RIA Security - Broken By Design

Joonas Lehtinen
Vaadin Ltd, CEO

@joonaslehtinen
#geecon #vaadin

vaadin }>

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

}
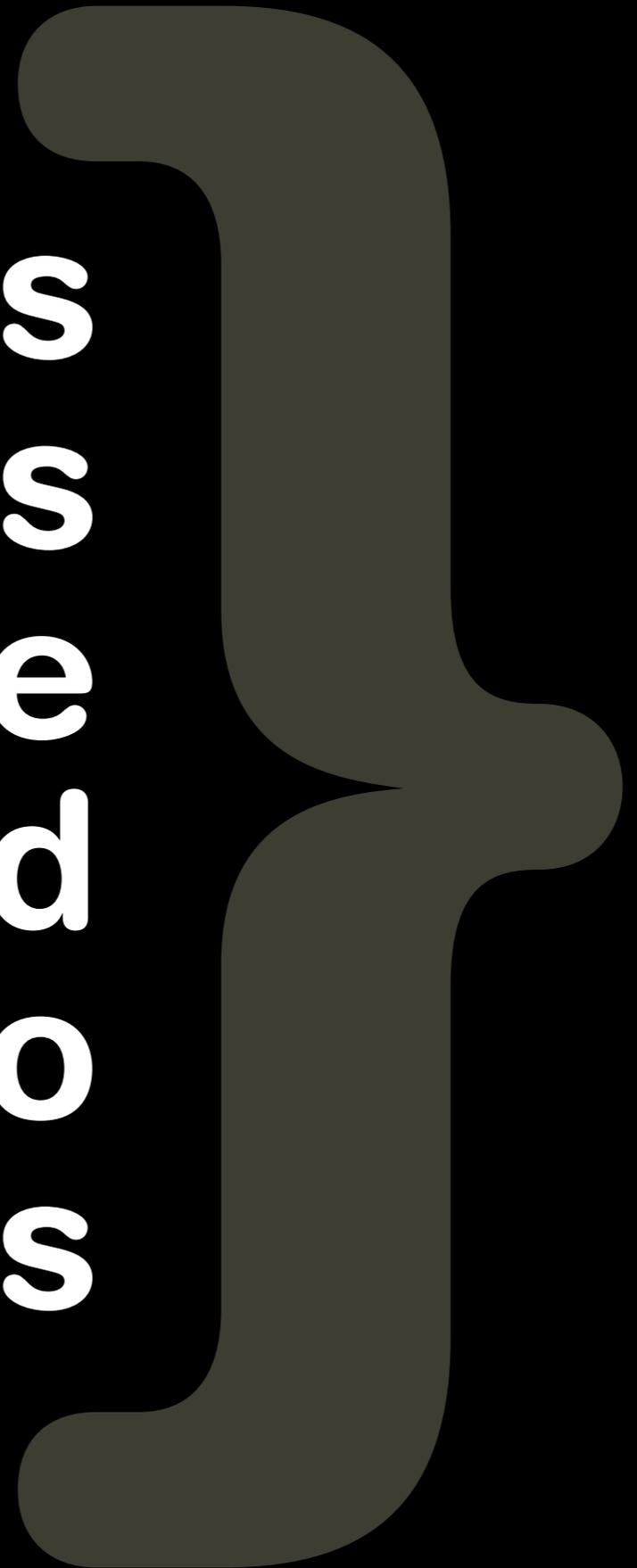
no system should be designed to be insecure

