КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ИНФОРМАТИКА

Журнал в журнале. Выпуск №1
Материалы VII Международной научно-технической конференции
«Конструкторско-технологическая информатика», Москва, 2018
Редакционная коллегия: Шептунов С.А., Капитанов А.В., Карлова Т.В., Запольская А.Н.

УДК 004.415.2

РАЗРАБОТКА МЕТОДА РЕПЛИКАЦИИ ДАННЫХ МЕЖДУ КЛИЕНТАМИ В МНОГОПОЛЬЗОВАТЕЛЬСКИХ ВЕБ-ПРИЛОЖЕНИЯХ РЕАЛЬНОГО ВРЕМЕНИ, ПОСТРОЕННЫХ НА БАЗЕ ПРОТОКОЛА WEBSOCKET

Синицин И.В.[1], Леонов Е.А.[1], Аверченков А.В.[1], Шептунов С.А.[2]
[1]Брянский государственный технический университет, г. Брянск, РФ
[2]Институт конструкторско-технологической информатики Российской академии наук, Москва, РФ
E-mail: johnleonov@gmail.com

В статье обоснована актуальность применения протокола WebSocket при построении веб-приложений реального времени. Сделан краткий обзор современных подходов к разработке веб-приложений реального времени. Предложены принципы построения веб-приложений с использованием связующего программного обеспечения обеспечивающего актуализацию данных на всех подключенных клиентах непосредственно в момент изменения данных. Разработана математическая модель процесса согласования данных между клиентами в связующем программном обеспечении. На базе разработанной модели предложен вариант архитектуры, в основу которого лег шаблон Model-View-ViewModel. В предложенной архитектуре приложения в качестве View выступает представление данных на клиенте, ModelView представлено на уровне связующего программного обеспечения и обеспечивает согласование данных между клиентами, а Model является представлением хранимых данных и взаимодействует с используемой системой управления базами данных.

Ключевые слова: *WebSocket, веб-приложения, приложения реального времени, репликация данных, MVVM, связующее программное обеспечение, распределенные информационные системы*

Введение

На ранних этапах создания Internet перед разработчиками протокола HTTP не стояла задача сделать его интерактивным, так как протокол был изначально разработан именно для передачи HTML и незначительного количества сопутствующих внедрений [11-12]. В связи с этим определилась стабильная архитектура построения веб-приложений, основанная на использования в основном только методов GET и POST.

За последнее десятилетие с бурным развитием Internet многие пришли к пониманию необходимости разработки и внедрения концепции интернет-приложений реального времени. Были предприняты попытки построения концепций таких приложений на базе уже имеющегося HTTP протокола. Они получили общее название «Comet» – модель построения веб-приложений, которая позволяет веб-серверу отправлять данные клиенту без дополнительных запросов. Данный подход, в основном, базируется на long polling механизме запросов – таких HTTP-запросов, ответ на которые дается не сразу, а только по некоторому событию.

На практике long polling – это запросы для предотвращения проблем с соединением, например, из-за прокси-серверов, часто ограничивают по времени. Одним из примеров использования данного подхода в построении веб-приложения является использование Bayeux-протокола, позволяющего обмениваться асинхронными сообщениями по протоколу HTTP. Использование данного протокола, например, в SaleForce Streaming API [1], позволяет оформить подписку на конкретные события, а также отменить ее, если необходимо. Кроме уже упомянутой проблемы с ограничением long polling-запросов по времени, что кроме этого ведет к увеличению трафика (в случае активного использования весьма значительного), данные подходы не позволяют реализовать полнодуплексное соединение, так как действия, происходящие между запросами, могут быть «потеряны» для пользователя.

Для устранения недостатков данных подходов был предложен протокол WebSocket – асинхронный протокол поверх TCP-соединения, специально предназначенный для передачи сообщений в режиме реального времени. Данный

протокол быстро получил популярность, приобрел множество реализаций, предоставляющих удобное API для взаимодействия серверной и клиентской стороны. Одной из наиболее популярных реализаций является библиотека Socket. IO [2], написанная на JavaScript и Node.JS. Также популярность набирает реализация WebSocket для Spring Framework [7-8].

Построение приложения на базе протокола WebSocket связано с определенными трудностями на этапе проектирования его архитектуры [6]. Рассмотрим основные особенности архитектуры приложений, понимание которых необходимо для построения системы реального времени. Приложение можно представить как подчиняющийся бизнес-логике процесс управления базой данных через интерфейс пользователя. Другими словами, можно выделить три основные составляющие приложения: базу данных, бизнес-логику и интерфейс пользователя.

