# Rock solid UI modeling using annotation processing

Case of study

@gdigugli

@jubaudry

# Speakers

- @gdigugli – Gilles Di Guglielmo
- Java Developer since 1999
- Software architect at

- @jubaudry – Julien Baudry
- Java Developer since 2007
- Senior developer at

LesFurets.com
Comparez futé

- ILOG – IBM
  - 2D Graphic toolkit
  - Rule Engine
- Prima-Solutions
  - SOA Platform for J2EE
  - Domain models code generators

- Prima-Solutions
  - SOA Platform for J2EE
  - Domain models code generators
  - Reinsurance software

# Agenda

Context

Quick demo

Modeling approach

Dependency Model

Field features

Extensions

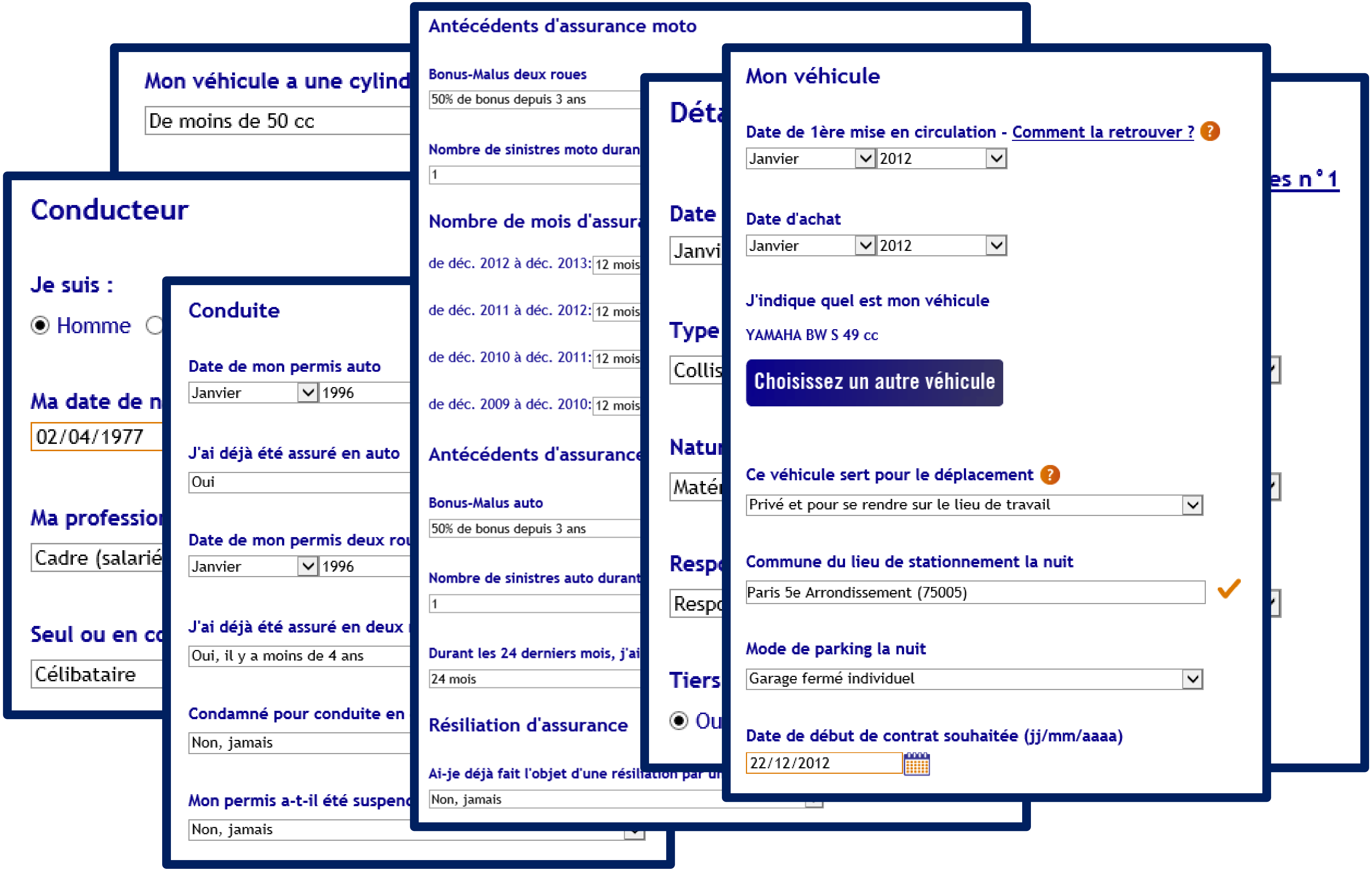
Live coding demo

Back to LesFurets.com

# Context at LesFurets.com

- 5 questions sets for an insurance aggregator
    - Car form (160 questions)
    - Motorbike form (180 questions)
    - Health form (50 questions)
    - Home form (70 questions)
    - Loan form (40 questions)
- A lot of questions with business rules linked by dependencies and business rules