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
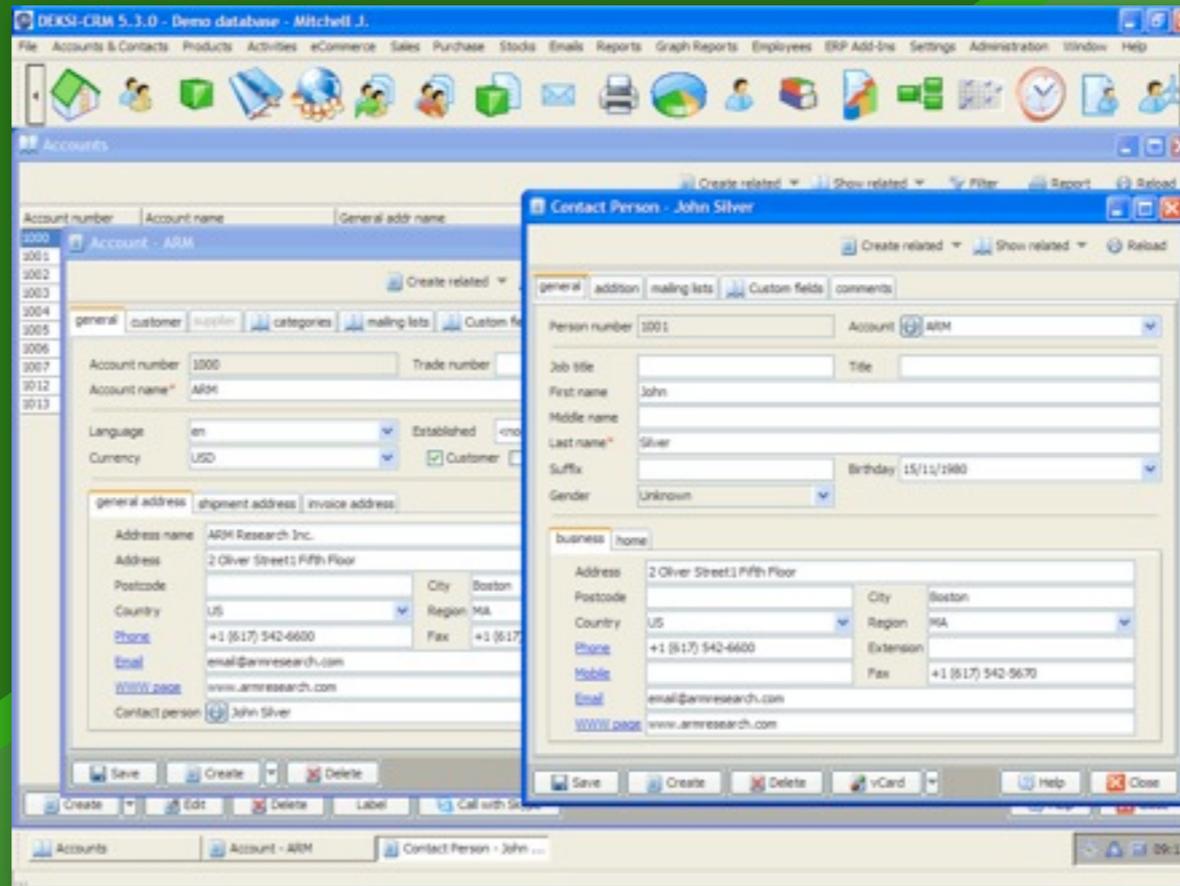
web
pages

