feature.Grouping or as a group by using wildcard character (*) like Ext.grid.* as it is commonly used with object-oriented programming languages. Another major difference is Ext JS offers freedom to work with 3rd party libraries through its extensible architecture. Wrapper components for 3rd party libraries can be easily created by extending Ext.Component class. Leaflet is an open source mapping JavaScript library. A sample wrapper component source code for Leaflet map is shown in Figure 1.

```
Ext.define('Ext.ux.LeafletMapView', {

        extend: 'Ext.Component',

        alias: 'widget.leafletmapview',

        config:{

                map: null

        },

        afterRender: function(t, eOpts){

                this.callParent(arguments);



                var leafletRef = window.L;

                if (leafletRef == null){
```

Figure 1. Wrapper component for Leaflet

Mixin is another brand-new feature for JavaScript. “mixins” offer which is reusable sets of behavior and configuration that can be ‘mixed in’ to a class (Spencer 2011). It can be defined as modern Object Oriented Programming pattern that allows for multiple inheritance (Garcia et al. 2013). A programmer implements mixins in exactly the same way as a derived class, except that the programmer cannot rely on the implementation of the mixin's superclass, only on its interface (Flatt et al. 1999). There are some works to support transitive mixins (“mixins of mixins”) as a way to model the interdependencies among different mixins (Neumann et al. 2007). Sencha Ext JS 4 offers to develop good-looking, modern web user interfaces including all of these advantages. Figure 2 illustrates Ext JS 4 MVC architecture.

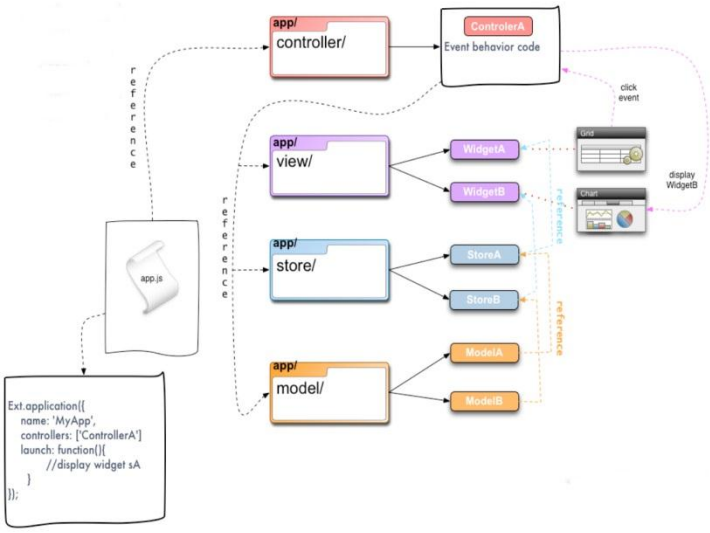

Figure 2. Ext JS 4 MVC architecture diagram (Loaine Groner 2012)

Mixin is another brand-new feature for JavaScript. “mixins” offer which is reusable sets of behavior and configuration that can be ‘mixed in’ to a class (Spencer 2011). It can be defined as modern Object Oriented Programming pattern that allows for multiple inheritance (Garcia et al. 2013). A programmer implements mixins in exactly the same way as a derived class, except that the programmer cannot rely on the implementation of the mixin's superclass, only on its interface (Flatt et al. 1999). There are some works to support transitive mixins (“mixins of mixins”) as a way to model the interdependencies among different mixins (Neumann et al. 2007). Sencha Ext JS 4 offers to develop good-looking, modern web user interfaces including all of these advantages. Figure 2 illustrates Ext JS 4 MVC architecture.

2.1. Model Layer

The model is where all the application’s data objects are stored. A model doesn’t know anything about views or controllers. The only thing a model should contain is data and the logic associated directly with that data. Any event handling code, view templates, or logic not specific to that model should be kept well clear of it. You know an application’s MVC architecture is violated when you start seeing view code in the models (MacCaw 2011). With MVC support, we can use database tables as JavaScript objects (models) as Figure 3 illustrates each model per layer. Model is collection of fields that provides to define each field with the associated type and defining associations like belongsTo, hasOne, hasMany (Boerman 2012). Also it's possible to add validations to each field like minimum & maximum length, required fields, regex expressions, etc. This approach keeps server away from unnecessary calls with handling these validations which are executed against validator functions at front-end before backend does. With Ext JS 4, we are able to add validations through other user-defined models in order to associate objects within others.