# Sample : old question set for motorbike



**Mon véhicule a une cylind**

De moins de 50 cc

**Conducteur**

**Je suis :**

● Homme ○

**Ma date de n**

02/04/1977

**Ma professio**

Cadre (salarié

**Seul ou en co**

Célibataire

**Conduite**

**Date de mon permis auto**

Janvier 1996

**J'ai déjà été assuré en auto**

Oui

**Date de mon permis deux rou**

Janvier 1996

**J'ai déjà été assuré en deux**

Oui, il y a moins de 4 ans

**Condamné pour conduite en**

Non, jamais

**Mon permis a-t-il été suspen**

Non, jamais

**Antécédents d'assurance moto**

**Bonus-Malus deux roues**

50% de bonus depuis 3 ans

**Nombre de sinistres moto duran**

1

**Nombre de mois d'assur**

de déc. 2012 à déc. 2013: 12 mois

de déc. 2011 à déc. 2012: 12 mois

de déc. 2010 à déc. 2011: 12 mois

de déc. 2009 à déc. 2010: 12 mois

**Antécédents d'assurance**

**Bonus-Malus auto**

50% de bonus depuis 3 ans

**Nombre de sinistres auto durant**

1

**Durant les 24 derniers mois, j'ai**

24 mois

**Résiliation d'assurance**

**Ai-je déjà fait l'objet d'une résiliation par u**

Non, jamais

**Déta**

**Date**

Janvi

**Type**

Collis

**Natur**

Matér

**Resp**

Resp

**Tiers**

● Ou

**Mon véhicule**

**Date de 1ère mise en circulation - Comment la retrouver ?** ?

Janvier 2012

**Date d'achat**

Janvier 2012

**J'indique quel est mon véhicule**

YAMAHA BW S 49 cc

**Ce véhicule sert pour le déplacement** ?

Privé et pour se rendre sur le lieu de travail

**Commune du lieu de stationnement la nuit**

Paris 5e Arrondissement (75005) ✓

**Mode de parking la nuit**

Garage fermé individuel

**Date de début de contrat souhaitée (jj/mm/aaaa)**

22/12/2012

es n°1

# Nature of dependencies

- Visibility
  - I declare a claim -> question set for this claim appears
- Value range
  - I’ve been owning a car for one year -> constraint on the date for a claim should be later than the car’s purchase date
- Reset
  - I change the number of occurred claims from 2 to 1 -> previous details of claim number 2 should be dropped
- Validation
  - I change my date of birth -> I could not obtain my car license before being 18 years old

# Complexity and bug hell

- Historical design was based on a page scope
- All the rules between fields were embedded in each page code
- Business rules were directly written on the widget values without MVC pattern
- Page navigation was triggering model updates that were sent to the server

- Governance of the business rules between fields was difficult
- Lots of side effects between rules
- Improving or adding new rules provided a lot of regressions
- Dependencies between fields was not documented
- Adding new fields or shuffling the fields order required a lot of testing

# The CSS ids : a limited starting point

- All the form fields were still having a CSS class and an ID for CSS skinning
    - No real taxonomy
    - No guaranty that CSS ids are unique
    - Styling is evolving with his own constraints
    - Not the original purpose of CSS

Using CSS on web forms hides an implicit model that could be leveraged

# Requirements

- Ensuring non regression even with frequent changes on forms
  - No unexpected side effects between business rules
  - Make unit testing possible
- Enabling a fast and up-to-date understanding of the form complexity
- Reducing the maintenance effort
- Supporting fields shuffle
- Supporting AB testing

# MDL4UI our OSS sandbox

- Available on github
  - http://github.com/lesfurets/mdl4ui
- Full framework and example
- Based on GWT and Twitter bootstrap
- Ready to fork and play
- Requires Java 6+ and Maven
- 50 sec to build and run from scratch

