RIA Security - Broken By Design

Joonas Lehtinen
IT Mill - CEO

vaadin }>

a system is secure if it is designed to be secure and there are no bugs no system should be designed to be insecure

not all bugs are security holes

not all
security holes
are found and
exploited

security broken by design?

advertises security holes and makes avoiding them harder

1.

RIA
GWT
Vaadin

2.

Security

- Architecture
- Complexity
- Attack surface

3.

Breaking in

- PayMate
- Attacks

vaadin }>

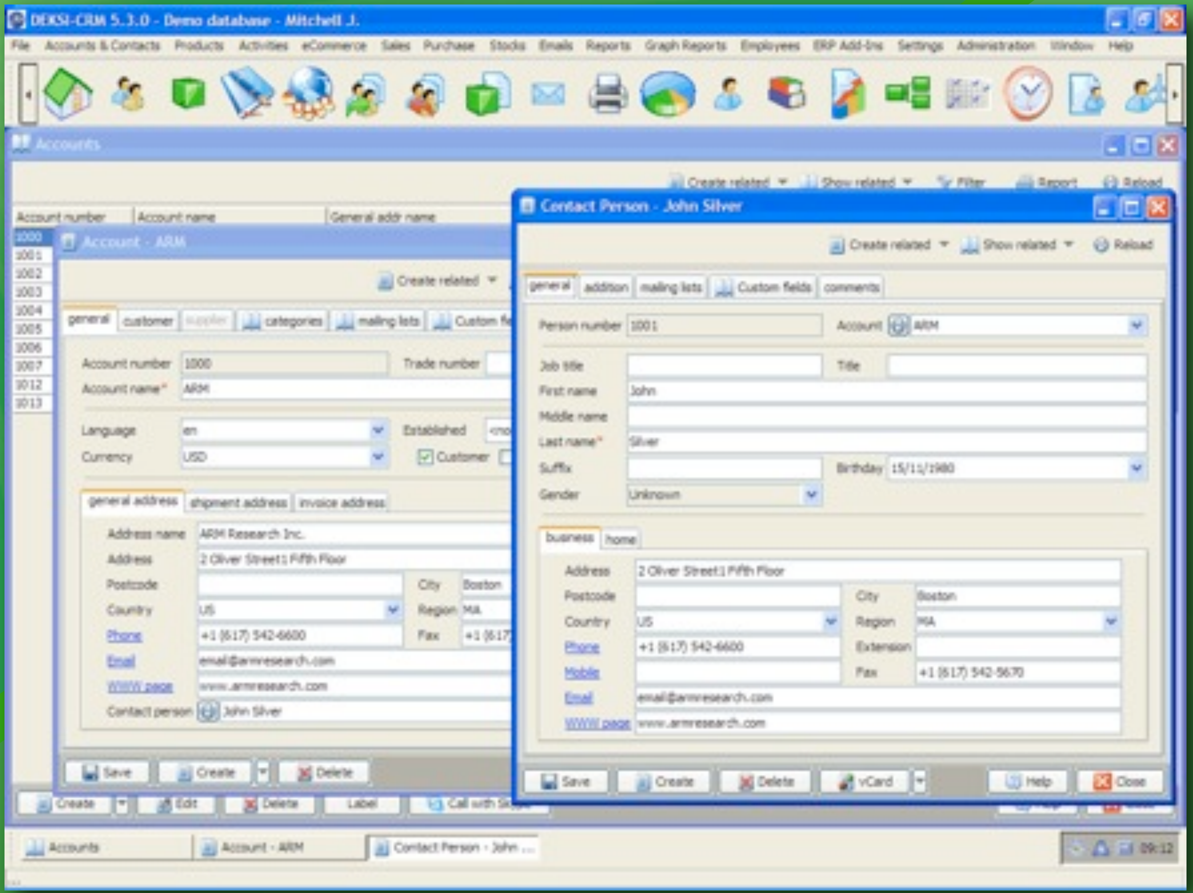
Rich Internet Application

vaadin }>

web platform

3D games

business software



Account number	Account name	General addr name
1000		
1001		
1002		
1003		
1004		
1005		
1006		
1007		
1012		
1013		

Account - ARM

general | customer | supplier | categories | mailing lists | Custom fields

Account number: 1000
Trade number:
Account name*: ARM
Language: en
Established: <no
Currency: USD
Customer ✓

general address | shipment address | invoice address

Address name: ARM Research Inc.
Address: 2 Oliver Street1 Fifth Floor
Postcode:
City: Boston
Country: US
Region: MA
Phone: +1 (617) 542-6600
Fax: +1 (617
Email: email@armresearch.com
WWW page: www.armresearch.com
Contact person: John Silver

Contact Person - John Silver

general | addition | mailing lists | Custom fields | comments

Person number: 1001
Account: ARM
Job title:
Title:
First name: John
Middle name:
Last name*: Silver
Suffix:
Birthday: 15/11/1980
Gender: Unknown

business | home

Address: 2 Oliver Street1 Fifth Floor
Postcode:
City: Boston
Country: US
Region: MA
Phone: +1 (617) 542-6600
Extension:
Mobile:
Fax: +1 (617) 542-5670
Email: email@armresearch.com
WWW page: www.armresearch.com

web pages

English
The Free Encyclopedia
2 904 000+ articles

日本語
フリー百科事典
591 000+ 記事

Deutsch
Die freie Enzyklopädie
914 000+ Artikel

Español
La enciclopedia libre
480 000+ artículos

Français
L'encyclopédie libre
812 000+ articles

Polski
Wolna encyklopedia
609 000+ haseł

Italiano
L'enciclopedia libera
574 000+ voci

Português
A enciclopédia livre
483 000+ artigos

Русский
Свободная энциклопедия
396 000+ статей

Nederlands
De vrije encyclopedie
540 000+ artikelen

User Interface Complexity

Tuesday, January 26, 2010

Web Sites

PHP

Wicket

JSP

Seam

JSF

Tapestry

Ajax Sugar

JQuery

Dojo

Scriptaculous

MooTools

Full RIA

Plugin	JavaScript
Flex	SmartClient
Silverlight	GWT
JavaFX	ExtJS

Client Side

Server Driven

vaadin }>

ZK

ICEFaces

UI logic runs in browser (as JavaScript or in plugin)