This is another great sample of object-oriented programming paradigm offered for JavaScript. Most of JavaScript libraries offer validation through just primitive type (string, integer, boolean, double, etc.) values.

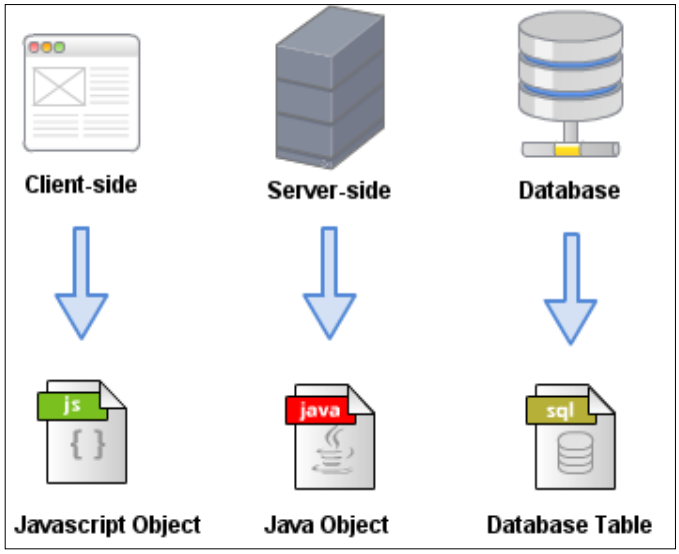

Figure 3. Representation of domain objects of each layer

2.2. View Layer

View classes describe all user-interface components that can be reused. Ext JS provides many powerful, rich and modern widgets that can be customizable and expendable. View layer is only responsible for application view objects that are going to be rendered. Ext JS offers many useful built-in functionalities for view components. For example, a grid component has built-in features like sorting, searching, grouping, filtering, etc. This is also a big advantage for developers to concentrate on application business instead of implementation details. MVC approach separates all business logic like event handling, user interactions, etc. from view classes.

2.3. Controller Layer

All controls and user interactions on view objects are defined in this layer regards to three-tier architecture. With this approach, same view objects can be used for different business operations with their own behaviors (*this is also known as re-usability*). Controller is the place where business logic goes. All dynamic actions are defined in controllers like event handlers, button actions, etc. ComponentQuery is another new feature which provides searching user-interface components by their type with a similar syntax to a CSS selector. With this singleton class, we are able to define actions towards to all selected components. For example, an action like "load all stores in the page/panel" or "disable all buttons in the page/panel" can be defined by ComponentQuery class with a query that selects all grid components in the view. With this feature once a behavior of a view object is defined, this can be applied to all instances of it.

Despite it is not mandatory, the recommended folder structure is shown in Figure 4. A new folder is created for each layer. Folder paths are fully configurable and should match with defined packages. For example, a class like "MyApp.view.user.RegistrationForm" should be located in "MyApp/view/user" folder with "RegistrationForm.js" file name.

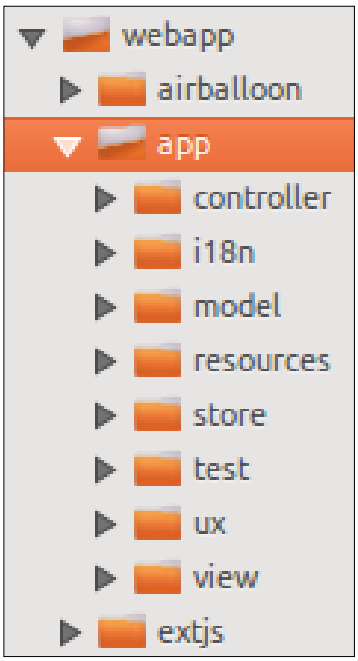

Figure 4. Recommended folder structure for Ext JS applications

3. Results and Discussion

OOP and MVC architecture does not only simplify software development stages; but also improves software performance. For a web application, these approaches are experienced during this study. In the following sections, each performance improvement and feature is discussed.