В последние годы активное развитие приобрели технологии NoSQL баз данных [9-10], вызванное, не в последнюю очередь, увеличением количества данных, обрабатываемых в Internet. Несмотря на это, реляционный подход к построению баз данных нисколько не утратил актуальность: простое для пользователя представление информации в виде отношений, развитый математический аппарат их представления, удобство и простота сопровождения делают данный подход идеальным при построении систем с малым и средним объемом обрабатываемых данных.

Интерфейс пользователя можно представить, как совокупность представлений (view) данных из нескольких таблиц (выборки), отображаемых посредством графических элементов. В конкретный момент времени пользователь может работать сразу с несколькими представлениями, которые, в том числе, могут использовать одни и те же данные, относящиеся к одной таблице, но входящие в разные выборки. При построении веб-приложения реального времени это означает, что должна быть возможность «подписаться» на события изменения отображаемых данных так, что при изменении одного отображения, данные которого участвуют в другом, изменения должны отобразить оба. Это приводит к основным концепциям предлагаемого подхода:

– должны фиксироваться изменения конкретных данных;

– изменение данных должно порождать информирующие события;

– пользовательские представления (view) должны иметь возможность подписываться на события, порождаемые изменением данных;

– использование протокола WebSocket при реализации.

Модель представлений данных

Формализуем предложенные концепции относительно реляционных баз данных. Рассмотрим структуру реляционной базы данных с точки зрения теории множеств. Пусть R – реляционная база данных, состоящая из k отношений T_i, $i = 1\ldots k$:

$$R = \{ T_1, T_2, \ldots, T_i, \ldots, T_k \}. \quad (1)$$

В свою очередь каждое отношение T_i представляет собой совокупность атрибутов a_i, $j = 1\ldots LT_i$, где $LT_i = |T_i|$:

$$T_i = \left\{ a_1, a_2, \ldots, a_j, \ldots, a_{|LT_i|} \right\}. \quad (2)$$

Любое отношение T_i может содержать от 0 до n_{T_i} ключей:

– первичных $Pk_i = \{a_1, a_2, \ldots, a_j, \ldots, a_{|LPk|}\}$;

– внешних $Fk_i = \{a_1, a_2, \ldots, a_j, \ldots, a_{|LFk|}\}$,

где $LPk = \left|Pk_j\right|, LFk = \left|Fk_j\right|$.

На каждом отношении T_i можно определить выборку $S_j^{T_i}$ – кортеж над отношением T_i, отобранный по некому предикату C:

$$S_j^{T_i} = \left\{ a_1^{T_i}, a_2^{T_i}, \ldots, a_j^{T_i}, \ldots, a_L^{T_i} \right\}.$$

Внешние ключи задают связь между двумя отношениями, то есть если между отношениями задан внешний ключ, то существует отображение атрибутов из отношения T_l в отношение T_h такое, что

$$fk : Fk_i^{T_l} \to S_j^{T_h} \text{ и } \left|Fk_i^{T_l}\right| = \left|S_j^{T_h}\right|. \quad (3)$$

При этом у пары $Fk_i^{T_l}$, $Sk_j^{T_h}$ должны совпадать типы данных у соответствующих пар атрибутов, а в рамках двух записей должны совпадать значения этих атрибутов.

Множество ключей таблицы может включать в себя уникальные ключи – ключи, однозначно идентифицирующие записи отношения:

$$Uk^{T_k} \in Pk^{T_k} \bigcup Fk^{T_k}. \quad (4)$$

На каждом отношении можно задать конечное количество:

$$S^{T_i} = \left\{ S_1^{T_i}, S_2^{T_i}, \ldots, S_j^{T_i}, \ldots, S_J^{T_i} \right\}. \quad (5)$$

Рассмотрим структуру отображения базы данных у пользователя. Пользователь получает доступ к базе через пользовательский интерфейс – совокупность элементов интерфейса V_i (панели, DataGrid (сетка данных), всплывающих уведомлений и т.д.):

$$UI = \{V_1, V_2, \ldots, V_i, \ldots, V_{LUI}\}, \quad (6)$$

где $LUI = |UI|$. Каждый из элементов интерфейса отображает данные некой совокупности выборок из одного или более отношений:

$$V_i = \{S_1^{T_1}, \ldots, S_{J_1}^{T_1}, S_1^{T_2}, \ldots, S_{J_2}^{T_2}, \ldots, S_{J_k}^{T_k}\} = \{S_j^{T_k} \mid T^k \in R\}. \quad (7)$$

