**МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РЕСПУБЛИКИ МОЛДОВА**  
**МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ**  
**ДЕПАРТАМЕНТ ИНФОРМАТИКИ**

**Артур Добря**

**РАЗРАБОТКА ПРИЛОЖЕНИЙ НА JAVA С ИСПОЛЬЗОВАНИЕМ “SPRING SECURITY” И “VAADIN”**

**Дипломная работа**

Шеф департамента: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(подпись)

Научный руководитель: **\_\_\_\_\_\_\_\_\_\_\_\_\_**

(подпись)

Автор: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**КИШИНЁВ-2018**

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ** 2

**I.**  **ОСОБЕННОСТИ НАПИСАНИЯ GUI** 3

1.1. Проблема проектирования GUI 3

1.2. Методы решения проблем с проектированием GUI 4

1.3. Сравнение технологий разработки GUI 5

1.4 Архитектура vaadin 6

1.5 Клиентская модель разработки 7

**II. SPRING SECURITY**

2.1. Проблема безопасности приложения

2.2. Методы решение проблем с защитой приложения

2.3. Архитектура технологии spring security

#### 2.3.1 Ключевые объекты контекста Spring Security

2.3.2 Аутентификация

**ВЫВОДЫ**

**БИБЛИОГРАФИЯ**

**ПРИЛОЖЕНИЕ**

**I ОСОБЕННОСТИ РАЗРАБОТКИ GUI**

**1.1. Проблема проектирования GUI**

В больших Enterprise приложений задействовано множество программистов разных направлений, кто сугубо пишет логику приложений, кто занимается проектированием графического интерфейса. Обычно GUI пишутся на таких языках как javascript/css/html. . Java не предлагает адекватных средств для создания современного веб-интерфейса ни с точки зрения дизайна, ни с точки зрения реализации клиент-серверного взаимодействия.

Поэтому было бы не плохо если бы Java предлагала свои вариант реализации GUI.

Для такой проблемы были придуманы framework’и Vaadin, GWT, Wicket, Tapestry которые реализует качественный графический интерфейс.

Такие фреймворки позволяют писать хороший GUI даже без знаний языков front-end разработки (таких как angular/js). Таким образом программист освобожден от изучения дополнительных языков и может верстать GUI на удобном ему языке Java.

**1.2. Методы решение проблем разработки графического интерфейса.**

Данные framework’и помогают облегчить программисту написание большого количества кода front-end. Такие framework’и позволяют писать код на java, который впоследствии транслируется в javascript/html/css. Каждая из библиотек (Vaadin, GWT, Wicket, Tapestry) предлагает свою реализацию графического интерфейса, у каждой свой набор готовых модулей. Например, все эти библиотеки реализует парадигму ООП, но только Vaadin и tapestry реализуют шаблон проектирования MVC, без которого не может обойтись сложное ЕЕ приложение. Если посмотреть со стороны конфигурации, то таким фреймворкам, как GWT, Wicket, Tapestry требуется хоть один конфигурационный xml-файл, в то время как Vaadin конфигурируется сугубо при помощи java кода, что упрощает работу с данной библиотекой.

**Google Web Toolkit (GWT)** — предоставляет разработчику специальное Java API и позволяет разрабатывать компонентно-ориентированные графические пользовательские интерфейсы (GUI), избегая работы с JavaScript и абстрагируясь от протокола HTTP и DOM моделей, лежащих в основе браузеров. Все это достигается использованием GWT компилятора, который не генерирует Java байт-код, точнее он генерирует JavaScript! GWT компилятор берет клиентский Java код и генерирует JavaScript. Скомпилированное GWT приложение состоит из фрагментов HTML, XML и JavaScript. Если вы хотите, чтобы ваше веб-приложение соединялось с веб-сервером, GWT и здесь вам поможет. В арсенале GWT есть Remote Procedure Call (RPC) – механизм удаленного вызова процедур, который осуществляет простой обмен Java-объектами между клиентом и сервером в обе стороны.

