**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 проектирования GUI 4

1.3. Сравнение технологий разработки GUI 5

1.4 Архитектура vaadin 6

1.5 Клиентская модель разработки GUI 7

**II. SPRING SECURITY**

2.1. Проблема безопасности приложений

2.2. Технологии используемые для защиты приложений

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 свой вариант реализации GUI.

Для решения такой проблемы были разработаны фреймворки Vaadin, GWT, Wicket, Tapestry, которые реализуют 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 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 разработки GUI**

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

**выводы**

**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.

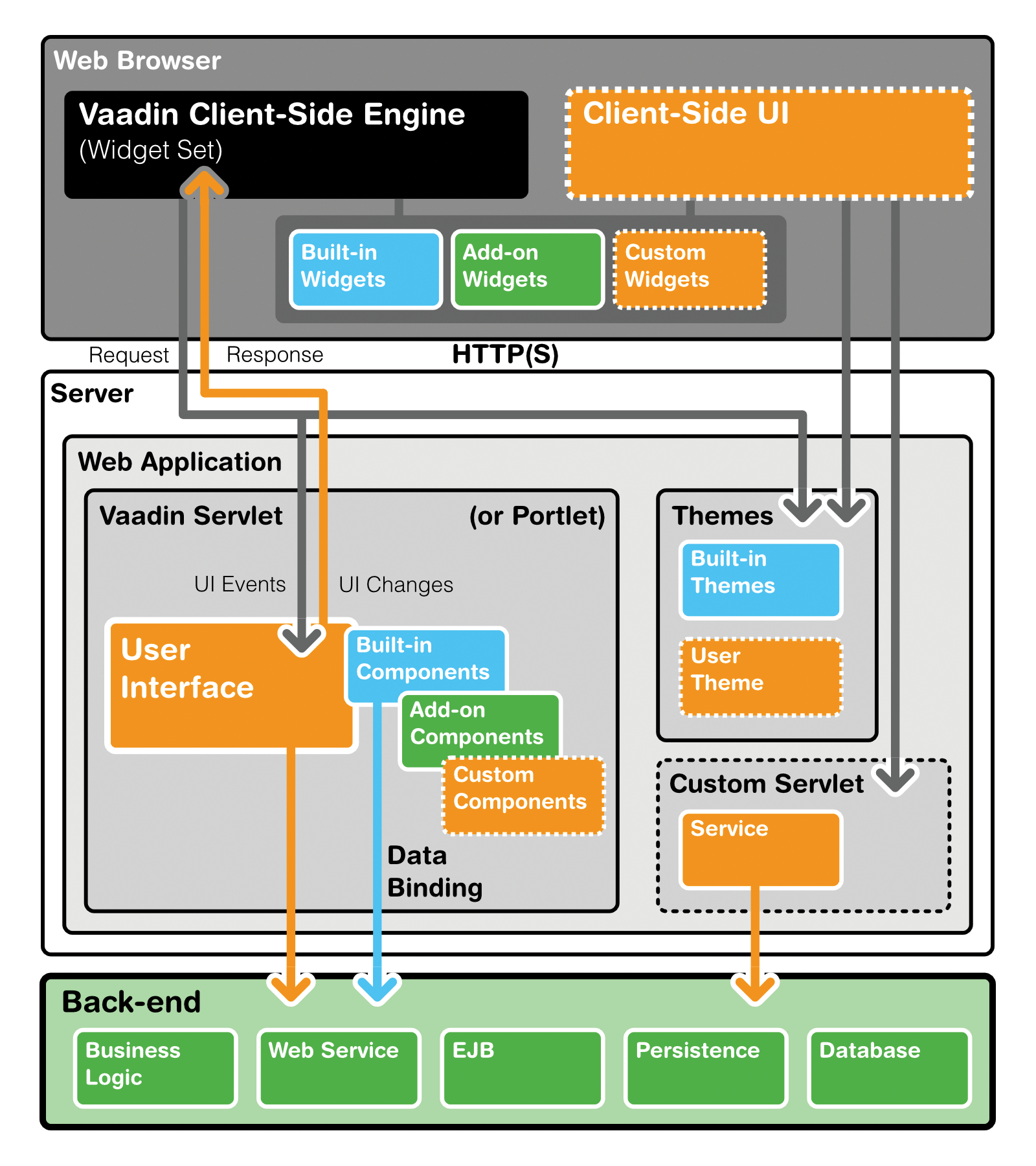


Рис. 2. Архитектура Vaadin

**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.

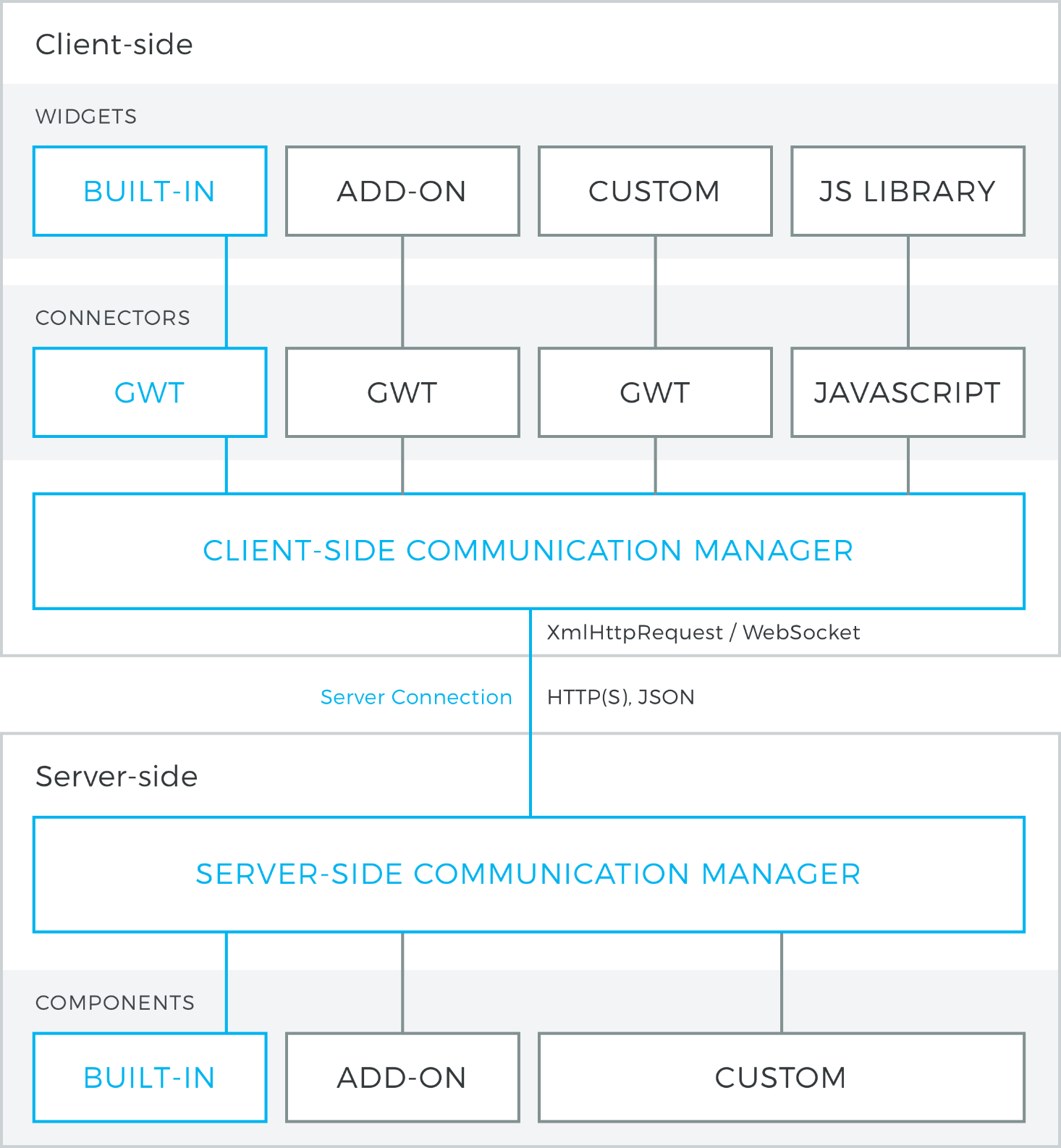
**User interface -** Это модуль который отвечает за получение данных от клиента и контейнера и направляет данные в **back-end** модуль.