Модель взаимодействия клиента

В упрощенном виде сценарий поведения пользователя можно представить с помощью двух функций – отображения интерфейса пользователя на базу данных

$$f : UI \to R,$$

и обратного к нему отображения:

$$g = f^{-1} : R \to UI.$$

Так как передача данных в реальных системах может выполниться неверно или вообще не выполниться, например, из-за сбоя соединения, процессы обновления пользовательских интерфейсов, описываемые данными отображения, должны выполняться парой. Тогда выполнение пользователем действия с интерфейсом можно записать как с помощью следующей функции:

$$F(f, g, UI, R) = g(f(UI, R), UI). \quad (8)$$

При стандартном подходе к проектированию веб-приложений данные отображения выглядят довольно тривиально: совокупность (объединение) выборок из одного или нескольких отображений $\bigcup_{j,k} S_j^{T_k}$ служит для формирования элементов интерфейса пользователя V_i. Воздействие пользователя на базу данных происходит в обратном направлении – на основании данных из графического интерфейса заполняются соответствующие подмножества атрибутов отношений и происходит их обновление, и, в зависимости от удачного или неудачного изменения данных в базе, интерфейс пользователя обновляется к новым данным или откатывается к предыдущей версии. И, так как в сессии обновления данных участвует лишь один пользователь, изменения базы затрагивает лишь графические элементы V_i.

Модель многопользовательского доступа к данным

Данная система усложняется с введением в нее нескольких пользователей. Пусть задано множество пользователей U вида

$$U = \{u_1, u_2, \ldots, u_i, \ldots, u_n\}. \quad (9)$$

Рассмотрим сначала простой случай, когда все пользователи используют один элемент интерфейса V_j (например, редактируемая сетка). Хотя у каждого i-го пользователя одна и та же форма-представление, объекты выборок могут различаться (так как каждый пользователь может запрашивать свои данные):

$$v_j^{u_i} \in V_j \sim \{s_j^{u_i T_k}\} \in \{S_j^{T_k}\}, \quad (10)$$

где $v_j^{u_i}$ – форма представление V_j для пользователя u_i, $s_j^{u_i, T_k}$ – выборка s_j из отношения T_k для пользователя u_i. Теперь при выполнении отображения f интерфейсы пользователей, участвующих в отображении измененяемых данных, связанные с обновляемыми данными, также должны обновиться. Таким образом, появляется новое отображение, затрагивающее связанные с выборками данные:

$$g' : s_j^{u_i, T_k} \to \bigcup_{l \neq i} s_j^{u_l, T_k}. \quad (11)$$

Тогда функция (8) записывается в виде:

$$F(f, g', \bigcup_i UI^{u_i}, R) = g'(f(UI^{u_l}, R), \bigcup_{i \neq l} UI^{u_j}), \quad (12)$$

где $u_i, u_j, u_l \in U$.

Рассмотрим теперь случай с двумя пользовательскими элементами интерфейса, изменение одного из которых влечет изменение другого: $v_1^{u_i}$ и $v_2^{u_j}$, с выборками, имеющими пересечение, то есть

$$v_1^{u_i} \cap v_2^{u_j} = \bigcup_{k,l} s_l^{T_k} \neq \emptyset, v_1^{u_i} \neq v_2^{u_j}. \quad (13)$$

Нетрудно заметить, что в данном случае, помимо отображения g', задающего преобразование данных из одной выборки, присутствует также отображение, задающее преобразование данных между двумя различными выборками. Пусть при воздействии из элемента интерфейса V_1 на базу R при помощи отображения f затра-

гиваются выборки $\left\{S_j^{T_k,V_1}\right\}$. Непустое подмножество данных выборок используется в форме $V_2 \sim \left\{S_l^{T_k,V_2}\right\}$: $V_1 \bigcap \overline{V_2} = Q = \left\{S_q^{T_k,V_Q}\right\}$. Тогда существуют множества $\overline{V_2} = V_2 \setminus Q, V_1 = V_1 \setminus Q$ такие, что $V_1 \bigcap \overline{V_2} = \varnothing$, $V_2 \bigcap \overline{V_1} = \varnothing$. Таким образом имеем структурное отображение:

$$vs: V_1 \to V_2 \sim \overline{V_1} \bigcap Q \to \overline{V_2} \bigcap Q. \quad (14)$$