Client Side

Server Driven

UI logic runs in server (framework updates UI in browser)

vaadin }>

Google Web Toolkit

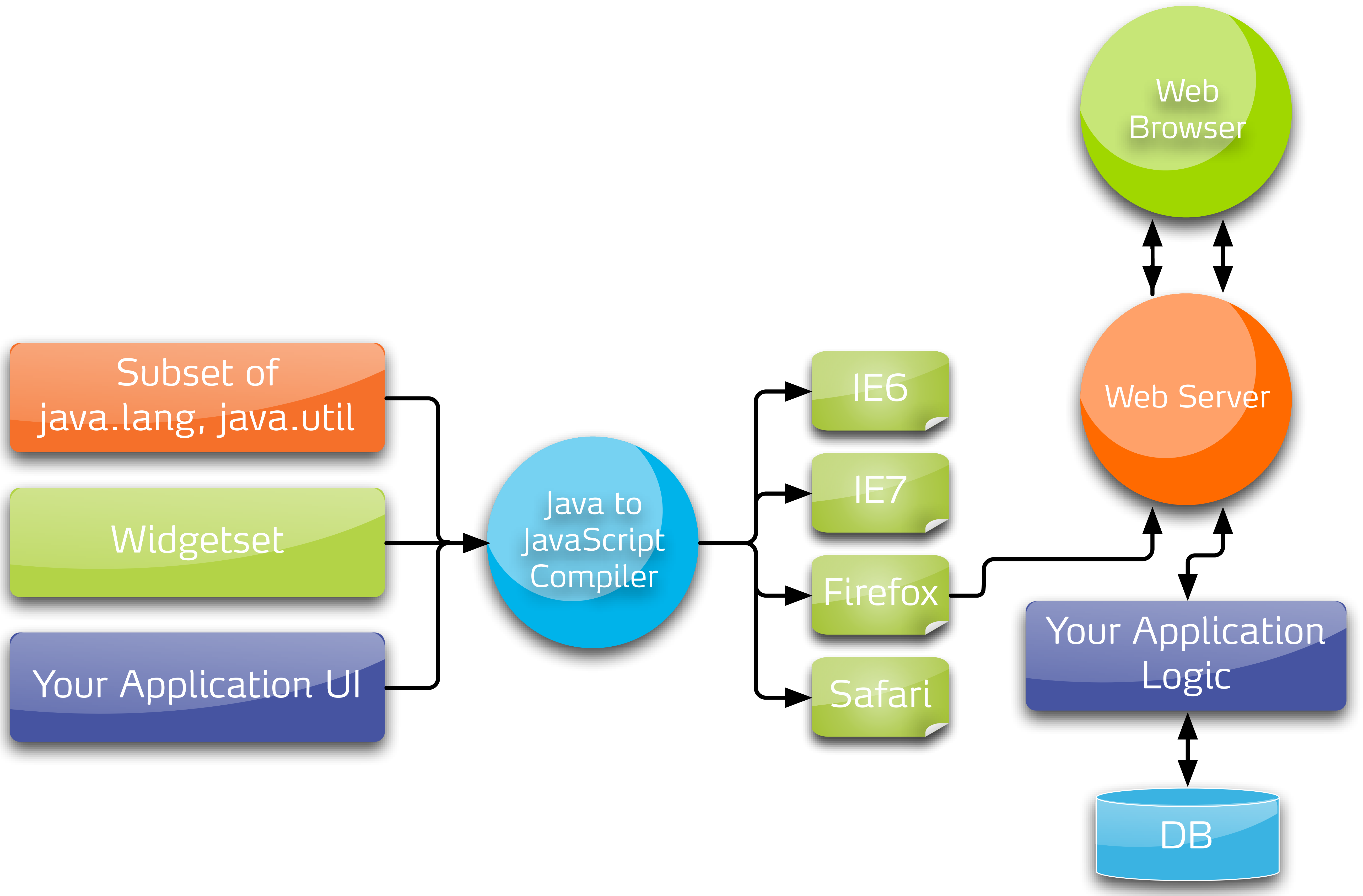
vaadin }>

Apache -licensed

User Interface Framework

for building RIA in Java

vaadin }>


Subset of java.lang, java.util
Widgetset
Your Application UI
Java to JavaScript Compiler
IE6
IE7
Firefox
Safari
Web Browser
Web Server
Your Application Logic
DB