**Back-end -** Это главный модуль, в котором и описывается вся логика приложения. Основные главные подмодули приложения - это база данных, сервисы(GUI, spring security), которые обеспечивают 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 на загруженной в браузере веб-странице. (https://habrahabr.ru/company/xakep/blog/244477/)

**1.5 Клиентская модель разработки**

****

В момент, когда пользователя заходит на сайт происходит подгрузка интерфейса с сервера.

Весь интерфейс отображается при помощи виджетов, которые создают UI компоненты.

Клиентская часть включает в себя 2 типа виджетов. GWT виджеты и Vaadin-specific widgets.

Vaadin-specific widgets это компоненты, из которых состоит весь интерфейс. GWT виджеты в свою очередь являются "мостами", через которые получают данные об определенном виджете, который должен быть отображён у пользователя на экране, и формируют уже конечный элемент интерфейса.

У библиотеке Vaadin есть большое количество встроенных компонентов, которые можно использовать для создания своего интерфейса, а также при необходимости можно создавать свои компоненты интерфейса.

**1.6 События и слушатели.**

Vaadin реализует событийно-ориентированное программирование. Каждый раз, когда пользователь совершает действие, например нажимает кнопку или выбирает элемент из списка, приложение должно быть готово обработать запрос пользователя.

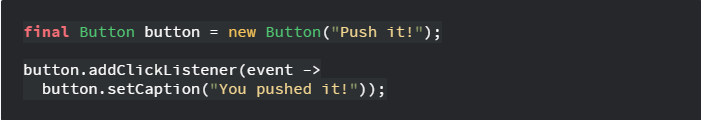
Для обеспечения такого рода связи Vaadin был построен на основе Event-listener pattern (или более извостного Observer pattern).

В основе этого шаблона стоит свзяь между компонентом, который создает событие и компанентом, который его отлавливает. Обычно для кажлого события есть только один слушатель.

Событие может представлять собой любое действие, которое изменяет состояние объекта. Чаще всего изменение состояния компонента происходит благодоря действиям пользователя. Но так же возможно, что сама система производит события независимо от пользователя. Например достижение определленой даты календаря.

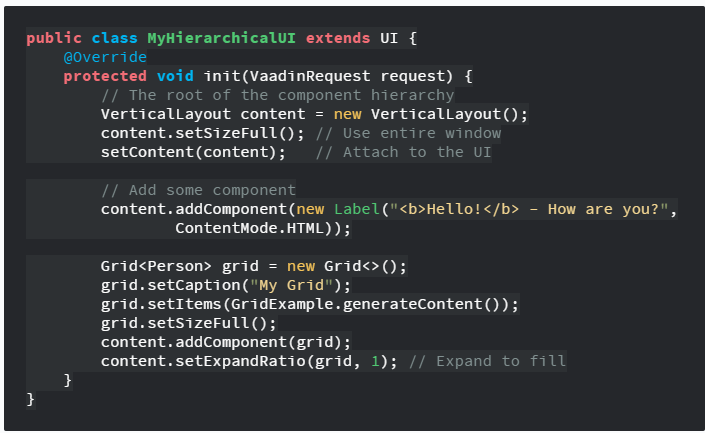
Чтобы получить события определенного типа, в приложении должна быть функция(метод) вида add\*listener().

Например оброботка события кнопки будет выглядеть так:



На стороне клиента будет создана кнопка с надписью “Push it!”, а в самом приложении будет создан слушаеть, который сработает при нажатии этой кнопки и выведет “you pushed it!”