Это значит, что по имеющемуся подмножеству Q отображение vs может восстанавливать $\overline{V_2}$ – разницу, между V_2 и Q. Теперь с помощью vs необходимо построить отображение, восстанавливающее данные пользователя i по подмножеству измененных данных пользователя j. Пусть для i-го пользователя известна подвыборка $q^{u_i} \in Q$, по которой нужно найти недостающее подмножество $\overline{v_2^{u_i}} \in \overline{V_2}$. Без дополнительной информации о $v_2^{u_i}$ существует $\prod_{j=1}^{j<\left|\overline{v_2}\right|} \left|s^{u_i,T_j}\right|$ различных выборок, которых можно восстановить по известному q^{u_i}, где $\left|s^{u_i,T_j}\right|$ – количество кортежей в выборке S^{T_j} для пользователя u_i.

В случае, если известны уникальные ключи uk^{u_i,T_j}, однозначно идентифицирующие кортежи s^{u_i,T_j}, количество вариантов восстанавливаемых $v_2^{u_i}$ сокращается до одного. Таким образом, по имеющимся q^{u_i} и $uk_2^{u_i}$ можно сделать выборку из базы, восстанавливающую $v_2^{u_j}$, которая задается отображением вида

$$g'': v_1^{u_i} \to v_2^{u_j} \sim \{q^{u_j}, uk_2^{u_j}, R\} \to v_2^{u_j}. \quad (15)$$

Общий случай выглядит следующим образом:

$$G: v_j^{u_i} \to \bigcup_{k,l} v_l^{u_k} \sim \bigcup_{k,l} \left\{q_l^{u_k}, uk_l^{u_i}, R\right\}. \quad (16)$$

Функция (2) преобразуется к виду:

$$F\left(f, G, \bigcup_i UI^{u_i}, R\right) = G\left(f\left(UI^{u_l}, R\right), \bigcup_i UI^{u_j}\right), \quad (17)$$

где $u_i, u_j, u_l \in U$.

Для завершения данной модели нужно ввести несколько вспомогательных преобразований. Элемент интерфейса V_i задается совокупностью атрибутов отношений, присутствующих в выборках $\bigcup_{j,k} S_j^{T_k} \sim \bigcup_{j,k} a_{j,k}^{T_i}$. Для отображения данный элемента в конкретные атрибуты отношений базы данных необходимо ввести отображение из V_i в подвыборку $S_j^{T_k}$ каждого отношения, участвующего в представлении:

$$fv: V_i \to S^{T_k}, k = 1 \ldots \left|V_i\right|. \quad (18)$$

Обратное преобразование выполняется не полностью на V_i, а на подмножество элементов $V_i^{S_{T_k}}$, учувствовавших в преобразовании fv:

$$fv^{-1} = gv: S^{T_k} \to V_i^{S_{T_k}}, k = 1 \ldots \left|V_i\right|. \quad (19)$$

Архитектура приложения

Рассмотрим пример построения архитектуры веб-приложения на базе рассмотренного подхода. На рисунке 1 представлена схема архитектуры веб-приложения, построенного на базе шаблона MVVM (Model-View-ViewModel) [3]. Пусть пользователь пытается изменить базу данных, взаимодействуя с представлением (view). При этом происходит его «подписка» на события изменения отображаемых у него данных. Его действия запускают функцию $f(UI, R)$, передавая команды манипулирования данными в слой бизнес-логики приложения.