**User Interface
Complexity**

Web Sites

PHP

Wicket

JSP

JSF

Spring MVC

Ajax Sugar

JQuery

Dojo

YUI

Full RIA

Plugin | JavaScript

Flex | SmartClient

Silverlight | GWT

JavaFX | ExtJS

Client Side
Server Side

vaadin }>

ZK

ICEFaces

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

Client Side

Server Side

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

vaadin }>

Google Web Toolkit

vaadin }>

Google Web Toolkit

Web Browser

Subset of java.lang, java.util

Widgetset

Your Application UI

Java to JavaScript Compiler

IE6

IE7

Firefox

Safari

Web Server

Your Application Logic

DB
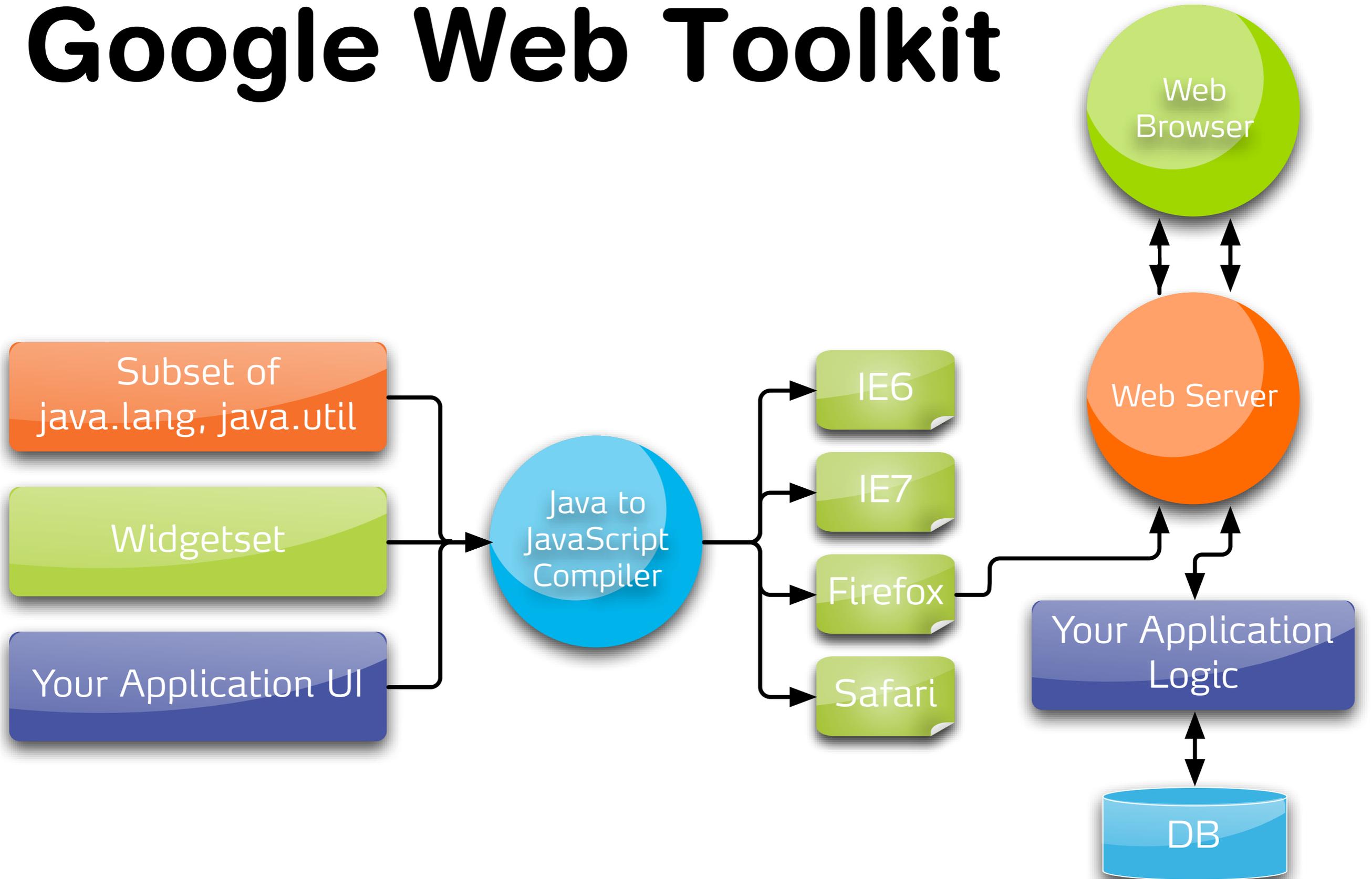
vaadin }>

Vaadin }

Vaadin Framework

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

"Web 1.0"