**1.7 Создание графического интерфейса**



В этом классе объявленны основные компоненты страницы

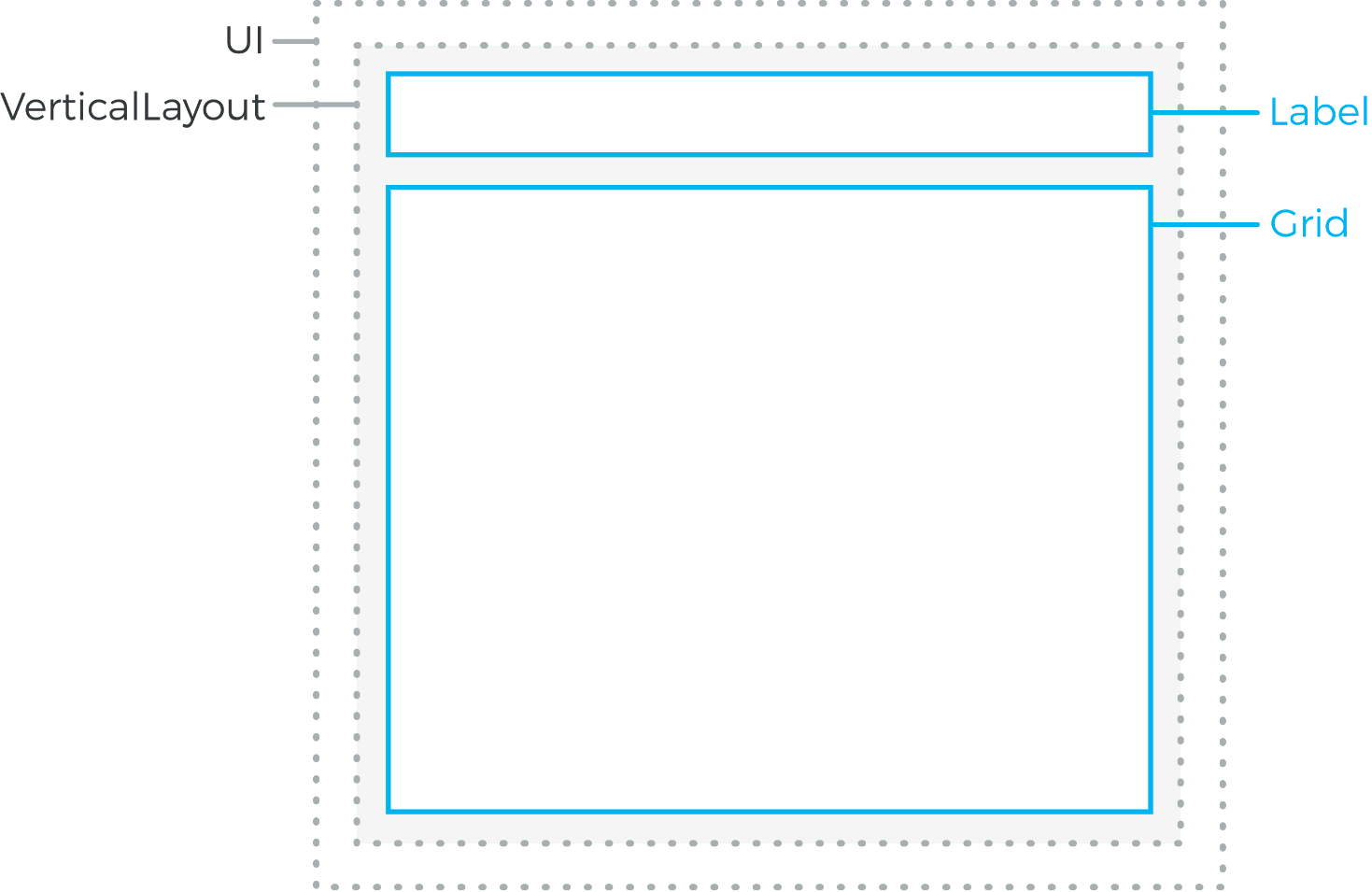
VerticalLayout content = new VerticalLayout();

content.setSizeFull();

setContent(content);

В первую очередь следует объявить, каким образом наши эллементы будут отображаться на странице, в данном случае мы выбрали вертикальное расположение элементов.

В итоге наши элементы будут расположены в таком порядке:



Далее создаем объект Grid, в котором расположим имя, фамилию и год рождения человека.

Grid<Person> grid = new Grid<>();

grid.setCaption("My Grid");

grid.setItems(GridExample.generateContent());

grid.setSizeFull();

Добавляем таблицу как компонент системы. Теперь Vaadin сможет транслировать java код в html/css и отобразить у клиента на страничке:

content.addComponent(grid);

content.setExpandRatio(grid, 1);



**II Spring Security**

**2.1. Проблема безопасности приложения**

В любом крупном приложении требуется модуль, который обеспечил бы защиту персональных и корпоративных  данных, а так же обеспечил шифрование паролей пользователей и любых данных, которые не должны попасть к злоумышленнику. Было бы неэффективно, если бы каждая компания или разработчик в отдельности придумывал свой метод защиты данных. Это бы стоило большого количества  времени и больших усилий со стороны разработчиков: разработать грамотный и эффективный метод защиты данных. Чтобы избежать потери времени, были разработаны фреймворки, которые направлены сугубо на защиту и кодировку данных.