3.1. Resource Compression and Dynamic Loading

A tool named Sencha Cmd (*Command*)[3] can be used to compress resources and minify their sizes to improve page loading speed. After JavaScript source compression using Sencha Cmd for an application that uses all sources, required source size becomes 850 KB which offers almost 100% performance improvement than classic way as it is shown in Table 2. This tool also excludes unnecessary sources before source compression.

Table 2. Compressed file size

Approach	File Size (KB)
Source size without compression	1500
Source size with compression	850

1 http://www.sencha.com/products/sencha-cmd

Compression minimizes resource size that is going to be loaded. This helps server to load resources more quickly and making views more responsive. Another filter named GZIP[4] compression is used to compress responses from server and decompress them after it is loaded into browser. GZIP compression is standardized respectively as RFC1952[5] by IETF[6] (Internet Engineering Task Force). Almost all web browsers accept GZIP encoding for compression by looking HTTP (Hyper Text Transfer Protocol) Response Header "Content-Encoding" property as it is illustrated in Figure 5.

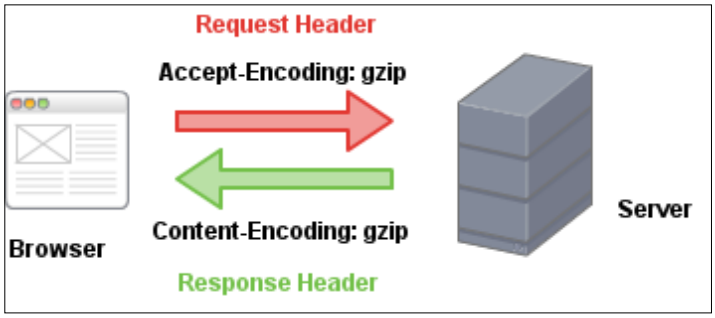

Figure 5. A diagram that illustrates how GZIP Compression works

3.2. Internalization

Internalization (i18n) is the process of planning and implementing products and services so that they can easily be adapted to specific local languages. Internalization can be applied by using resource bundle implementation. Different languages can be easily added by just adding its resource bundle into the system. A resource bundle is simply a properties file that contains key-value pairs which keys contain variables and value contains language specific equivalents of these variables as it is shown in Figure 6. Resource bundles contain locale based information and provide separating this information from source code. The content of interface can be changed by just editing these resource bundles without making any changes to source code. This approach brings together all language specific content and prevents users from possible mistakes that can occur while editing source code. Figure 7 shows Turkish and English user interface localized through using resource bundles.

grid.actions=İşlemler success=Başarılı info=Bilgilendirme	grid.actions=İşlemler success=Başarılı info=Bilgilendirme