Vaadin

vaadin }>

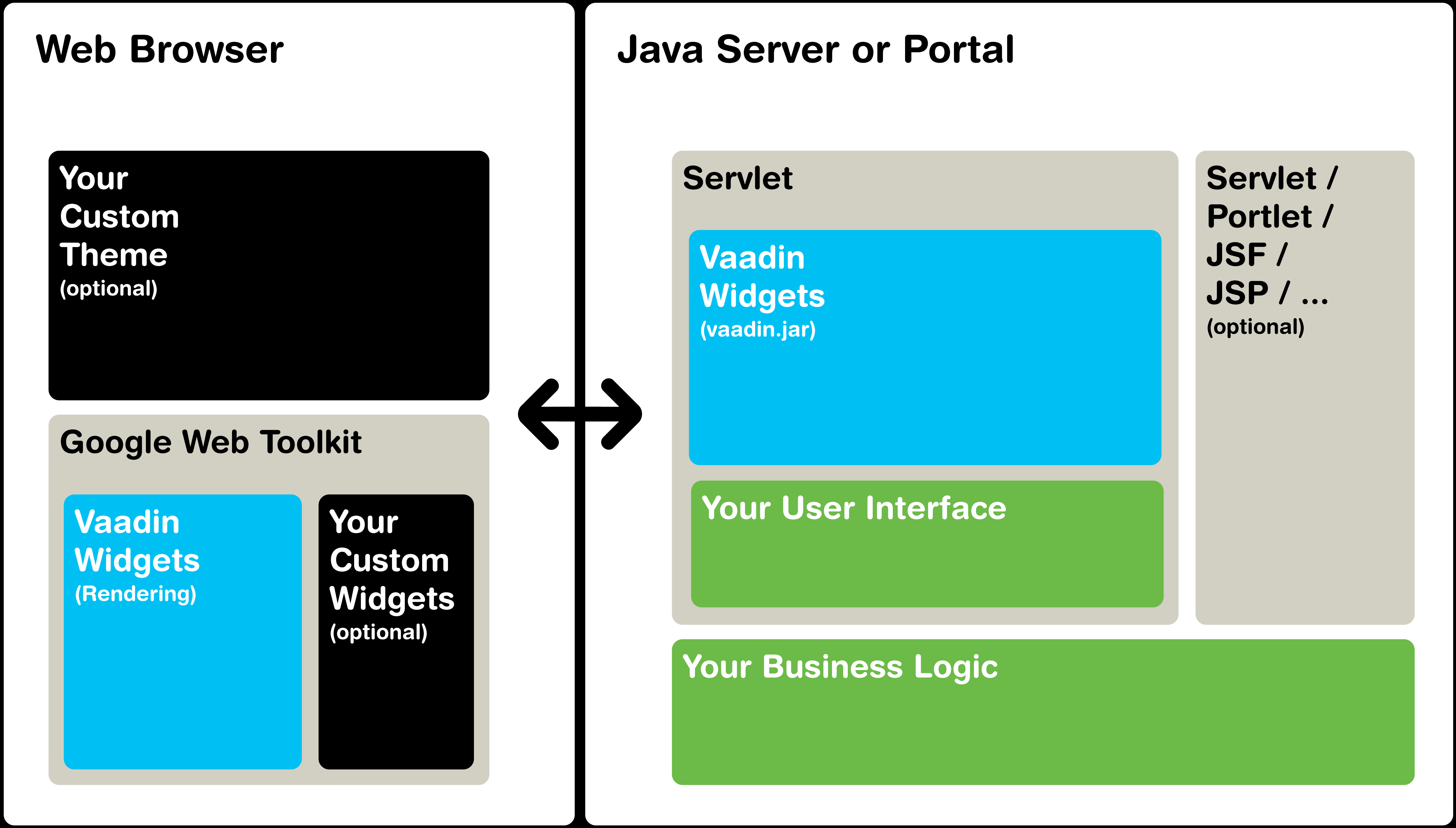
Apache -licensed

User Interface Framework

for building RIA in 100% Java

vaadin }>

architecture

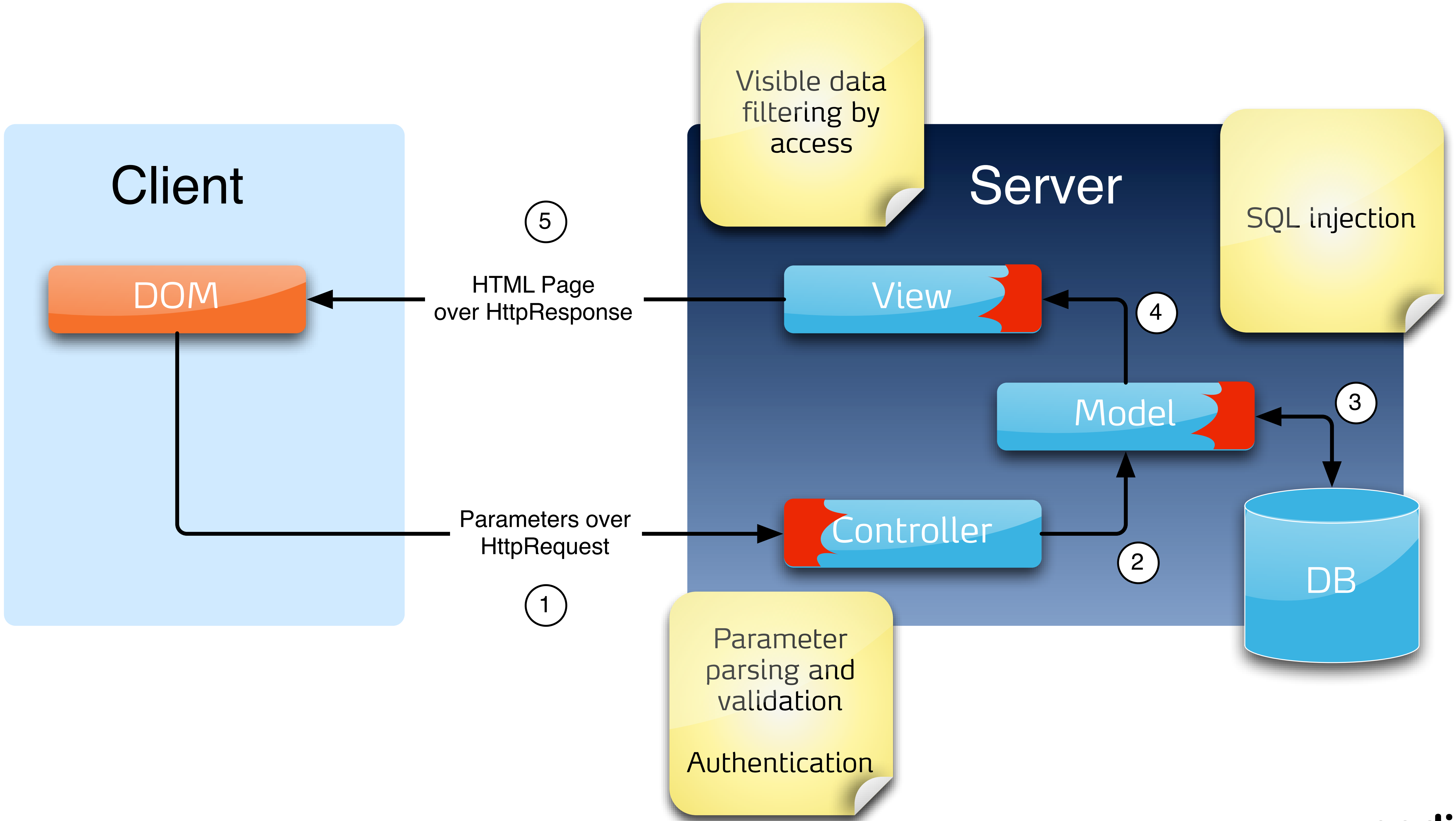


Web Browser
Your Custom Theme (optional)
Google Web Toolkit
Vaadin Widgets (Rendering)
Your Custom Widgets (optional)
Java Server or Portal
Servlet
Vaadin Widgets (vaadin.jar)
Your User Interface
Servlet / Portlet / JSF / JSP / ... (optional)
Your Business Logic