Visible data filtering by access

Server

SQL injection

5

Client

DOM

HTML Page over HttpResponse

View

4

Model

3

Parameters over HttpRequest

Controller

2

DB

1

Parameter parsing and validation

Authentication

vaadin }>

perjantaina 13. toukokuuta 2011
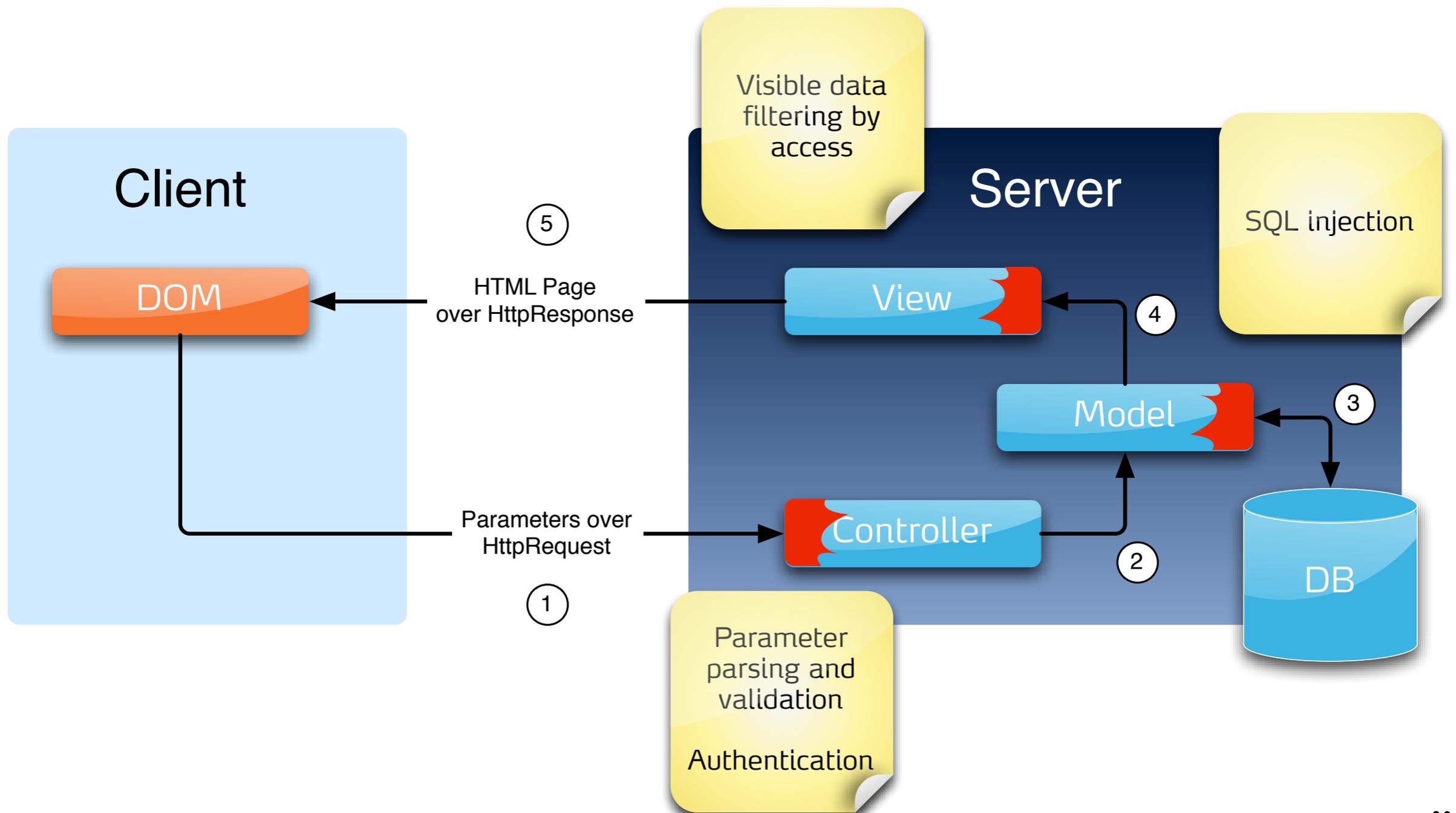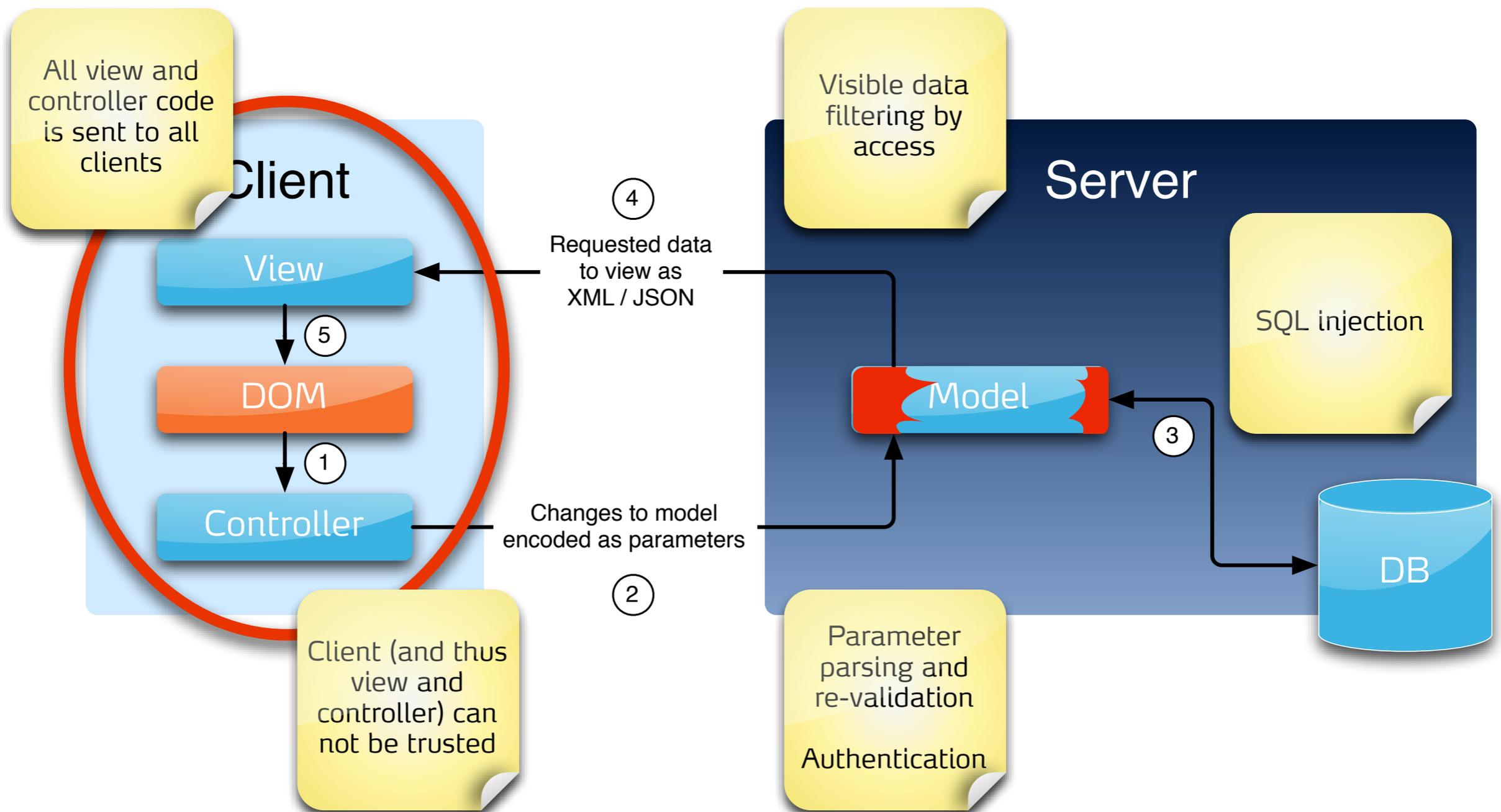
Rule #1