Для решения этой проблемы были придуманы такие фреймоврки, как Shiro, Spring security, JAAS. Все они в разной степени решают одну и ту же проблему, они все способны защитить персональные данные пользователя, а так же провести регистрацию новых пользователей и авторизацию в системе. Для решения такой задачи в данной работе был выбран Spring security так как он и по сей день активно поддерживается самими разработчиками, постоянно выходят новые версии, которые способны поддерживать защиту приложения на высоком уровне, в то время как, например, Shiri не поддерживается обновлениями уже 4 года.

С другой стороны есть Jaas, который все ещё поддерживается разработчиками, но главный минус в том, что он сложен в конфигурации. Требуются хороший навык у разработчика, чтобы грамотно спроектировать и эффективно реализовать работу защиты данных. В то время как SPring security очень прост в конфигурации (базируется в основном на аннотациях), что упрощает разработку для неопытных программистов. Так же Spring Security тесно интегрирован с остальными модулями Spring.

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

**2.2.1 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. 

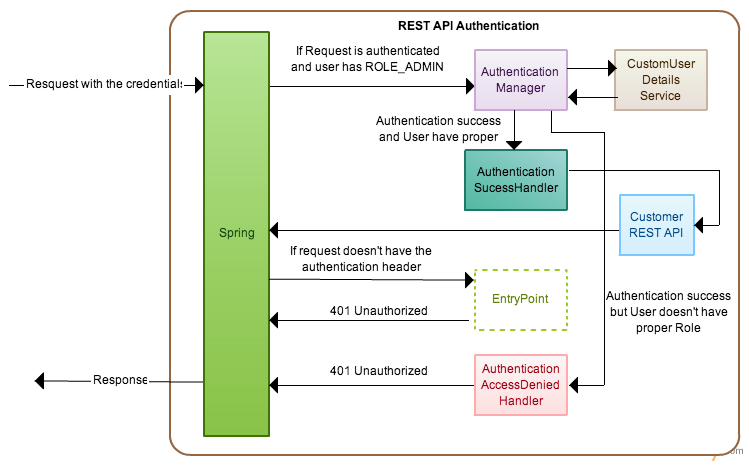
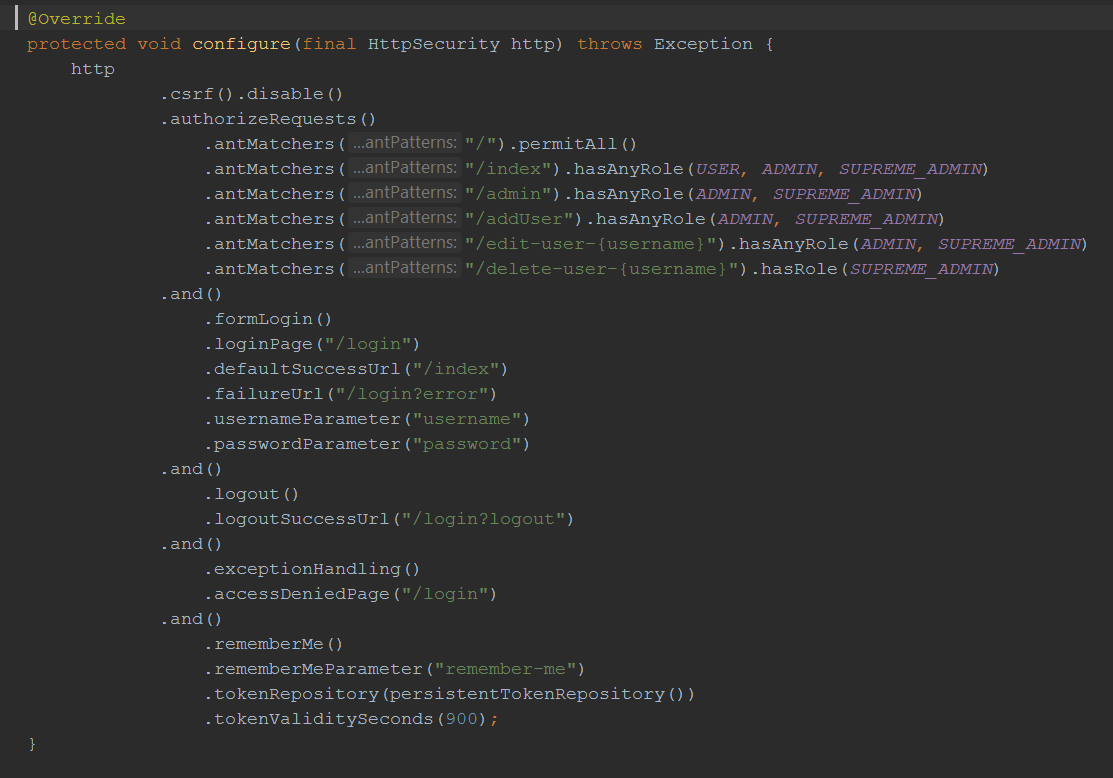
**2.3 Аутентификация**