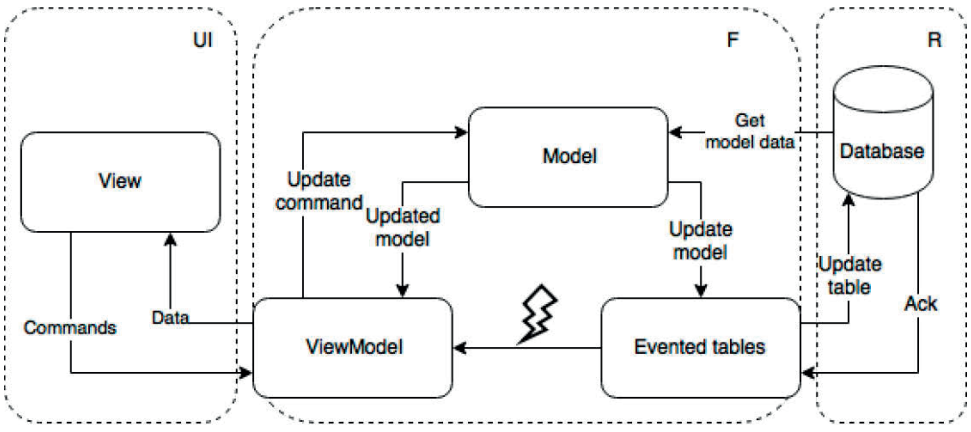

Рисунок 1. Схема архитектуры веб-приложения реального времени:
UI –интерфейс пользователя; R – база данных; F – приложение, обеспечивающее бизнес-логику

Далее *ViewModel* разделяет присланные пользователем данные на подвыборки, заполняет ими соответствующие модели (*model*) и через специальный программный слой (*evented tables*) происходит их обновление в базе данных (преобразование *fv*). На этом функция *f* завершает работу, возвращая текущее состояние базы данных.

Если функция *f* успешно завершила работу, слой *evented tables* «поджигает» события об изменении таблиц, передавая в качестве параметров измененные данные. *ViewModel* реагирует на данные события, восстанавливая модели, которые в текущий момент используют пользователи (преобразование fv^{-1}). Это возможно благодаря уникальным ключам моделей, задействованных пользователями. Так как протокол WebSocket асинхронный, приложение отсылает пользователям затронутые изменениями данные и завершается преобразование G.

Заключение

Хотя предлагаемый подход не лишен недостатков, например, таких, как сложность реализации слоя бизнес-логики или проблематичность интеграции данного решения в уже работающие проекты (в основном из-за необходимости отслеживания изменений моделей), он может хорошо зарекомендовать себя в случаях, где критически важно иметь полное представление о событиях в динамической системе, не налагая дополнительных условий на соединение [4-5].

Литература

1. Sales Force Streaming API. Developer guide. // URL: https://resources.docs.salesforce.com/ sfdc/ pdf/api_streaming.pdf. (д.о. 11.10.2018).
2. Rai R. Socket.IO real-time web application development. Packt Publishing Ltd., 2013, pp. 109-120.
3. Фримен Э. Паттерны проектирования. СПб.: Питер, 2011. – 656 с.
4. Pratihast A.K., DeVries B., Avitabile V. e.a. Design and Implementation of an Interactive Web-Based Near Real-Time Forest Monitoring System // URL: http://dx.doi.org/10.1371 /journal.pone.0150935. (д.о. 11.10.2018).
5. Pimentel V., Bradford N.G. Communicating and Displaying Real-Time Data with Web-Socket // IEEE Internet Computing 16 (2012). – P. 45-53. doi: 10.1109/MIC.2012.64.
6. Ma K., Sun R. Introducing WebSocket-Based Real-Time Monitoring System for Remote Intelligent Buildings // Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China // URL: https:// www.researchgate.net/ publication/271029311_ Introducing_ Web-Socket-Based_Real-Time_ Monitoring_ Sys-tem_for_Remote_Intelligent_ Buildings. (д.о. 11.10.2018).
7. Johnson R., Hoeller J., Donald K. Spring Framework Reference Documentation. WebSocket Support // URL: https://docs.spring. io /spring/docs/5.0.0.BUILD-SNAPSHOT/spring -framework-reference/html/websocket.html. (д.о. 11.10.2018).
8. Kane C., Carroll A., Brener A. Spring in Ac-tion, 4th Edition: Covers Spring 4 // Manning Publications Co. 20 Baldwin Road Shelter Island, NY 1996. – P. 485-510.
9. Nayak A., Poriya A., Poojary D. Type of NOSQL Databases and its Comparison with Relational Databases // International Journal of Applied Information Systems (IJAIS) // Foundation of Computer Science FCS, New York, USA Vol.5 – No.4, March 2013. – P. 16-19.
10. Sharma S. An Extended Classification and Comparison of NoSQL Big Data Models. IJBDI 2 (2015). – P. 201-221.
11. Berners-Lee T., Fielding R., Frystyk H. Hypertext Transfer Protocol – HTTP/1.0. // URL: https:// tools.ietf.org/html/rfc (д.о. 11.10.2018).
12. Krishnamurthy B., Mogul J.C., Kristol D.M. Key differences between HTTP/1.0 and HTTP/1.1. // WWW ‘99 Proceedings of the eighth international conference on World Wide Web. Elsevier North-Holland, Inc. New York, NY, 1999. – P. 1737-175.