Never trust the browser

Server Side RIA
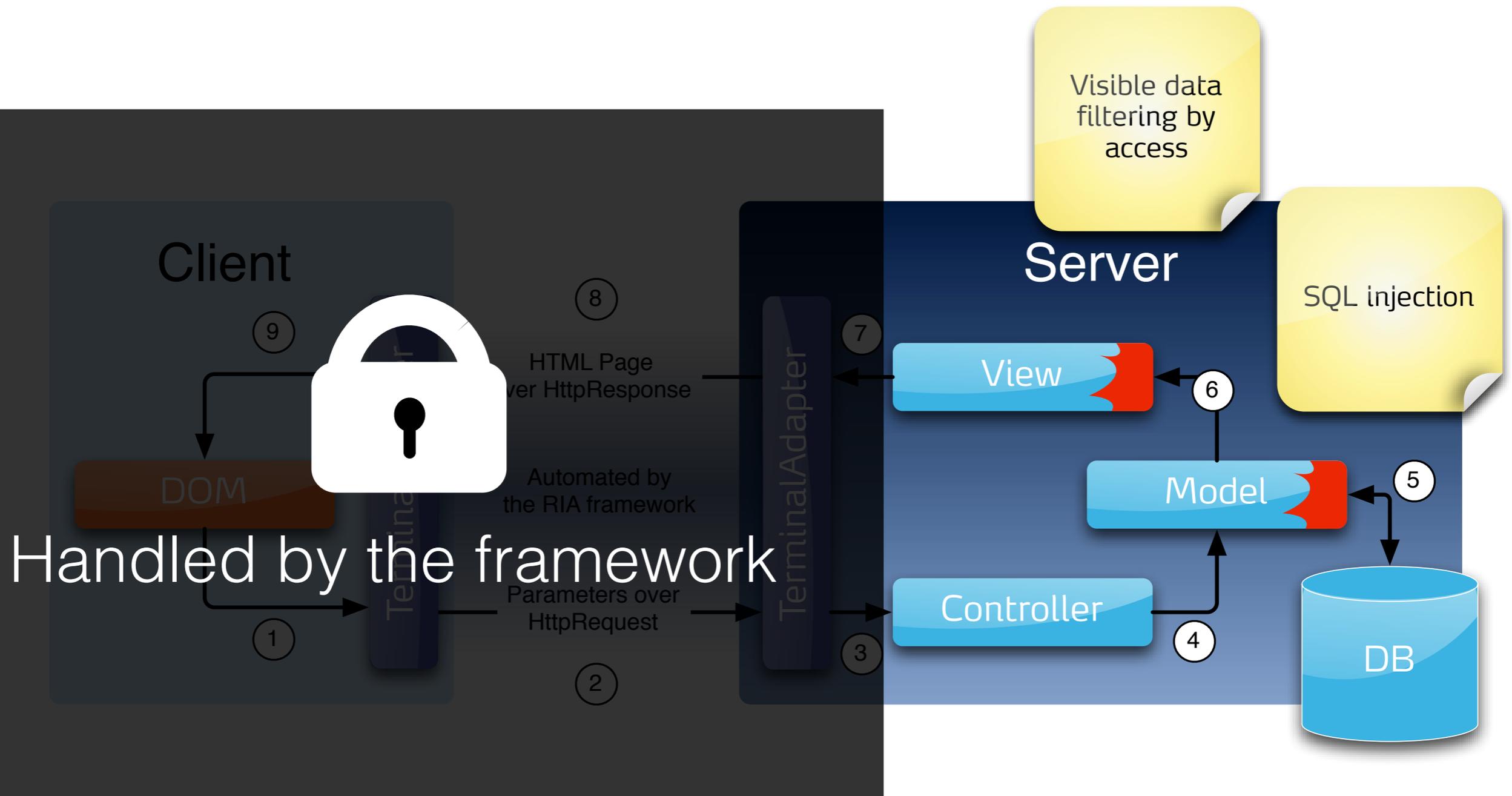
Rule #2

Complexity is a hiding-place for security-flaws

complexity

Aspect	Server Side	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 Side 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~~
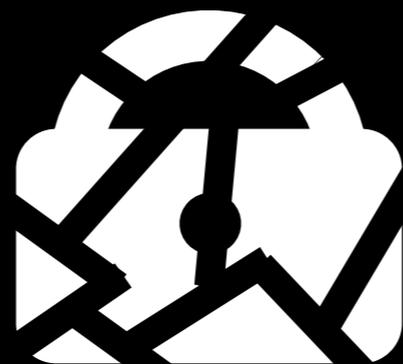
Breaking In

PayMate

Local demo

➜ http://localhost:8080/paymate/

Online demo

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

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

vaadin }>

GWT version
Client Side RIA
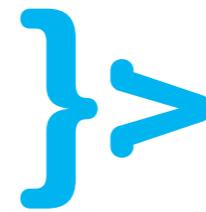
Vaadin version
Server Side RIA