Security

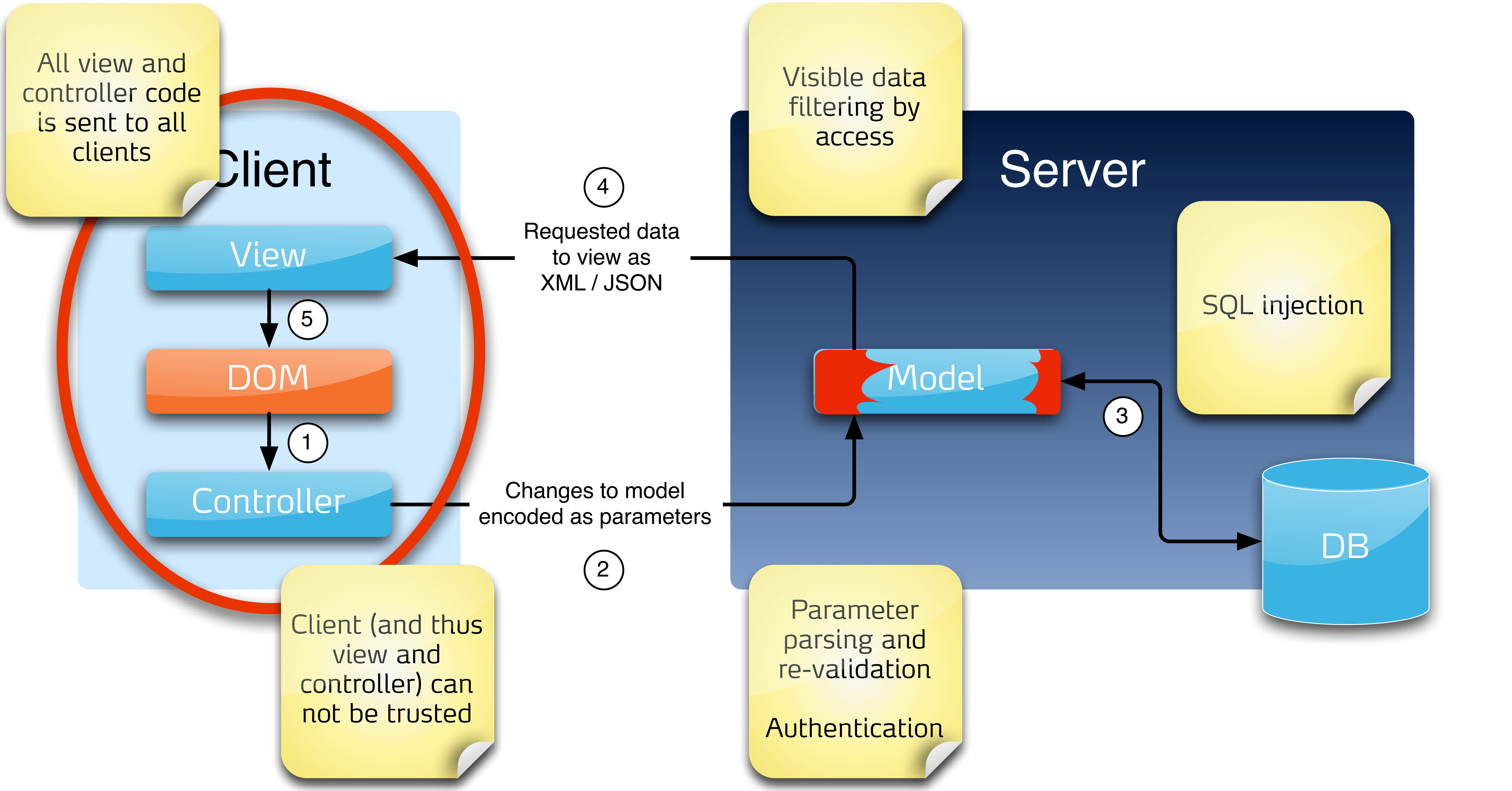
vaadin }>

“Web 1.0”


Client
DOM
Server
Visible data filtering by access
SQL injection
View
Model
Controller
DB
Parameter parsing and validation
Authentication
HTML Page over HttpResponse
Parameters over HttpRequest
1
2
3
4
5