**WE ACCEPT PULL REQUESTS**

Context

**Quick demo**

Modeling approach

Dependency model

Field features

Extensions

Live coding demo

Back to LesFurets.com

Context

Quick demo

Modeling approach

Dependency model

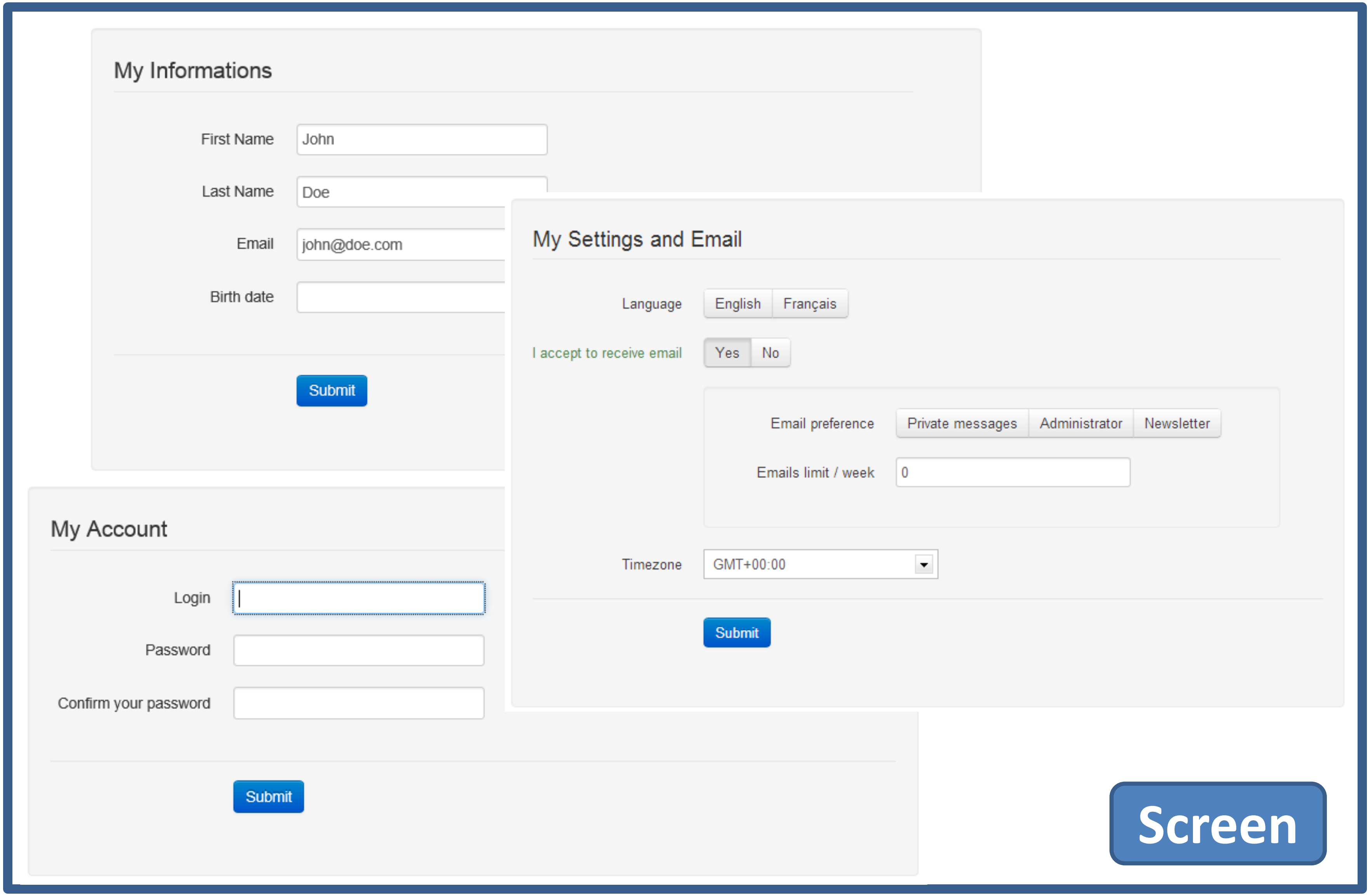
Field features

Extensions

Live coding demo

Back to LesFurets.com

# MDL4UI model concepts



## My Informations

First Name: John

Last Name: Doe

Email: john@doe.com

Birth date:

## My Account

Login:

Password:

Confirm your password:

## My Settings and Email

Language: English | Français

I accept to receive email: Yes | No

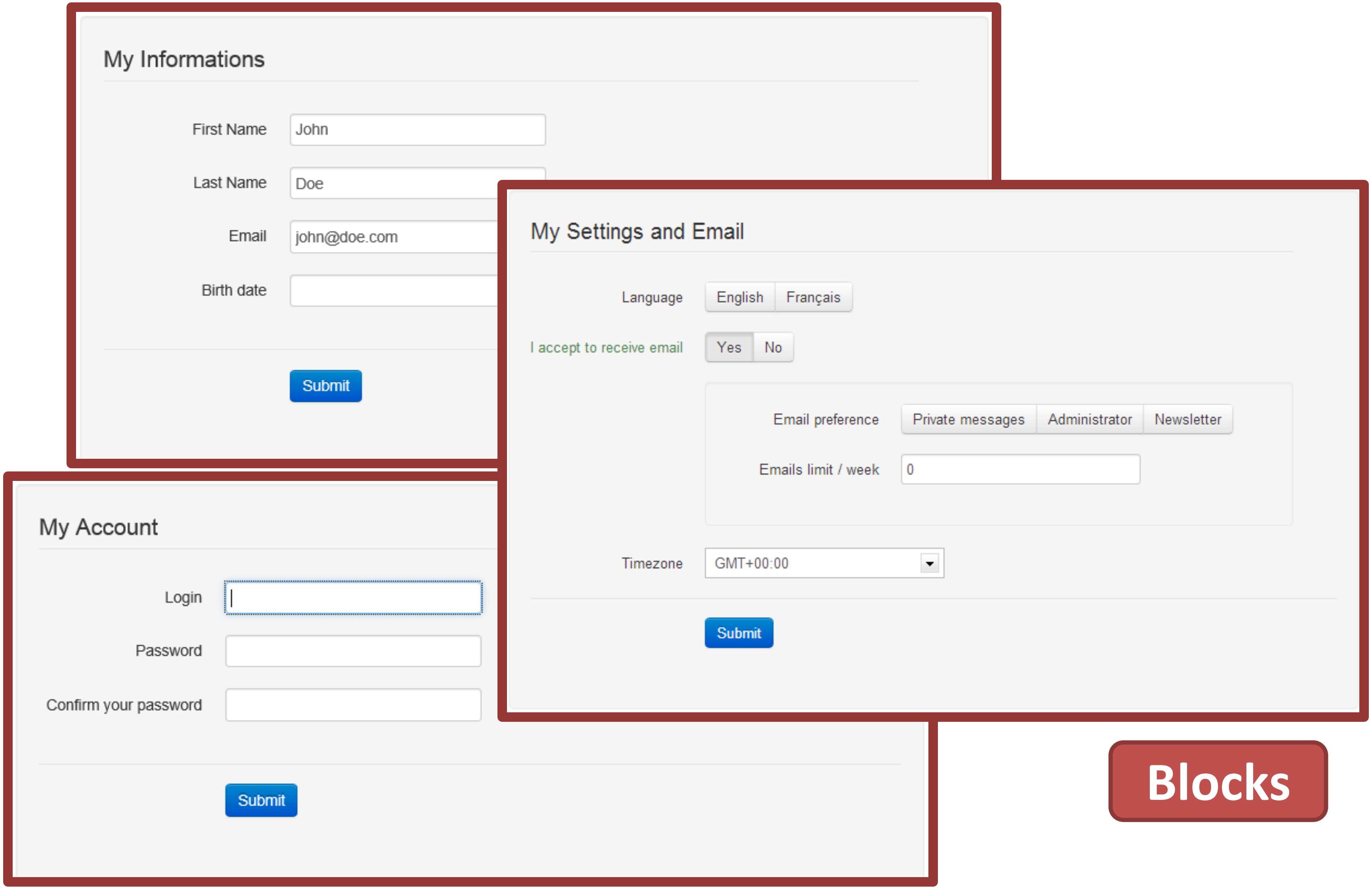
Email preference: Private messages | Administrator | Newsletter

Emails limit / week: 0

Timezone: GMT+00:00

Screen

# MDL4UI model concepts



## My Informations

First Name: John

Last Name: Doe

Email: john@doe.com

Birth date:

## My Account

Login:

Password:

Confirm your password:

## My Settings and Email

Language: English | Français

I accept to receive email: Yes | No

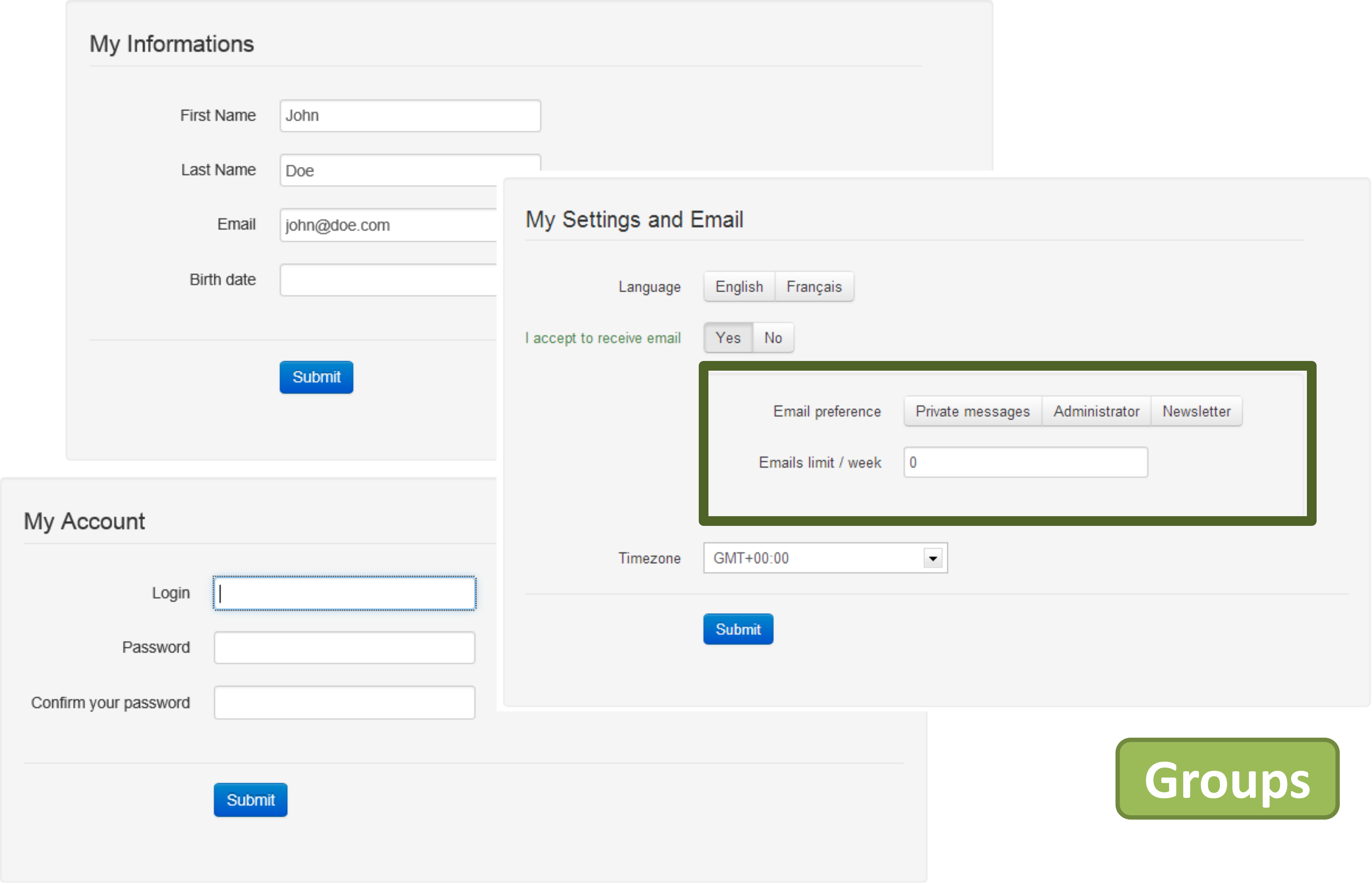
Email preference: Private messages | Administrator | Newsletter

Emails limit / week: 0

Timezone: GMT+00:00

**Blocks**

# MDL4UI model concepts



My Informations

First Name: John

Last Name: Doe

Email: john@doe.com

Birth date:

Submit

My Account

Login:

Password:

Confirm your password:

Submit

My Settings and Email

Language: English | Français

I accept to receive email: Yes | No

Email preference: Private messages | Administrator | Newsletter

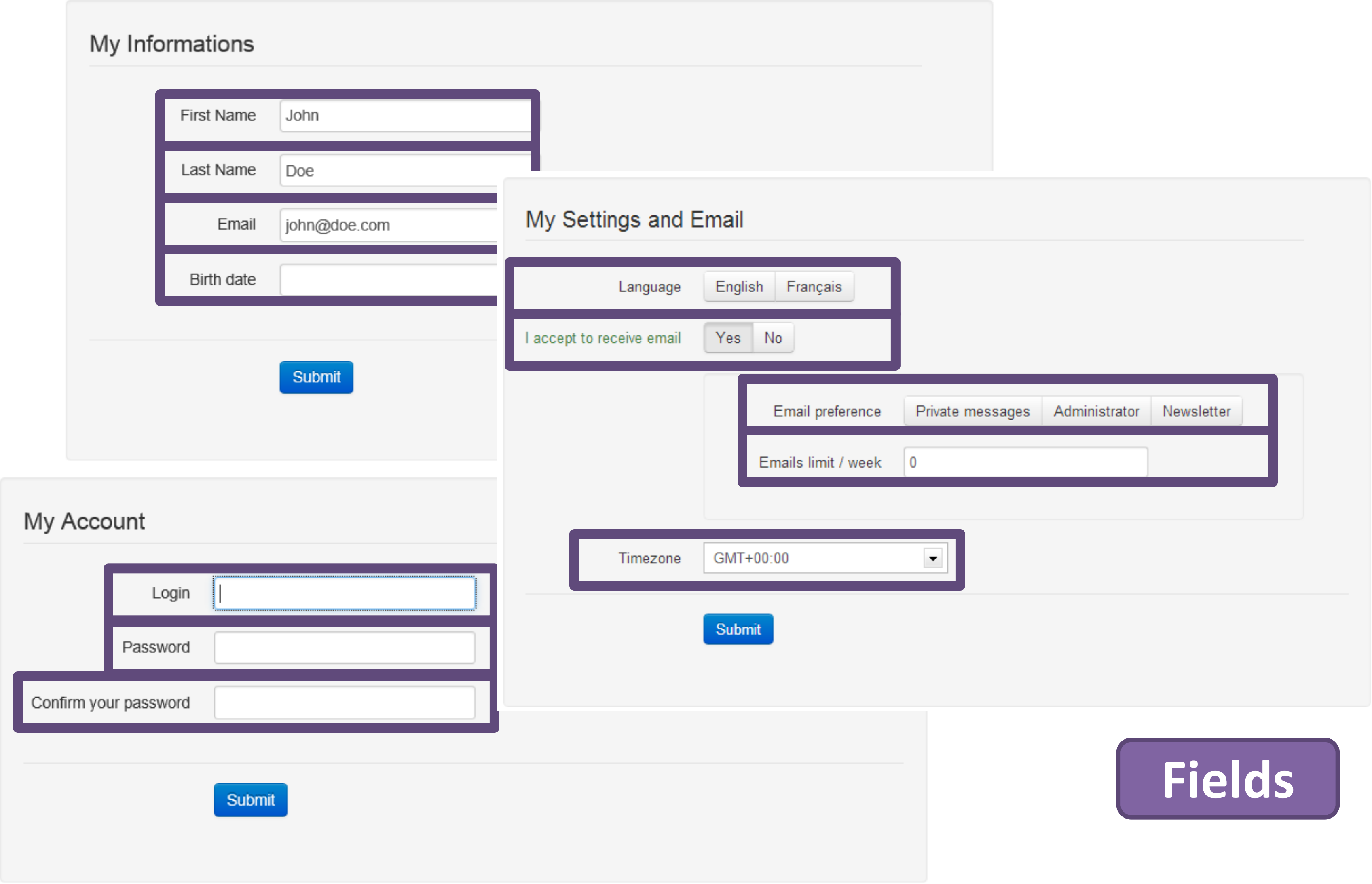
Emails limit / week: 0

Timezone: GMT+00:00

Submit

**Groups**

# MDL4UI model concepts



### My Informations

First Name: John

Last Name: Doe

Email: john@doe.com

Birth date:

### My Account

Login:

Password:

Confirm your password:

### My Settings and Email

Language: English | Français

I accept to receive email: Yes | No

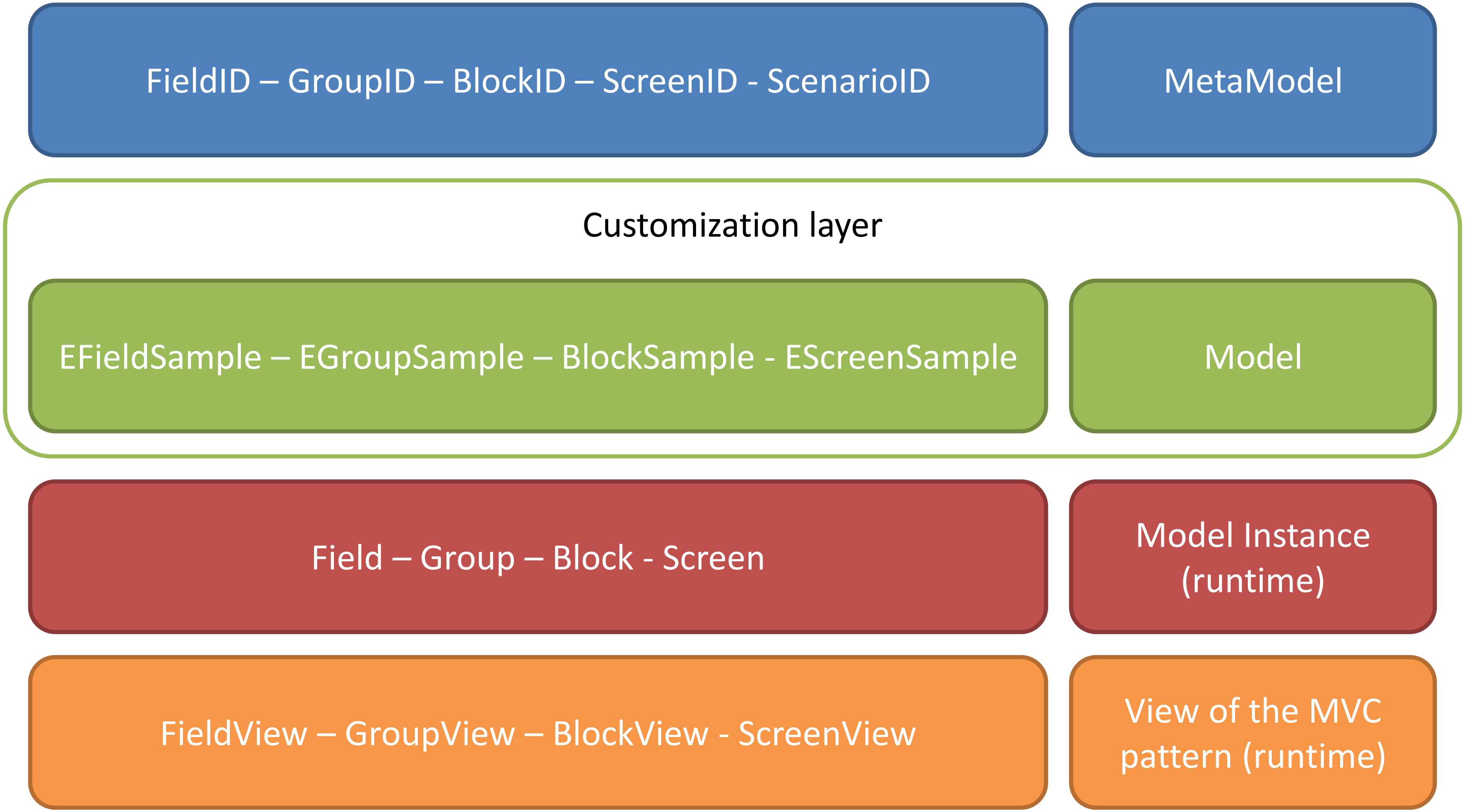
Email preference: Private messages | Administrator | Newsletter