Figure 6. Samples from Turkish and English resource bundles

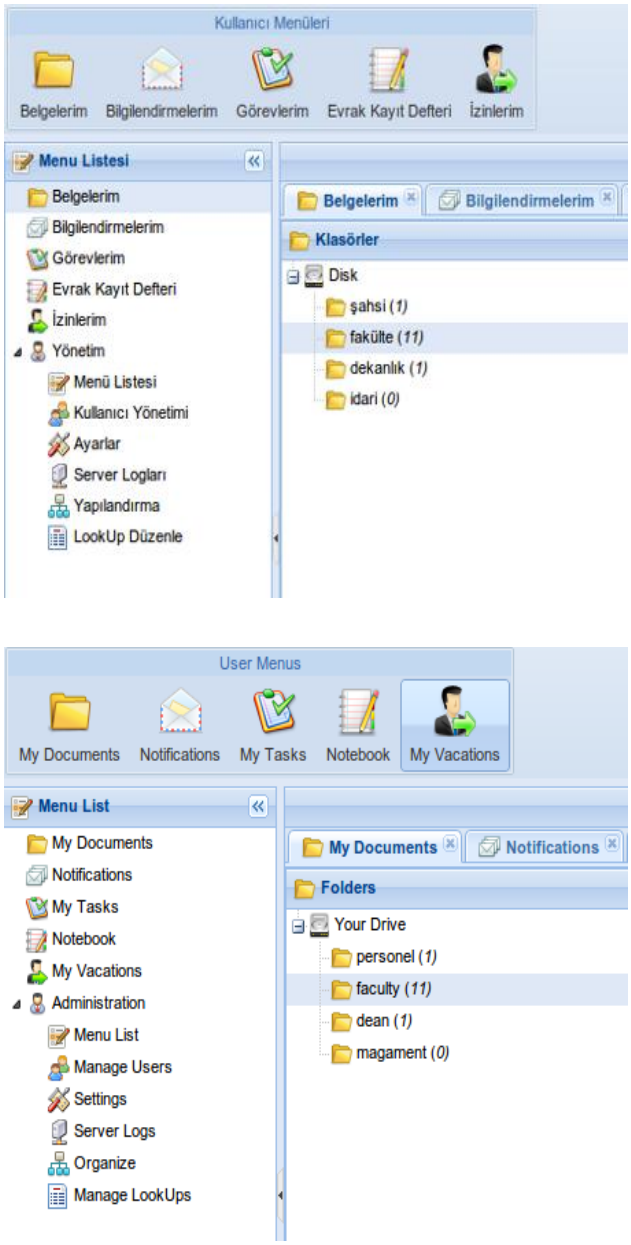

Figure 7. English and Turkish interfaces side-by-side

3.3. Client-side logging

Another new feature for client-side development is client-side logging. An open-source library named log4js-ext which is an extension for Ext JS 4 provides client-side logging with details like priority, time, category and it also has a built-in log viewer window to show stored logs with all details as it is shown in Figure 8.

2 http://gzip.org
3 http://tools.ietf.org/html/rfc1952
4 http://ietf.org

Main features of this library can be listed as:

- It allows logging with categories including various log priorities like info, warning, debug, error.
- Offers Nested Diagnostic Context (NDC) to distinguish interleaved log output from different sources
- Offers nicely highlighted log viewer GUI (Graphical User Interface) to search logs through their content, priority level and category.
- Library offers to log nested complex objects; not just simple messages
- Provides advanced and customizable log formats
- Uses very similar syntax to highly popular logging libraries

4. Conclusion

JavaScript is a powerful scripting language directly interpreted by web browsers. There are some deficiencies about its architecture that must be overcome in order to offer advanced approaches for both software development and performance.

Yet JavaScript doesn't have a built-in object-oriented programming support, an imitation of it can be built for taking advantages of object-oriented concepts. In this study, a JavaScript library that encapsulates these deficiencies is tested to reveal its effects.Ext JS 4 is a powerful and extensible JavaScript framework that supports both MVC and OOP approaches. Every object of Ext JS is derived from Ext.Object and it supports inheritance, encapsulation with demand on resource loading *(which is also known as dynamic loading)*. This is done by a class named Ext.Loader which basically defines paths for packages as it is discussed before. This approach improves application startup performance by 525% as it is shown in Table 1. With source compression and dynamic source loading features, web pages are loaded almost 100% faster than before because of reduced source size as it is shown in Table 2. Whenever a class needs another, this requirement is defined by a 'requires' relation which guaranties required sources to be loaded before launching the class. 'mixins' feature can be used to meet multiple inheritance deficiency for client-side objects.