vaadin }>

Client Side RIA

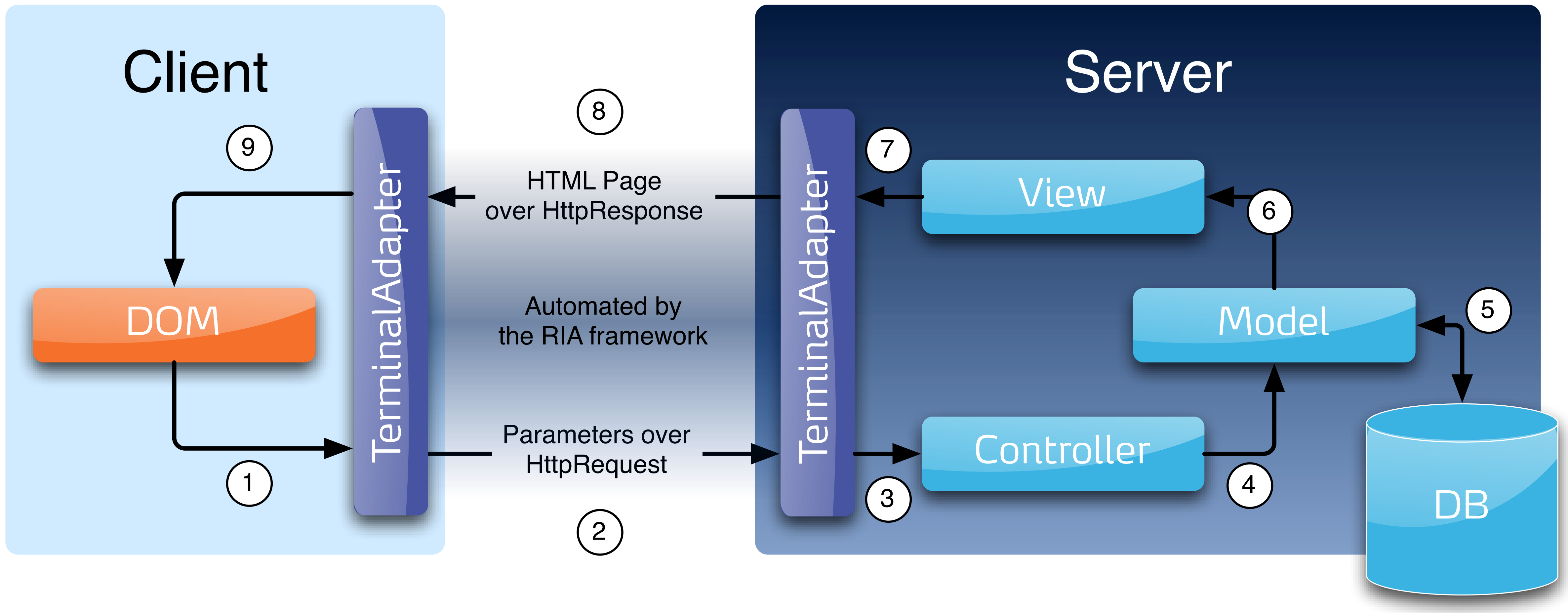

All view and controller code is sent to all clients
Client
View
5
DOM
1
Controller
Client (and thus view and controller) can not be trusted
4
Requested data to view as XML / JSON
Changes to model encoded as parameters
2
Visible data filtering by access
Server
SQL injection
Model
3
DB
Parameter parsing and re-validation
Authentication

Rule #1

Never trust the browser

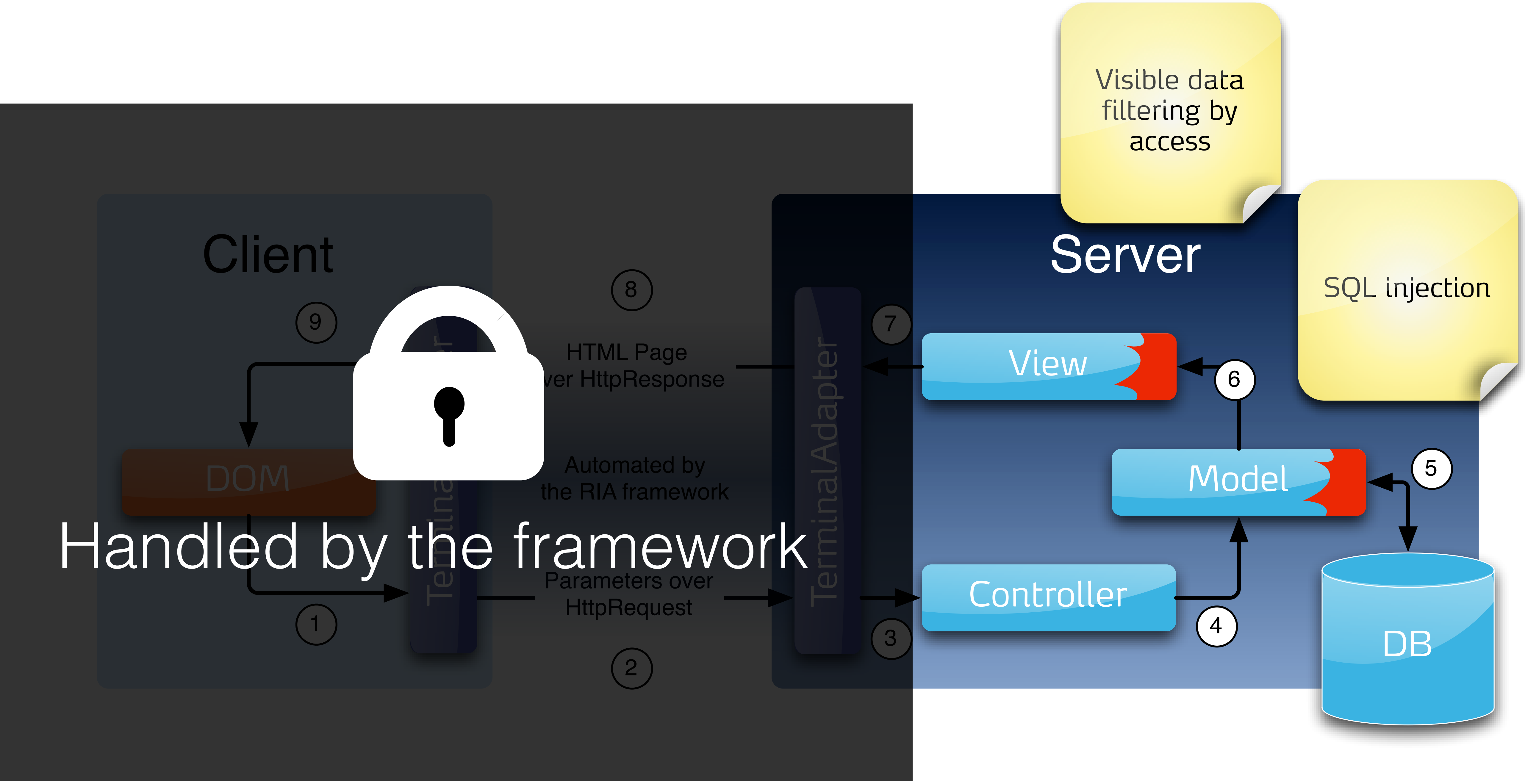
vaadin }>

Server Driven RIA


Client
Server
9
8
7
TerminalAdapter
HTML Page over HttpResponse
TerminalAdapter
View
6
DOM
Automated by the RIA framework
Model
5
Parameters over HttpRequest
Controller
DB
1
2
3
4

vaadin }>

Server Driven RIA



vaadin }>

Rule #2

Complexity is a hiding-place for security-flaws

vaadin }>

complexity

Aspect	Server Driven	Client Side
No access to server resources	-	X
Web-service API design	-	X
Communication design	-	X
Client-side validation	-	X
Server-side validation	X	X
Untrusted runtime	-	X
Exposed runtime	-	X
Highly dynamic language	-	X