[Custom code]
Running on Client

User Inteface

Web Service API Async

Web Service API

[Custom code]
Running on Server

Web Service API Impl

User Inteface

Business Logic

DB
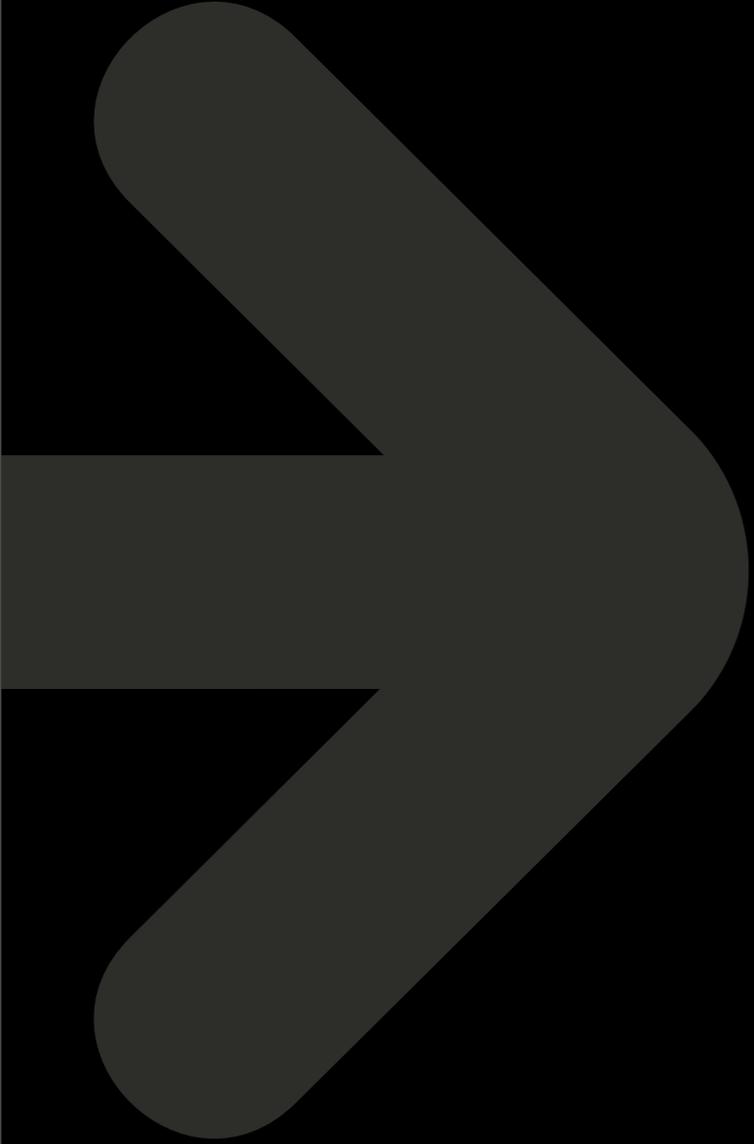
Case #1
Injection

```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;
}
```

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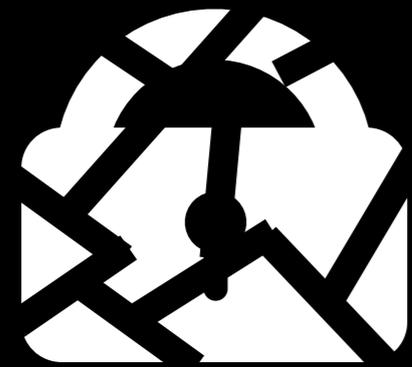
attack