(http://www.quizful.net/post/gwt-tutorial-introduction)

**Apache Tapestry**— это фреймворк с открытым кодом для создания динамических, сильных, хорошо масштабируемых веб-приложений в Java. Tapestry реализует MVC («Модель-представление-контроллер») модель в Java. Фреймворк построен на стандартной Java API Servlet, таким образом, он работает с любым servlet контейнерном или сервером приложений.

Tapestry следит за изменениями классов Java страниц, классов компонентов, классов реализации услуг, шаблонов HTML и компонентов файлов свойств, и вносит изменения в работающее приложение без необходимости перезагрузки. Это позволяет быстро просмотреть результат изменения кода приложения и повысить эффективность разработки.

(http://ru.bmstu.wiki/Apache\_Tapestry)

**Vaadin** отлично подходит для написания хорошего графического интерфейса, так как в нем присутствует большое количество тем для верстки, много различных элементов управления интерфейсом, простой и понятный редактор, с котором ещё проще создавать страницы. Большое количество документации и постоянной поддержки frameworka’a со стороны разработчиков Vaadin. Также стоит заметить, что он тесно интегрирован со Spring. Проектирование логики графического интерфейса схож с проектированием на Spring. В нем, как и Spring, есть специальные аннотации и готовые классы для реализации паттерна MVC.

В итоге Vaadin получается простым, понятным и, самое главное, быстрым среди всех своих конкурентов. Его логика легко понятна, как более опытным разработчикам так и начинающим, так как методы реализации логики схож с известным всем Spring.

**1.3. Сравнение технологий**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Framework** | **ООП** | **Только java** | **Конфигурация без XML** | **Реализация патерна**  **MVC** |  |
| Vaadin | **+** | **+** | **+** | **+** |  |
| GWT | **+** | **-** | **-** | **-** |  |
| Wicket | **+** | **+** | **-** | **-** |  |
| Tapestry | **+** | **+** | **+** | **+** |  |

**1.4 Описание архитектуры Vaadin**

Vaadin поддерживает все распространенные браузеры как обычных компьютеров, так и мобильных устройств и планшетов. Вся разработка ведется на Java, но Java-код выполняется только на сервере, на стороне клиента же выполняется чистый JavaScript.

Структурно Vaadin состоит из серверного API, клиентского API, набора компонентов пользовательского интерфейса с обеих сторон, механизма тем для оформления интерфейса и модели данных, позволяющей связывать серверные компоненты непосредственно с данными.