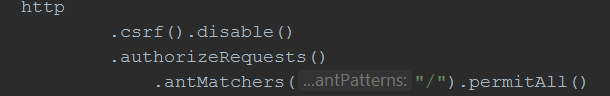
Рис 3 Алгоритм Аутентификации

При попытки неавторизованного пользователя войти на защищенный ресурс, Spring security попросит пользователя пройти регистрацию на сайте, чтобы человек смог получить права доступа, нужные для того, чтобы получить тот ресурс, который он запрашивал. После регистрации идет новая попытка получить доступ к ресурсу. Запрос попадает на сервер, далее его перехватывает модуль **Authentication manager**, который отвечает за защиту данных и предоставления прав доступа пользователям, который, используя сервис **User details  service,** смотрит, какими правами обладает данный пользователь. Если права доступа совпадают с требованием ресурса, то пользователь получает доступ к нему, если нет - то доступ не предоставляется.

Данная модель предоставления прав доступа, реализованна одним классом в Java – WebSecurityConfiguration.

В нем определен главный мето configure, в котором описывается стратегия по которой пользователи могут получить доступ к страницам.

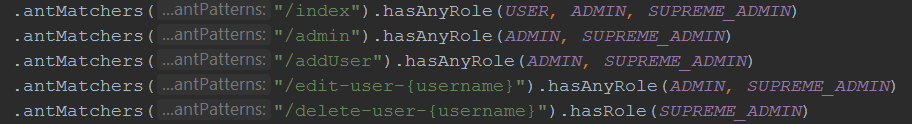


Именно тут происходит весь процесс аутентификации. В этих строчках 

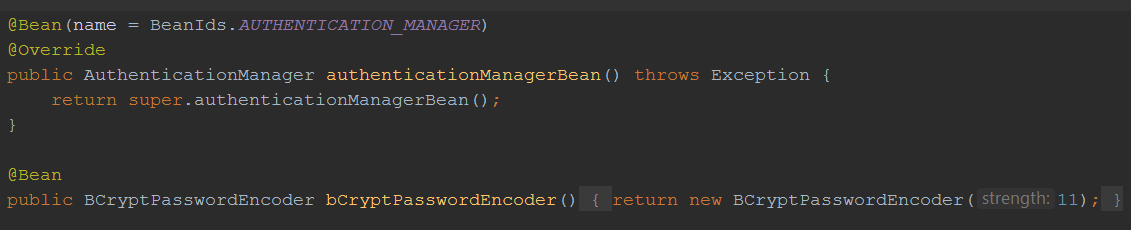
клиенту разрешается свободно получить доступ к главной странице “/” на которой у него будет возможность пройти авторицазию.

Строчка csrf().disable() говорит о том, что защита от csrf(Cross Site Request Forgery) атак отключена на стороне сервера, данная защита осуществляется на стороне клиента специальным Js скриптом.

Далее мы указываем как странницы мы хотим защитить от не авторизированых пользователей



Тут говорится, что на странницу index может перейти пользователь с ролью (user, admin, supreme\_admin) тем самым запрещаю доступ не авторизировануму пользователю. Для работы всей стратегии защиты нужен метод AuthenticationManager – котрый и будет раздавать всем права доступа.