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 Side RIA	**vulnerable**
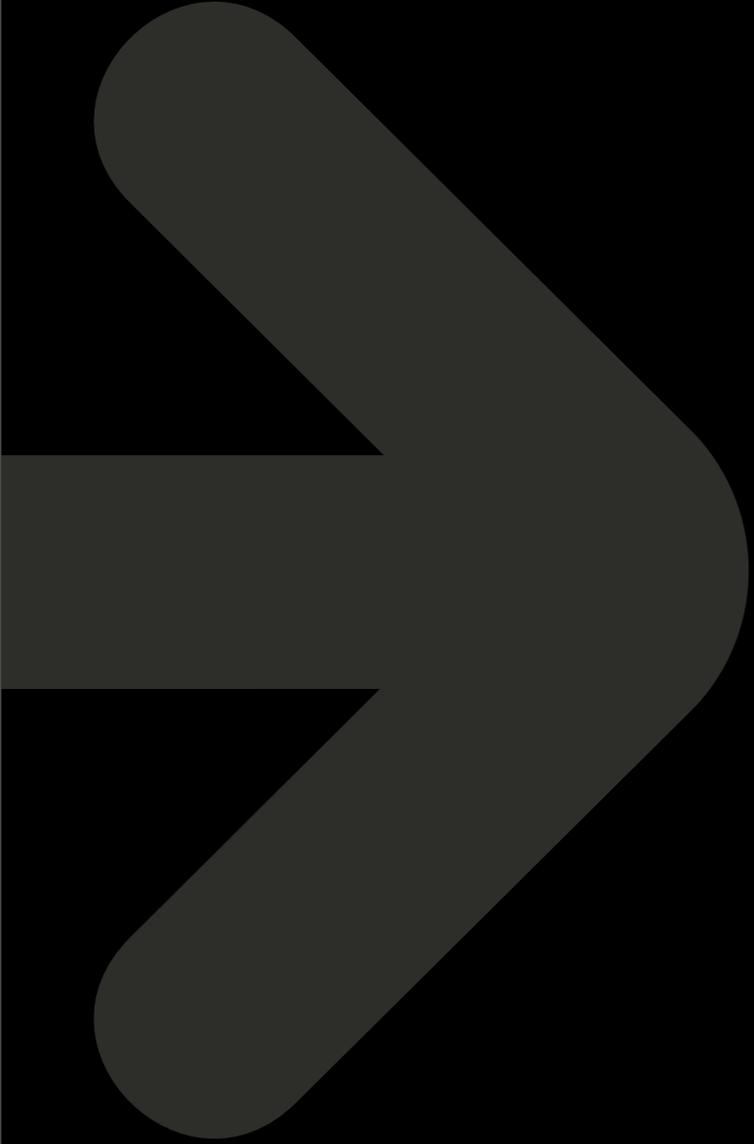
Case #2

Double validation

Missing double validation

- It is often convenient to do some data 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 validation

POST Data

4i¿¿0i¿¿7i¿¿http://localhost:8080/paymate/client-side/com.paymate.gwt.PayMateApplication/i¿¿29F4EA1240F157649C12466F01F46F60i¿¿com.paymate.gwt.client.PayMateServicei¿¿sendMoneyi¿¿Di¿¿java.lang.Stringi¿¿joonas@vaadin.comi¿¿1i¿¿2i¿¿3i¿¿4i¿¿2i¿¿5i¿¿6i¿¿-99999i¿¿7i¿¿

vaadin }>

```javascript
var xhr = document.body.childNodes[5].contentWindow.XMLHttpRequest;
```
Override the original XMLHttpRequest implementation
```javascript
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);
};
```

vaadin }>

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 Side RIA	**not vulnerable**

Case #3

Forging references

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

ref

ref

1. Client asks for list of objects,
references are returned

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 }>

perjantaina 13. toukokuuta 2011

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 Side 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 never 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
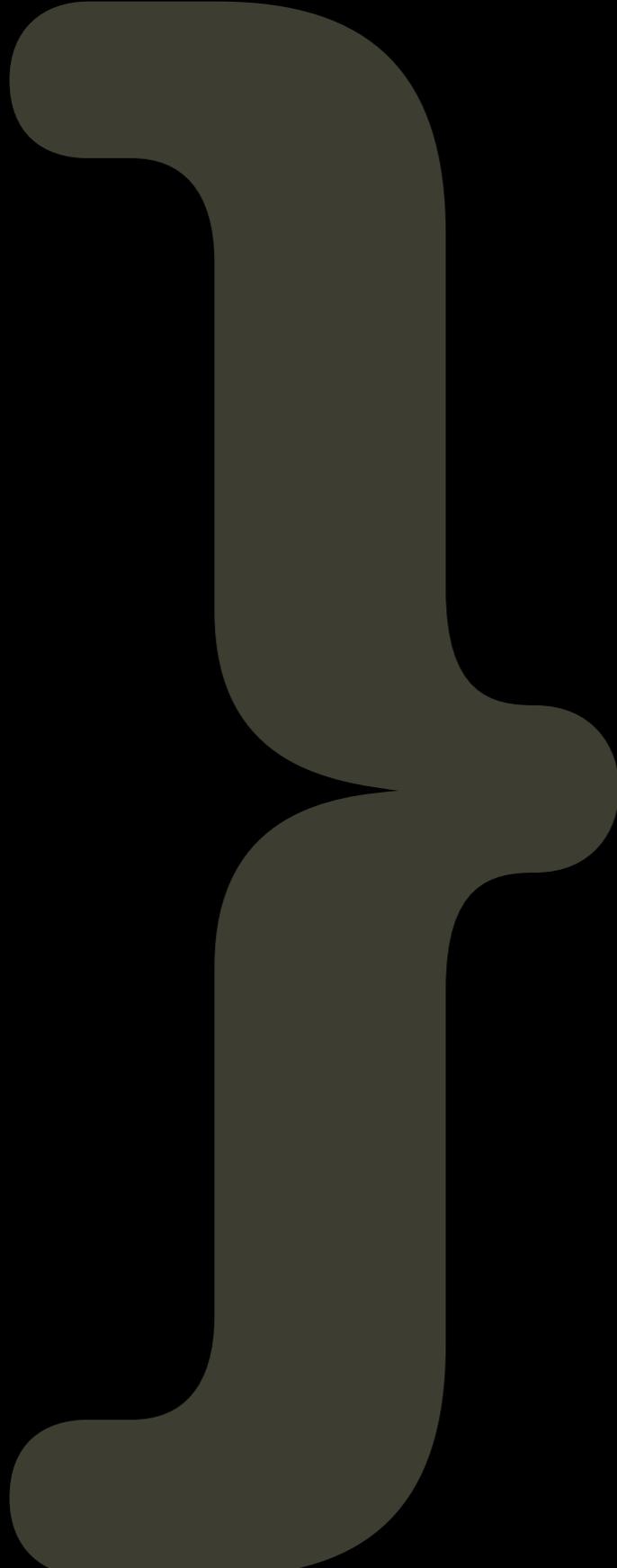
vaadin }>