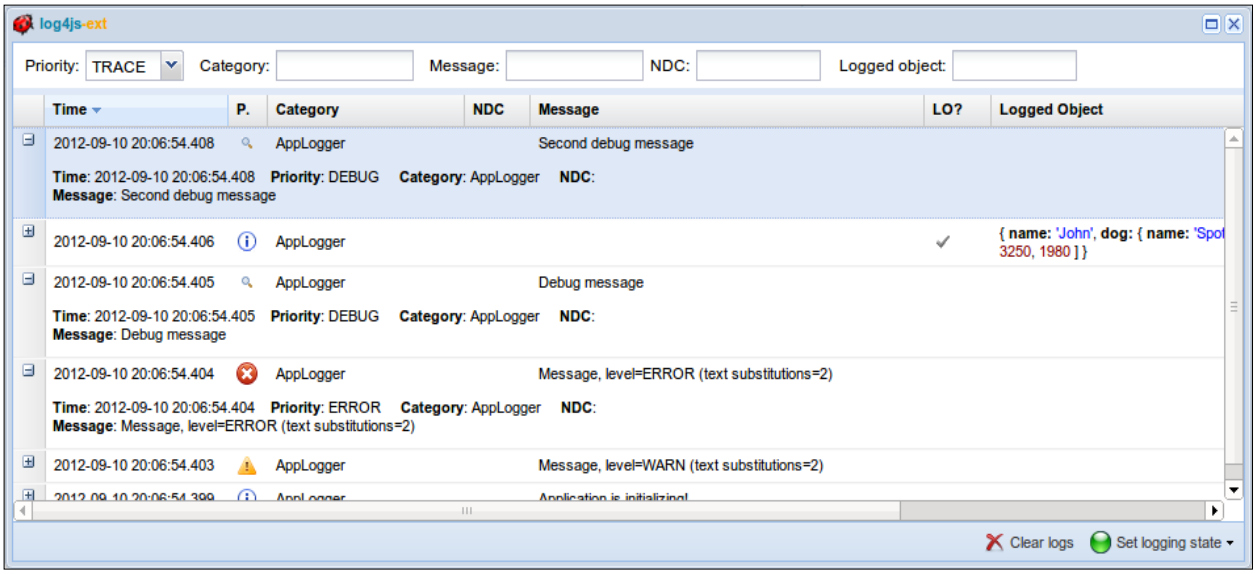

Figure 8. An overview of Log4js log viewer window

References

1. Boerman, R.: Using Model Associations in Sencha Touch 2 and Ext JS 4, http://appointsolutions.com/2012/07/using-model-associations-in-sencha-touch-2-and-ext-js-4/, July 2012.
2. Crockford, D., JavaScript: The Good Parts, s. 2, O'Reilly Media/Yahoo Press, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2008.
3. Flatt, M., Krishnamurthi, S., Felleisen, M., A programmer's reduction semantics for classes and mixins, In: Alves-Foss, J. (ed.) Formal syntax and semantics of Java, s. 13, Springer, 1999.
4. Garcia, J., Andresen, J.K., Grisogono, G., ExtJS in Action, 2nd ed., s. 19, Manning Publication Co., 180 Broad St. Suite 1323 Stamford, CT 06901, USA, 2013.
5. Groner, L., Ext JS 4 First Look, s. 272, Packt Publishing, Livery Place 35 Livery Street Birmingham B3 2PB, UK, 2012.
6. Groner, L.: ExtJS 4 MVC Architecture Mind Map, http://www.slideshare.net/loianeg/extjs-4-mvc-architecture-mind-map-13669488, July 2012.
7. Hasan, S.S., Isaac, R.K., An integrated approach of MAS-CommonKADS, Model–View–Controller and web application optimization strategies for web-based expert system development, Expert Syst. Appl., 38, 417–428, 2011.
8. MacCaw, A., JavaScript Web Applications, s.3, O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2011.
9. Neumann, G., Zdun, U., Strembeck, M., Object-based and class-based composition of transitive mixins, Inf. Softw. Technol., 49, 871–891, 2007.

10. Spencer, E.: Countdown to Ext JS 4: Dynamic Loading and New Class System, http://www.sencha.com/blog/countdown-to-ext-js-4-dynamic-loading-and-new-class-system, Jan. 2011.
11. Uyun, S., Rifqi, M., Implementation Of Model View Controller (MVC) Architecture On Building Web-based Information System, Islam Zeitschrift Für Geschichte Und Kultur Des Islamischen Orients, 47–50, 2010.
12. Varma, V., Software Architecture: A Case Based Approach, s. 94, Pearson Education India, New Delhi, India, 2009.
13. Zakas, N.C., Professional JavaScript for Web Developers, s. 202, Wrox, Indianapolis, Indiana, 2012.