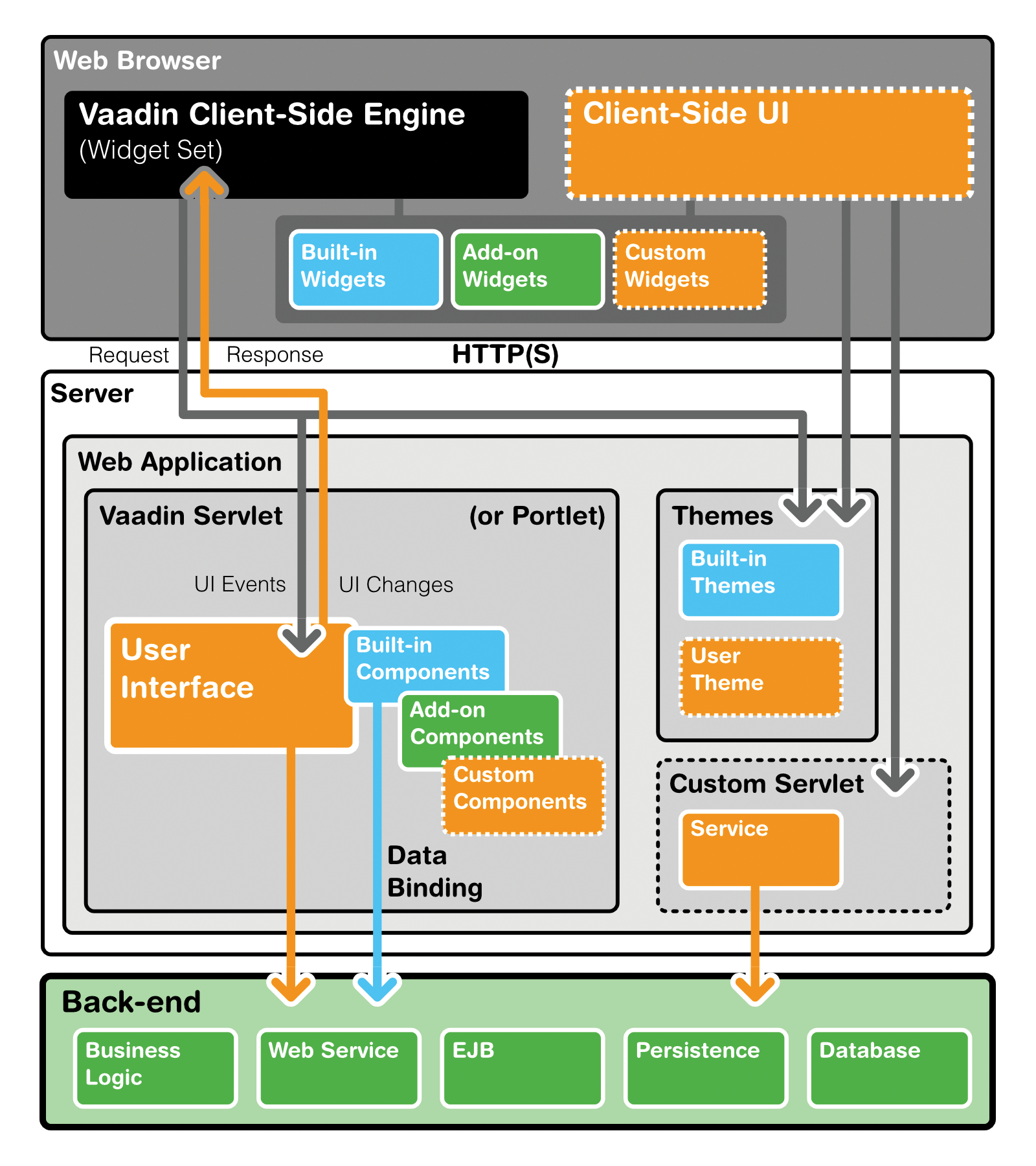


Рис. 2. Архитектура Vaadin

**Client side UI**  - Это интерфейс(сайт) который видит пользователь когда заходит на страницу, ему как пользователю сайта доступны. Так-же ему доступны различные Виджеты которые упрощают работу с интерфейсом. Вся работа Client side ui осуществляется при помощи Vaadin client side ui.

**Vaadin client side engine -** это так называемый контейнер, который и обеспечивает работу фреймворка vaadin. Этот контейнер координирует передачу сервлетов на сервер и получение ответа с сервера. Так-же он контролирует работу всех виджетов и тем.

**User interface -** Это модуль который отвечает за получение данных от клиента и контейнера и направляет данные в **back-end** модуль.

**Back-end -** Это главный модуль в котором и происходит вся логика приложения. Основные главные подмодули приложения это база данных, сервисы(GUI, spring security) Которые обеспечивает работу приложения.

Серверная модель разработки для Vaadin является основной и позволяет создавать законченные приложения без разработки на стороне клиента. При этом используется AJAX-движок Vaadin Client-Side Engine, который формирует пользовательский интерфейс в браузере. Серверный подход позволяет фактически забыть про то, что разработка ведется под веб, и разрабатывать пользовательский интерфейс почти как традиционную Java-программу с непосредственным доступом к данным и сервисам на сервере. При этом серверная часть Vaadin позаботится и о формировании пользовательского интерфейса в браузере, и об AJAX-взаимодействии между браузером и сервером. Движок Vaadin осуществляет обработку пользовательского интерфейса приложения серверной стороны в браузере и реализует все детали обмена информацией между клиентом и сервером.