Rule #3

Large surface: easy to attack, hard to defend

vaadin }>

Attack Surface: Web 1.0

- Web page HTML (presentation)
- Form parameters
- Parameter parsing
- Parameter validation
- Cross-site scripting (XSS)

Attack Surface: Client-side RIA

- Web page DOM (presentation)
- Form parameters (for hybrid solutions)
- Parameter parsing
- Parameter validation
- Cross-site scripting (XSS)

same as web 1.0

- UI logic can be
  - Evaluated: Black-box changes to white-box!
  - Changed
- Web services - a lot of **API is exposed** and can be called directly

Attack Surface: Server Driven RIA

- Web page DOM (presentation)
- ~~Form parameters (for hybrid solutions)~~
- ~~Parameter parsing~~
- Parameter validation
- Cross-site scripting (XSS)
- ~~UI logic can be~~
    - ~~Evaluated: Black box changes to white box!~~
    - ~~Changed~~
- ~~Web services - a lot of **API is exposed** and can be called directly~~

Breaking In

PayMate

Local demo

→ http://localhost:8280/paymate/

Online demo

→ http://vaadin.com/web/joonas/wiki/-/wiki/Main/RIA%20Security

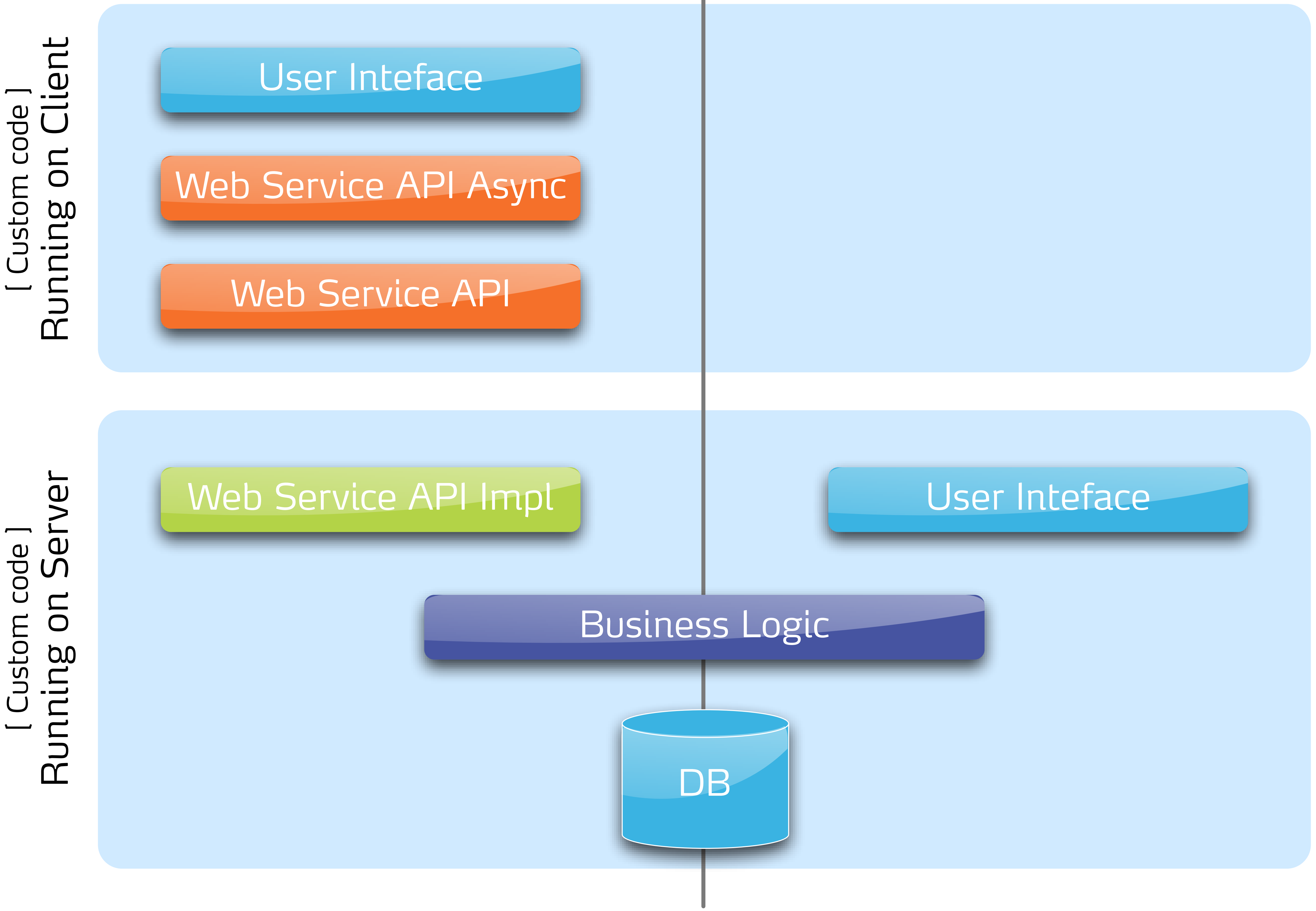
[no relation to paymate.com.au or paypal.com]

GWT version

Client Side RIA

Vaadin version

Server Driven RIA



Case #1
Injection

```
static public Account logIn(String email, String password) {

    Connection c = MockupDB.getConnection();
    Statement st;
    try {
        st = c.createStatement();
        ResultSet r = st.executeQuery("SELECT NAME,ID FROM ACCOUNT WHERE NAME='"
                + email + "' AND PASSWORD='" + password + "'");
        if (r.isBeforeFirst()) {
            r.next();
            return new Account(r.getString("NAME"), r.getInt("ID"));
        } else
            return null;

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