Получено 01.10.2018

Леонов Евгений Алексеевич, к.т.н., доцент Кафедры компьютерных систем и технологий (КСТ) Брянского государственного технического университета (БрГТУ). Тел. (8-483) 256-49-90. E-mail: johnelonov@gmail.com

Синицин Иван Владимирович, аспирант Кафедры КТС БрГТУ. Тел. (8-483) 256-49-90. E-mail: joker.n7@gmail.com

Аверченков Андрей Владимирович, д.т.н., профессор Кафедры КТС БрГТУ. Тел. (8-483) 256-49-90. E-mail: mahar@mail.ru

Шептунов Сергей Александрович, д.т.н., директор ФГАУН Институт конструкторско-технологической информатики Российской академии наук, 127055, Москва, Вадковский переулок, 18, стр. 1А. Тел. (8-499) 978-57-15. E-mail: ship@ikti.org.ru

DESIGNING OF DATA REPLICATION METHOD IN MULTIUSER REAL-TIME WEB APPLICATION BASED ON WEBSOCKET PROTOCOL

Sinitsin I.V.[1], Leonov E.A.[1], Averchenkov A.V.[1], Sheptunov S.A.[2]
[1]Bryansk State Technical University, Bryansk, Russian Federation
[2]Institute for Design-Technological Informatics RAS, Moscow, Russian Federation
E-mail: johnleonov@gmail.com

Over the past decade with rapid growing of the Internet technologies, developers have come to understanding of need to develop and implement real-time web applications. The ways based on HTTP-requests are no longer responses to modern criteria of full-duplex connection with asynchronous events. This paper reports WebSocket protocol based approach for designing or real-time web applications. One of the most difficult parts of the task is methods of data replication, which take place in multiuser application with real-time listening event of data changes. The paper describes this problem through the mathematical model of the ways of real-time communications between web-client users with shared data over the relational database. In additional, there is a sample of real-time web application architecture used suggested approach and Model-View-ViewModel pattern.

Keywords: *WebSocket, web application, real-time applications, data replication, MVVM, middleware applications*

DOI: 10.18469/ikt.2018.16.4.09

Leonov Evgeny Alekseyevich, Bryansk State Technical University, Blvd. 50 Let Oktyabrya, 7b Bryansk, 241035, Russian Federation; Associate Professor of the Department of Computer Technologies and Systems, PhD in Technical Sciences. Tel. +74832564990. E-mail: johnleonov@gmail.com

Sinitsin Ivan Vladimirovich, Bryansk State Technical University, Blvd. 50 Let Oktyabrya, 7b Bryansk, 241035, Russian Federation; PhD student of the Department of Computer Technologies and Systems. Tel. +74832564990. E-mail: joker.n7@gmail.com

Averchenkov Andrey Vladimirovich, Bryansk State Technical University, Blvd. 50 Let Oktyabrya, 7b Bryansk, 241035, Russian Federation; Professor of the Department of Computer Technologies and Systems, PhD in Technical Sciences. Tel. +74832564990. E-mail: mahar@mail.ru

Sheptunov Sergey Aleksandrovich, Institute for Design and Technological Informatics of RAS; Vadkovskiy pereulok, 18, str. 1A, Moscow, 127055, Russian Federation; Director, PhD in Technical Sciences. Tel. +74999785715. E-mail: ship@ikti.org.ru

References

1. SalesForce Streaming API. Developer guide. Available at: https://resources.docs.salesforce.com/sfdc/pdf/api_streaming.pdf. (accessed 11.10.2018).
2. Rai R. Socket.IO real-time web application development. Packt Publishing Ltd., 2013, pp. 109-120.
3. Freeman E. *Patterny proektirovaniya* [Design Patterns]. St. Petersburg, Piter Publ., 2011. 656 p.
4. Pratihast A.K., DeVries B., Avitabile V. Design and Implementation of an Interactive Web-Based Near Real-Time Forest Monitoring System. *PLoS One*, 2016. Available at: http://dx.doi.org/10.1371/journal.pone.0150935. (accessed 11.10.2018).
5. Pimentel V., Bradford N.G. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing*, 2012, vol. 16, no. 4, pp. 45-53. doi: 10.1109/MIC.2012.64.
6. Ma K., Sun R. *Introducing WebSocket-Based Real-Time Monitoring System for Remote Intelligent Buildings. Shandong Provincial Key Laboratory of Network Based Intelligent Computing.* Jinan, University of Jinan. Available at: https://www.researchgate.net/publication/271029311_Introducing_WebSocket-Based_Real-Time_Monitoring_System_for_Remote_Intelligent_Buildings. (accessed 11.10.2018).