Серверная часть приложения Vaadin исполняется как обычный сервлет сервера приложений Java. Она представляет собой чистую Java в JAR-файле, который может добавляться к любому стандартному веб-приложению и работает на любом контейнере сервлетов или портлетов от Tomcat до Oracle WebLogic. Сервлет принимает HTTP-запросы от клиента и интерпретирует их как события конкретной пользовательской сессии. События ассоциированы с компонентами пользовательского интерфейса и доставляются к обработчикам (event listeners), определенным в приложении. Если логика пользовательского интерфейса вносит изменения в компоненты пользовательского интерфейса со стороны сервера, сервлет рендерит их для отображения в веб-браузере и формирует ответ. Движок клиентской части, выполняемый в браузере, получает ответ и на его основе производит изменения в загруженной в браузере веб-странице. (https://habrahabr.ru/company/xakep/blog/244477/)

**1.5 Клиентская модель разработки**

Клиентская модель позволяет разрабатывать виджеты и приложения на языке Java, которые затем компилируются в выполняемый в браузере JavaScript с помощью компилятора Vaadin Compiler, основанного на Google Web Toolkit (GWT).

схема Можно использовать и непосредственно JavaScript. Это предоставляет полный доступ к структуре DOM и максимальный контроль над браузером.

(<https://habrahabr.ru/company/xakep/blog/244477/>)

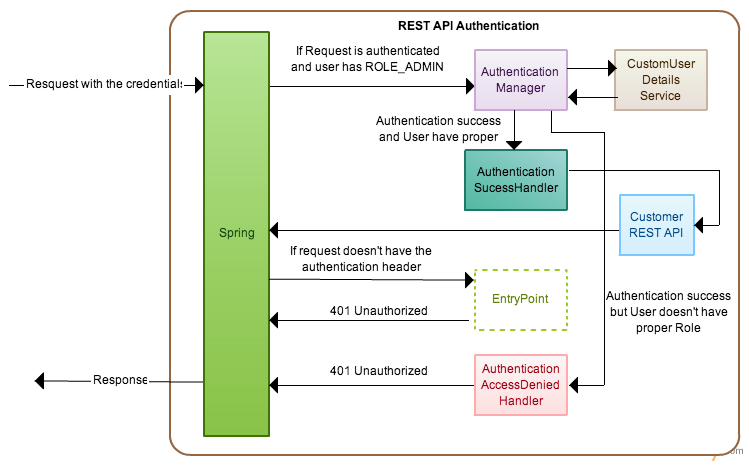
**II Spring Security**

**2.1. Проблема безопасности приложения**

В любом проекте, который хоть как-то связан с веб-разработкой, требуется модуль который будет защищать данные пользователя.

**2.3 Архитектура spring security**

**Spring Security** это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.



+

**2.3.1 Ключевые объекты контекста Spring Security:**

**SecurityContextHolder**, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе работающем в настоящее время с приложением. По умолчанию SecurityContextHolder использует ThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того, чтобы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy).

**SecurityContext**, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

**Authentication** представляет пользователя (Principal) с точки зрения Spring Security.

**GrantedAuthority** отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.

**UserDetails** предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

**UserDetailsService**, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса.Позволяет получить из источника данных объект пользователя и сформировать из него объект UserDetails который будет использоваться контекстом Spring Security.

**2.3.2 Аутентификация**

Пользователю будет предложено войти в систему предоставив имя (логин или email) и пароль. Имя пользователя и пароль объединяются в экземпляр класса UsernamePasswordAuthenticationToken(экземпляр интерфейса Authentication) после чего он передается экземпляру AuthenticationManager для проверки.

- В случае если пароль не соответствует имени пользователя будет выброшено исключение BadCredentialsException с сообщением “Bad Credentials”.

- Если аутентификация прошла успешно возвращает полностью заполненный экземпляр Authentication.

- Для пользователя устанавливается контекст безопасности путем вызова метода SecurityContextHolder.getContext().setAuthentication(…), куда передается объект который вернул AuthenticationManager.

(https://habrahabr.ru/post/203318/)