```java
static public Account logIn(String email, String password) {

    Connection c = MockupDB.getConnection();
    Statement st;
    try {
        st = c.createStatement();
        ResultSet r = st.executeQuery("SELECT NAME,ID FROM ACCOUNT WHERE NAME='"
                + email + "' AND PASSWORD='" + password + "'");
        if (r.isBeforeFirst()) {
            r.next();
            return new Account(r.getString("NAME"), r.getInt("ID"));
        } else
            return null;

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

vaadin }>

```
SELECT NAME,ID
FROM ACCOUNT
WHERE NAME='
' OR TRUE OR ''='
' AND PASSWORD=''
```

attack

SQL injection

vaadin }>

Injection

- Cures:
  - **Isolation: Keep data and execution separate**
  - Validation: Reject suspicious data
  - Escaping: Modify the data to keep it from affecting the execution

Client Side RIA	**vulnerable**
Server Driven RIA	**vulnerable**

Case #2
Double validation

vaadin }>

Missing double validation

- It is often convenient to do data some validation in the user interface logic
- Attacker can always bypass any validation done in the browser
- Thus **everything must be validated (again) in the server!**
- Lack of double validation is almost impossible to notice in testing or normal use

attack

rewriting client-side logic

vaadin }>

attack

forging http transport

vaadin }>

POST Data

```
4ï¿¿0ï¿¿7ï¿¿http://localhost:8080/paymate/client-
side/com.paymate.gwt.PayMateApplication/ï
¿¿29F4EA1240F157649C12466F01F46F60ï
¿¿com.paymate.gwt.client.PayMateServiceï
¿¿sendMoneyï¿¿Dï¿¿java.lang.Stringï
¿¿joonas@vaadin.comï¿¿1ï¿¿2ï¿¿3ï¿¿4ï¿¿2ï¿¿5ï¿¿6ï
¿¿-99999ï¿¿7ï¿¿
```

```
var xhr = document.body.childNodes[5].contentWindow.XMLHttpRequest;
Override the original XMLHttpRequest implementation
xhr.prototype.originalSend = xhr.prototype.send;
xhr.prototype.send = function(a) {

    Create UI for our hack tool
    var panel = document.createElement("DIV");
    panel.innerHTML = "<textarea id='postdata' cols=80 rows=20> "+
      "</textarea><br/><button id='postbutton'>Post</button>";
    document.body.appendChild(panel);
    document.getElementById('postdata').value=a;

    Do the sending when the button is pressed
    var t = this; document.getElementById('postbutton').
    addEventListener("click",function() {
        t.originalSend(document.getElementById('postdata').value);
        document.body.removeChild(panel);
    }, true);
};
```

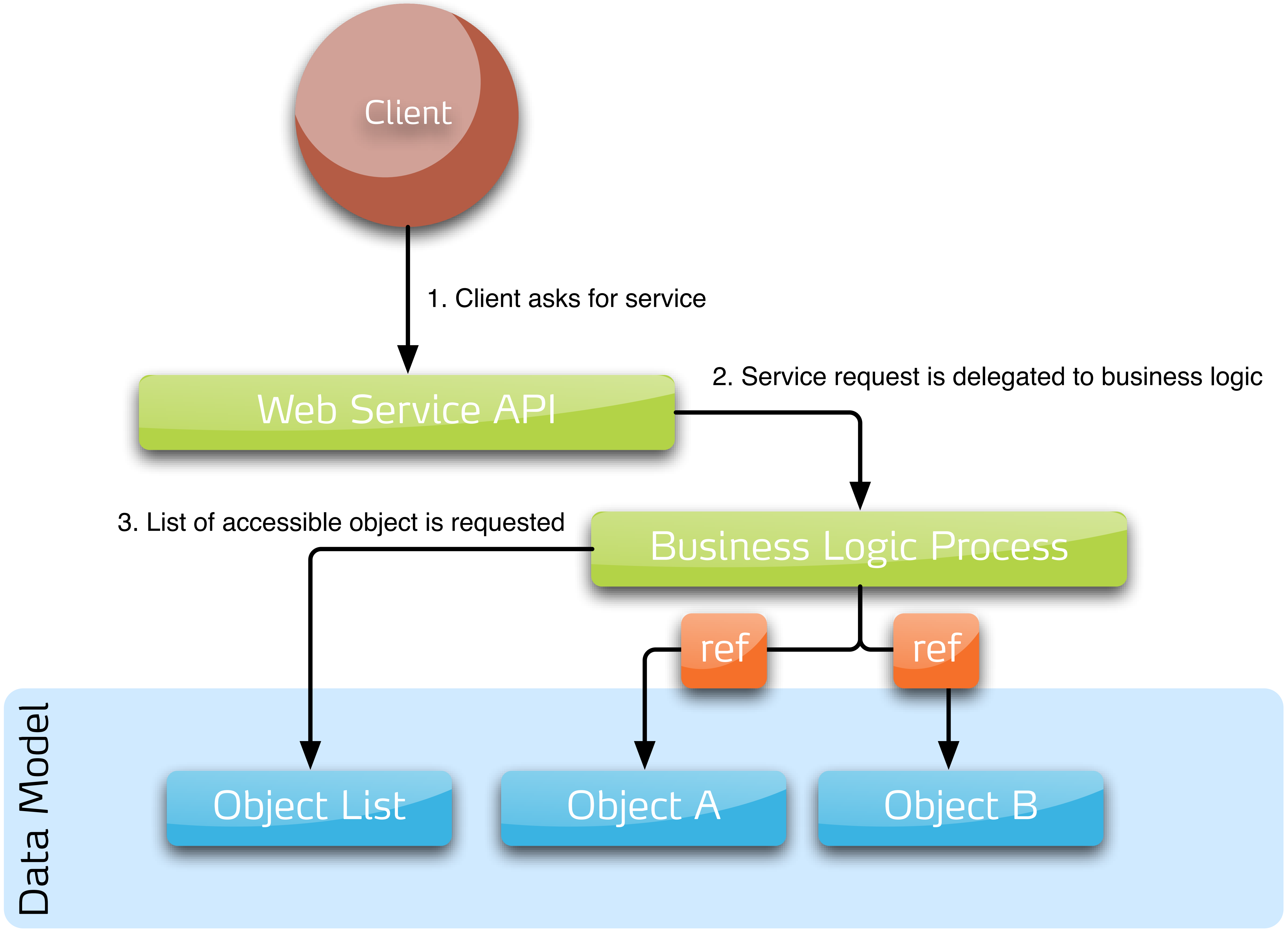
Double validation

- Cures:
  - Never skip server-side validation
  - Code review is a must - testing does not help
  - Never think server-validation could be seen as “extra work” that will be added later in the project