Emails limit / week: 0

Timezone: GMT+00:00

**Fields**

# Introducing MDL4UI model layers



FieldID – GroupID – BlockID – ScreenID - ScenarioID

MetaModel

Customization layer

EFieldSample – EGroupSample – BlockSample - EScreenSample

Model

Field – Group – Block - Screen

Model Instance (runtime)

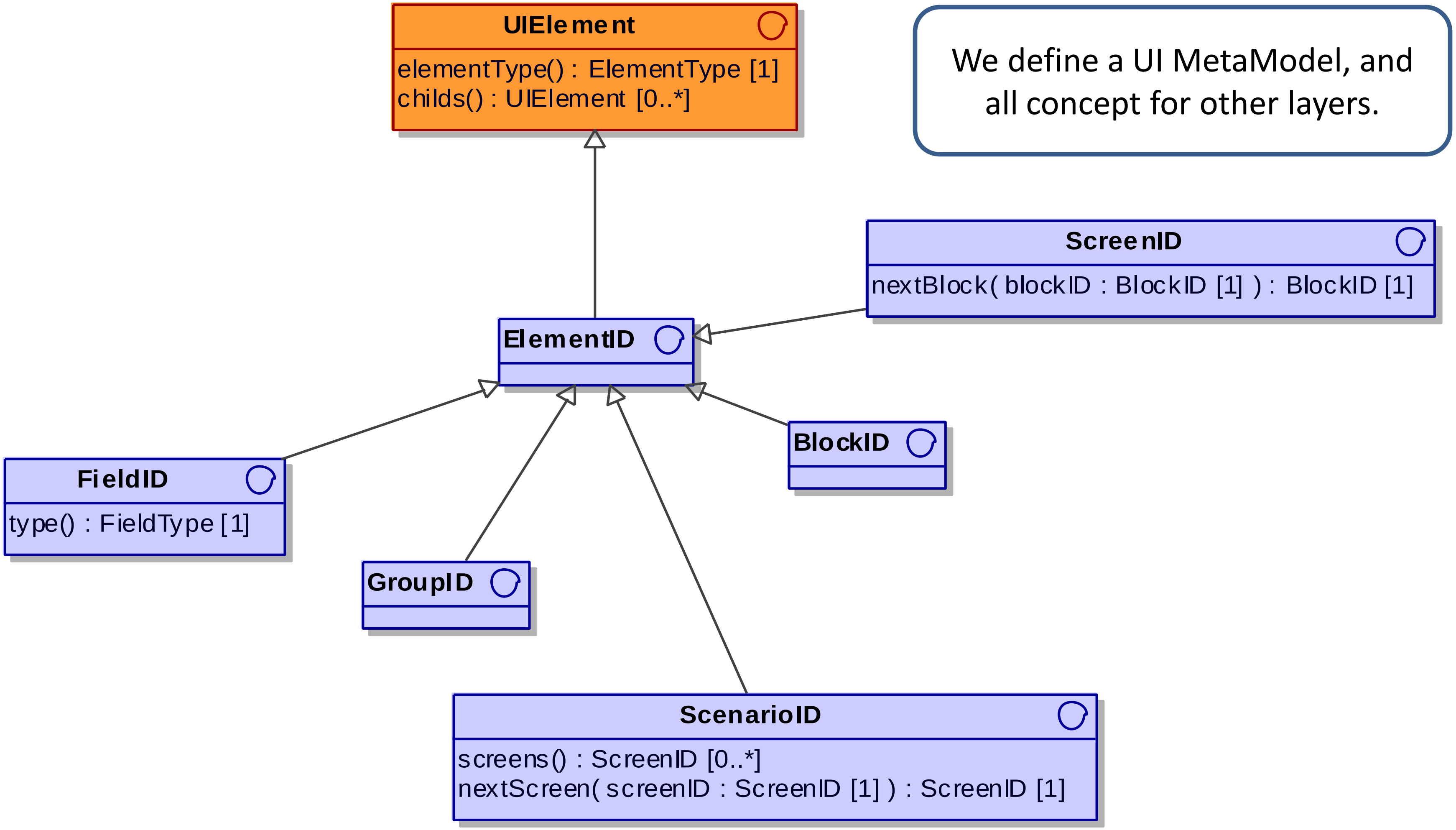
FieldView – GroupView – BlockView - ScreenView

View of the MVC pattern (runtime)