} summary
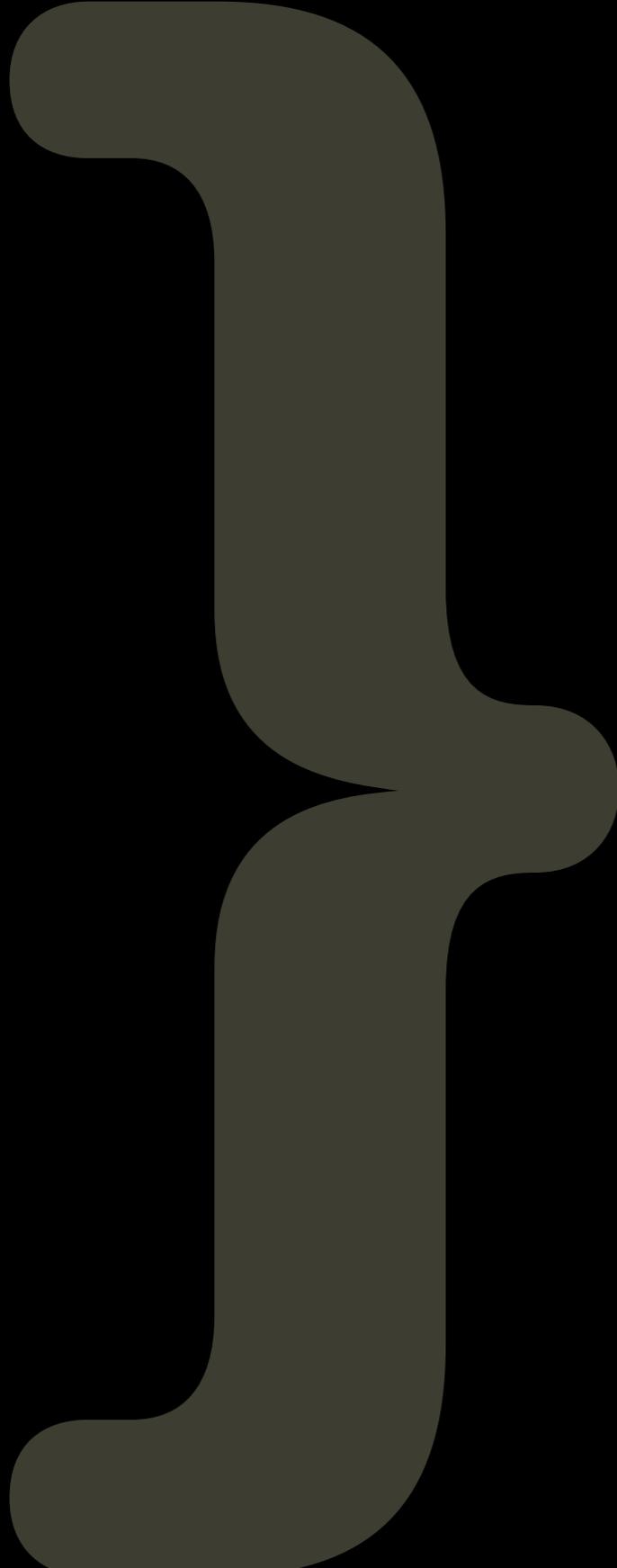
Rule #1
Never trust the browser

Rule #2
More complexity - less security

Rule #3
Large surface is hard to defend

Rule #4
There will be bugs

vaadin }>

**Questions
Comments**
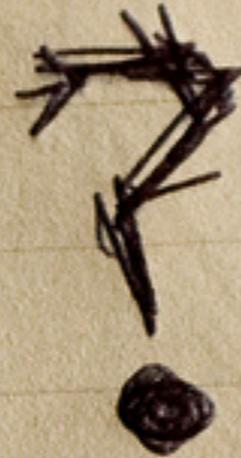
joonas@vaadin.com
+358-40-5035001
skype://joonaslehtinen