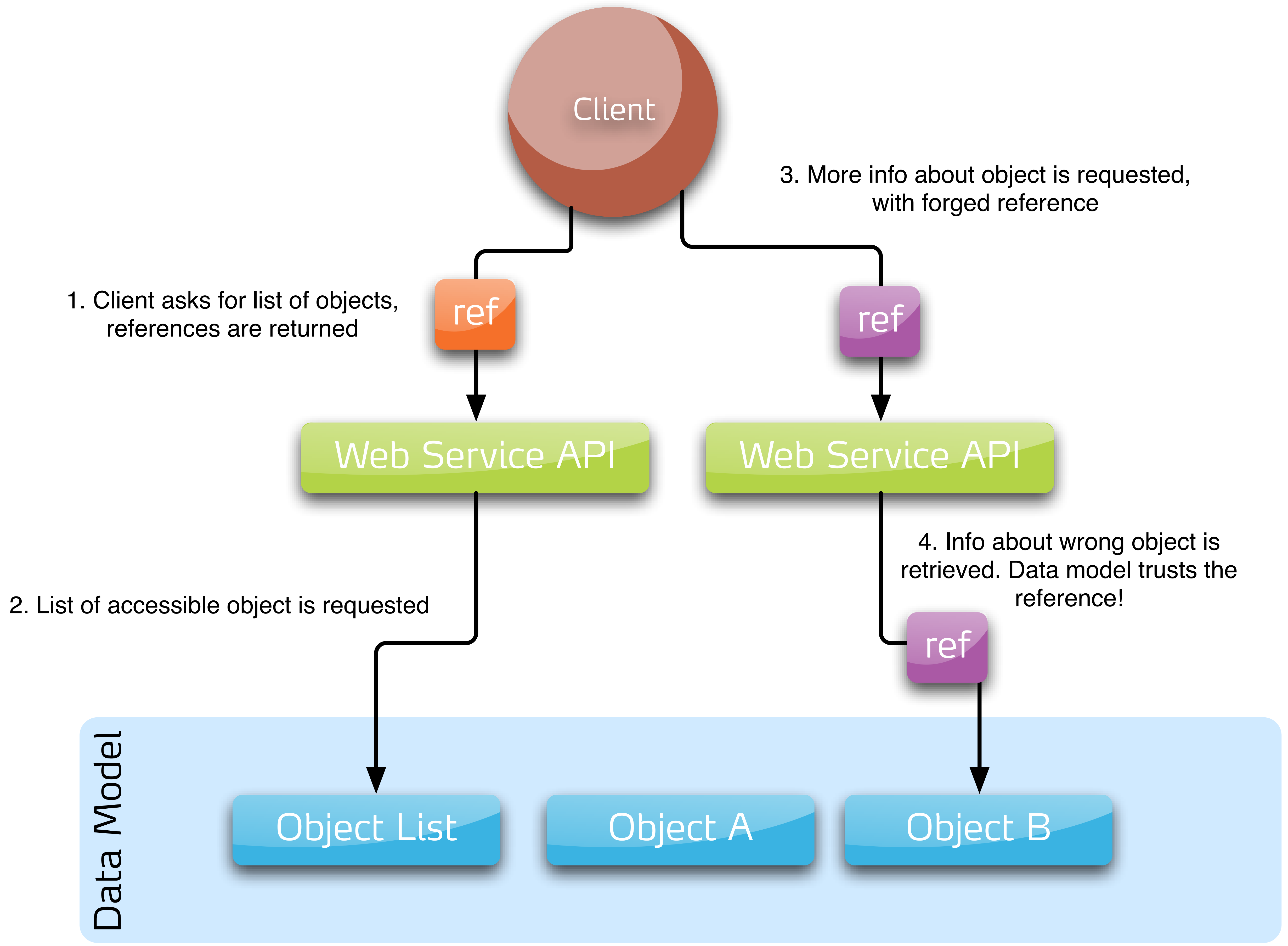
Client Side RIA	**vulnerable**
Server Driven RIA	**not vulnerable**

Case #3
Forging references

vaadin }>


Client
1. Client asks for service
Web Service API
2. Service request is delegated to business logic
3. List of accessible object is requested
Business Logic Process
ref
ref
Data Model
Object List
Object A
Object B

vaadin }>


Client
3. More info about object is requested, with forged reference
1. Client asks for list of objects, references are returned
ref
ref
Web Service API
Web Service API
4. Info about wrong object is retrieved. Data model trusts the reference!
2. List of accessible object is requested
ref
Data Model
Object List
Object A
Object B

vaadin }>

attack

requesting data with forged ids

vaadin }>

Forging references

- Cures:
  - Never pass any data-model level references to the client
  - Do all the access checks for each call from client

Client Side RIA	**vulnerable**
Server Driven RIA	**not vulnerable**

These bugs are just plain stupid!

[our team is smart enough to avoid them]

vaadin }>

really?

I can assure that	Yes	No
I would newer do mistakes like these		
Not even under pressure, late at night, on deadline		
And neither would the rest of the team, no-one		
Or the guys working for our off-shore contractor		
And we rigorously double review all of our code		
And trust we would spot 100% of these		
And we review all legacy code too		
We will newer have any “black boxes” in our system		

Rule #4

There will be bugs

summary

vaadin }>

Rule #1

Never trust the browser

Rule #2

More complexity - less security

Rule #3

Large surface is hard to defend

Rule #4

There will be bugs

vaadin }>

Questions
Comments

joonas@vaadin.com

skype://joonaslehtinen

+358-40-5035001

vaadin }>