# FieldID – GroupID – BlockID – ScreenID - ScenarioID

# MetaModel

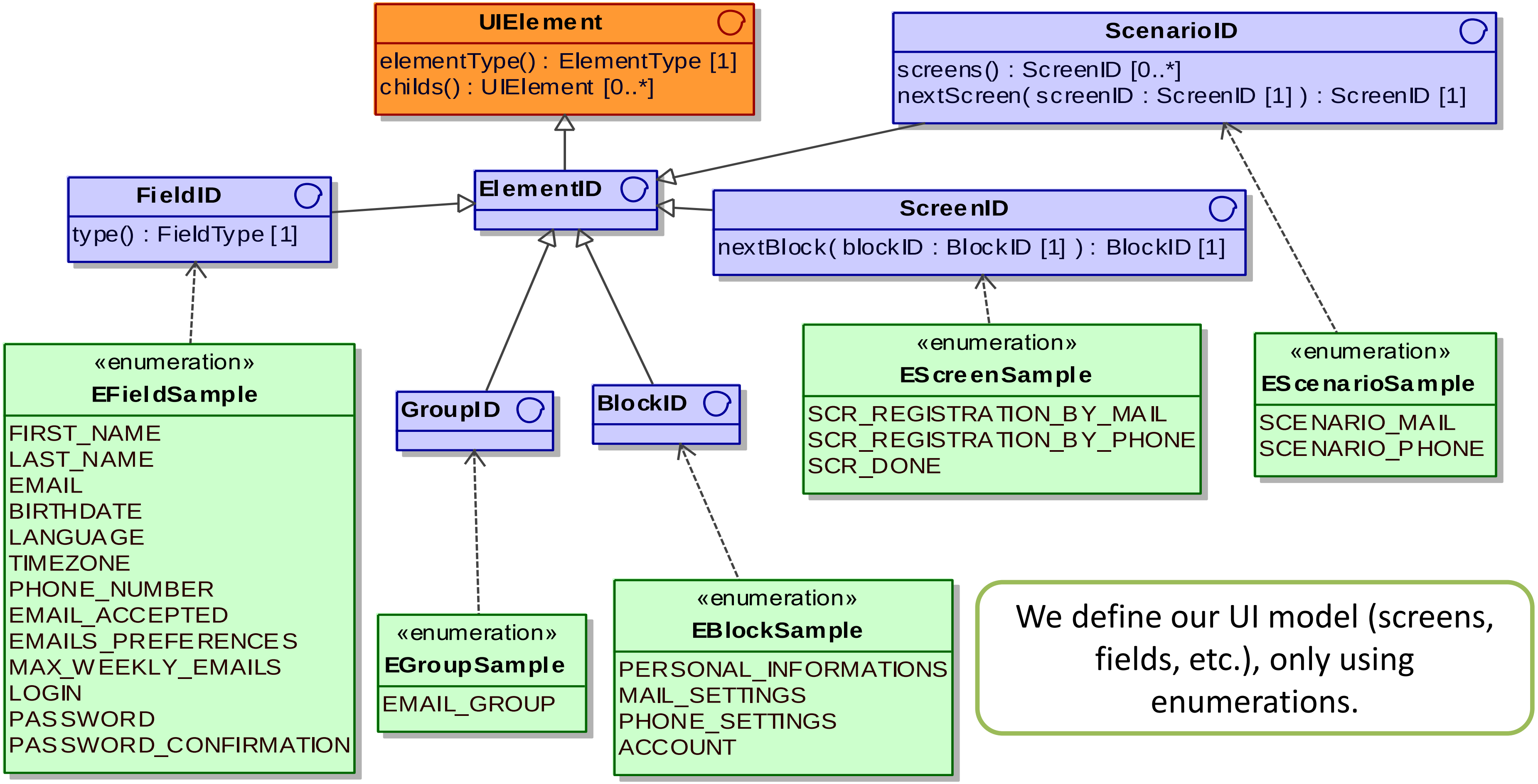
We define a UI MetaModel, and all concept for other layers.


UIElement
elementType() : ElementType [1]
childs() : UIElement [0..*]
ElementID
ScreenID
nextBlock( blockID : BlockID [1] ) : BlockID [1]
BlockID
FieldID
type() : FieldType [1]
GroupID
ScenarioID
screens() : ScreenID [0..*]
nextScreen( screenID : ScreenID [1] ) : ScreenID [1]

Customization layer

EFieldSample – EGroupSample – BlockSample - EScreenSample

Model



**UIElement**
elementType() : ElementType [1]
childs() : UIElement [0..*]

**ScenarioID**
screens() : ScreenID [0..*]
nextScreen( screenID : ScreenID [1] ) : ScreenID [1]

**FieldID**
type() : FieldType [1]

**ElementID**

**ScreenID**
nextBlock( blockID : BlockID [1] ) : BlockID [1]

**GroupID**

**BlockID**

«enumeration»
**EFieldSample**
FIRST_NAME
LAST_NAME
EMAIL
BIRTHDATE
LANGUAGE
TIMEZONE
PHONE_NUMBER
EMAIL_ACCEPTED
EMAILS_PREFERENCES
MAX_WEEKLY_EMAILS
LOGIN
PASSWORD
PASSWORD_CONFIRMATION

«enumeration»
**EGroupSample**
EMAIL_GROUP

«enumeration»
**EBlockSample**
PERSONAL_INFORMATIONS
MAIL_SETTINGS
PHONE_SETTINGS
ACCOUNT

«enumeration»
**EScreenSample**
SCR_REGISTRATION_BY_MAIL
SCR_REGISTRATION_BY_PHONE
SCR_DONE

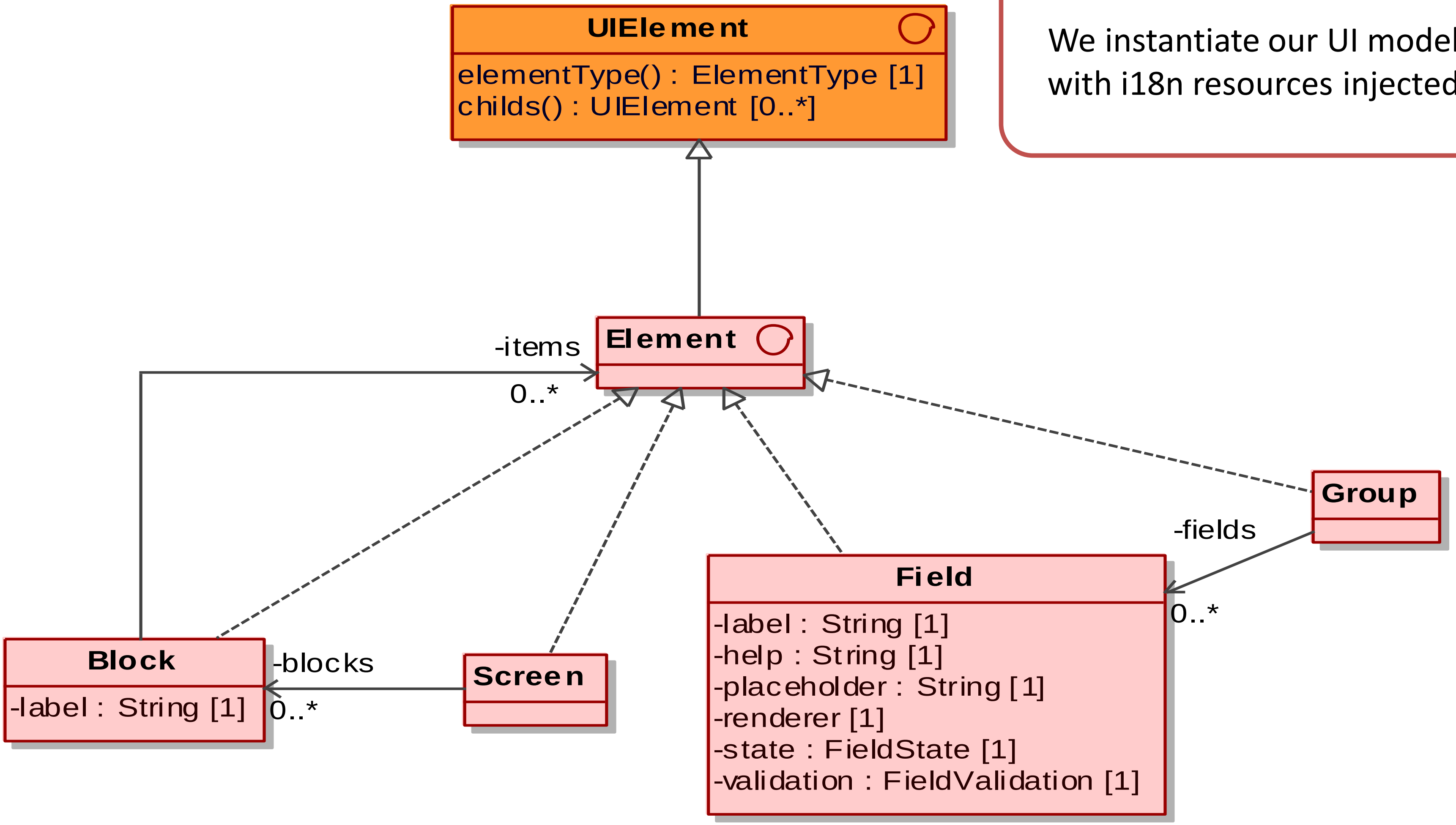
«enumeration»
**EScenarioSample**
SCENARIO_MAIL
SCENARIO_PHONE