7. Johnson R., Hoeller J., Donald K. *Spring Framework Reference Documentation. WebSocket Support.* Available at: https://docs.spring.io/spring/docs/5.0.0.BUILD-SNAPSHOT/spring-framework-reference/html/websocket.html. (accessed 11.10.2018).
8. Kane C., Carroll A., Brener A. *Spring in Action, 4th Edition. Covers Spring 4.* NY, Manning Publications Co., pp. 485-510.
9. DB-Engines Ranking. Available at: https://db-engines.com/en/ranking (accessed 11.10.2018).
10. Sharma S. An Extended Classification and Comparison of NoSQL Big Data Models. *IJBDI*, 2015, no. 2, pp. 201-221.
11. Berners-Lee T., Fielding R., Frystyk H. *Hypertext Transfer Protocol -- HTTP/1.0*. Available at: https://tools.ietf.org/html/rfc1945 (accessed 11.10.2018).
12. Krishnamurthy B., Mogul J.C., Kristol D.M. Key differences between HTTP/1.0 and HTTP/1.1. *WWW '99 Proceedings of the eighth international conference on World Wide Web*. Elsevier North-Holland, Inc. New York, NY, USA 1999, pp. 1737-1751

Received 01.10.2018

УДК 005.62 + 004.05

ОСНОВЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ ПРИ РАЗРАБОТКЕ САЙТОВ

Юлдашев Т.З., Шептунов С.А.
Институт конструкторско-технологической информатики Российской академии наук, Москва, РФ
E-mail: info@dlay.ru

В условиях широкого внедрения автоматизированной обработки данных особая важность придается вопросу защиты информации от неправомерного доступа к ней злоумышленников. Оценку защищенности на этапе веб-разработки следует проводить методом анализа угроз по разным критериям (например, программный код, разметка страницы, пароли и т.д.). В статье приведены основные ошибки разработчиков сайтов при создании приложений, проведен анализ методов их устранения и повышения уровня безопасности для пользователя. На основе проведенного анализа сделан вывод о необходимости уделять значительное внимание сетевой безопасности при разработке сайтов и их использовании. Знание основ информационной безопасности пользователями может помочь защитить данные пользователя и само приложение от самых популярных современных угроз.

Ключевые слова: *сетевая безопасность, разработка сайтов, информационная безопасность, создание приложений*

Введение

В современном мире процессы работы с информацией являются одними из наиболее значимых составляющих существования любой системы окружающего мира. Потеря информации, ее искажение, а также невозможность своевременного получения необходимых данных может иногда привести даже к катастрофическим последствиям. В деловой и частной жизни люди достаточно часто получают необходимую им информацию из веб-источников, поэтому при разработке сайтов следует уделять особое внимание информационной безопасности.

К сожалению, современный уровень веб-разработки не всегда находится на достаточном уровне безопасности, так как на практике при разработке проекта сайта на обеспечение его защищенности часто выделяется недостаточное количество финансовых и квалифицированных человеческих ресурсов.

Помимо того, что разработчик должен быть компетентен, ему необходимо соблюдать основные правила обеспечения безопасности при написании кода в разрабатываемом проекте.

Код должен быть понятным и масштабируемым, а также сбалансированным по следующим критериям: достаточно гибким, и, одновременно, стабильным и надежным. Необходимо создавать интуитивно-понятный интерфейс, оптимизировать базу данных и совершенствовать проект, используя новый инструментарий веб-разработки и современное оборудование.

Обеспечение информационной безопасности разрабатываемого проекта сайта для заказчика, с одной стороны, является очень важной составляющей, а с другой стороны, возникают вопросы о том, какой уровень безопасности им необходим. В отличие от производительности веб-сайта невозможно дать количественную оценку «достаточной безопасности» разрабатываемого проекта. Посто-