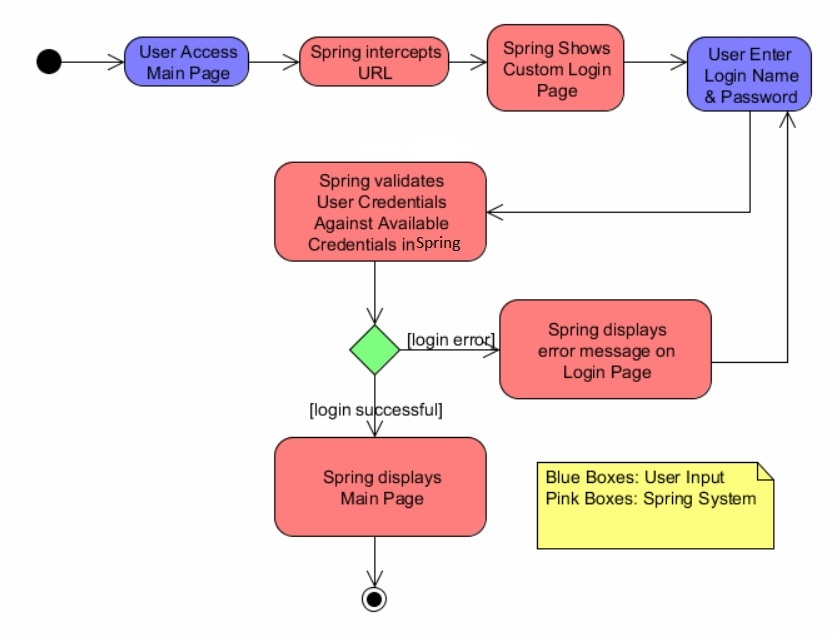


Так же стоит указать метод который будет шифровать наш пороль при регистрации, в данном примере пароль шифруется 11 раз перед тем как отправить на сервер.

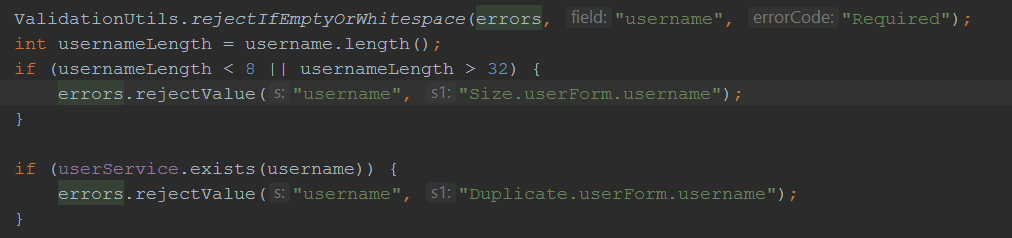
**2.4 Регистрация**

Регистрация – это процесс сохранения нового пользователя в базе данных приложения.

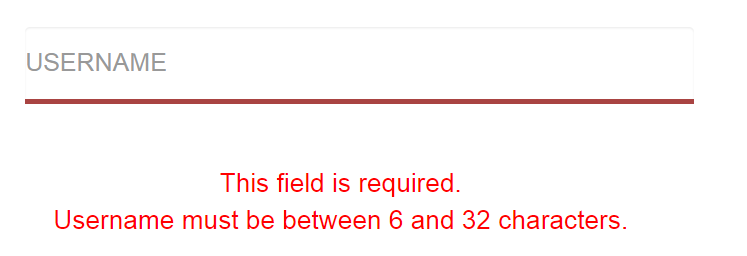
При запросе клиентом страницы с формой регистрацией, сервером будет создана пустая переменная в которой будет проинициирован новый пользователь после введением им данных.



Пользователь заходит на страницу регистрации, вводит свои данные, при отправки данных на сервер данные будут проверены валидатором. Первое что проверяется - это правильность ввода данных. Они проверяются по определенному шаблону.



Данный щаблон проверяет длину username, требования от 8 до 32 символов, если username не удовлетворяет данному шаблону, то клиенту отправляется сообщение об ошибке.



Если данные пользователя проходят проверку, то его данные передаются в контроллер где основываясь на этих данный фармируется новый авторизированый пользователь, который сразу же записывается в сессию сервера. Другая копия данных сохраняются в базу данных, чтобы потом пользователь мог зайти на сайт по этим данным.

Самое главное во всем этом процессе это то, что сессия сервера принимает в себя только пользователей кторые были созданы контекстом Spring security.