We define our UI model (screens, fields, etc.), only using enumerations.

# Field – Group – Block - Screen

## Model Instance (runtime)

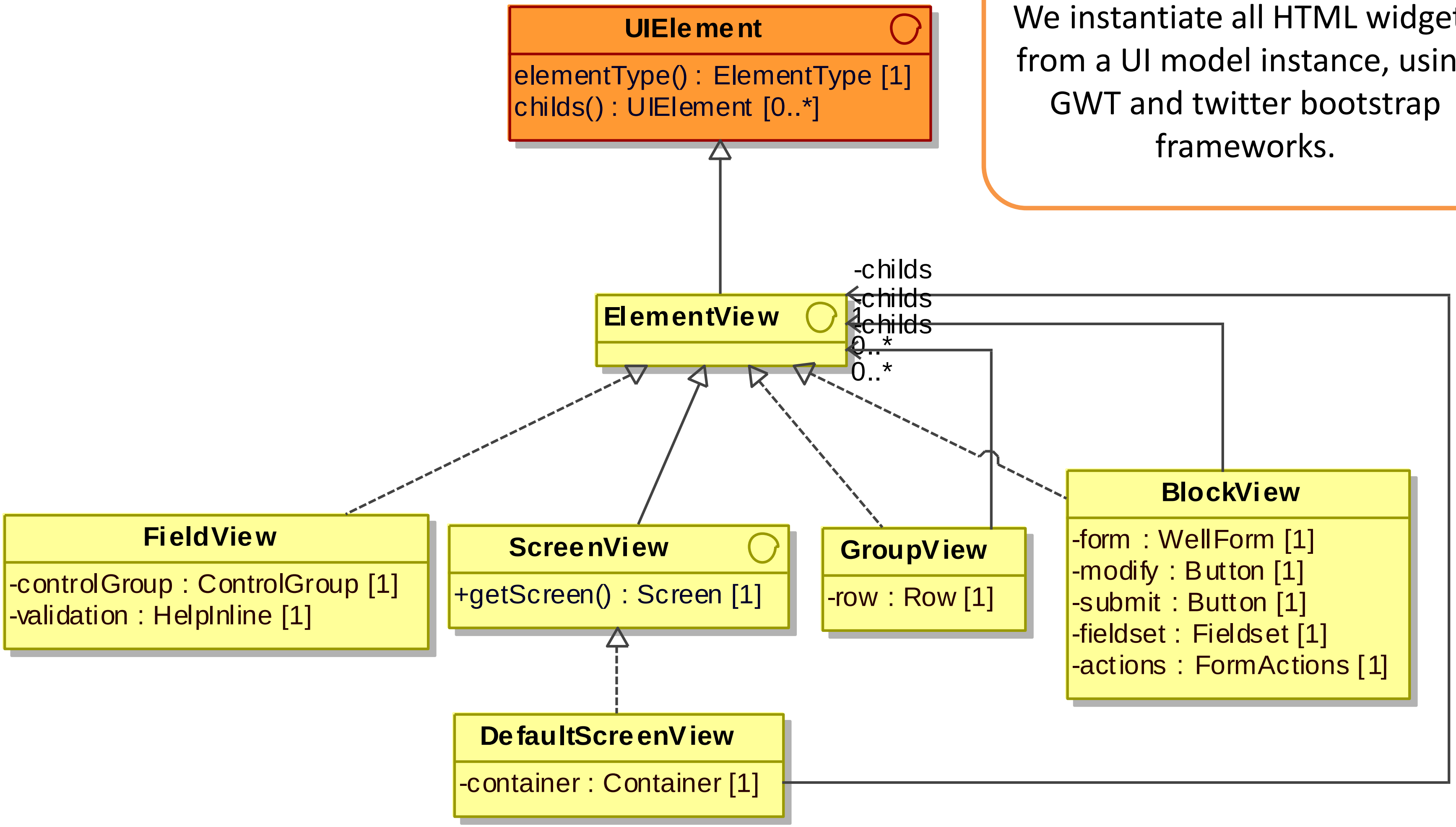
We instantiate our UI model, with i18n resources injected.



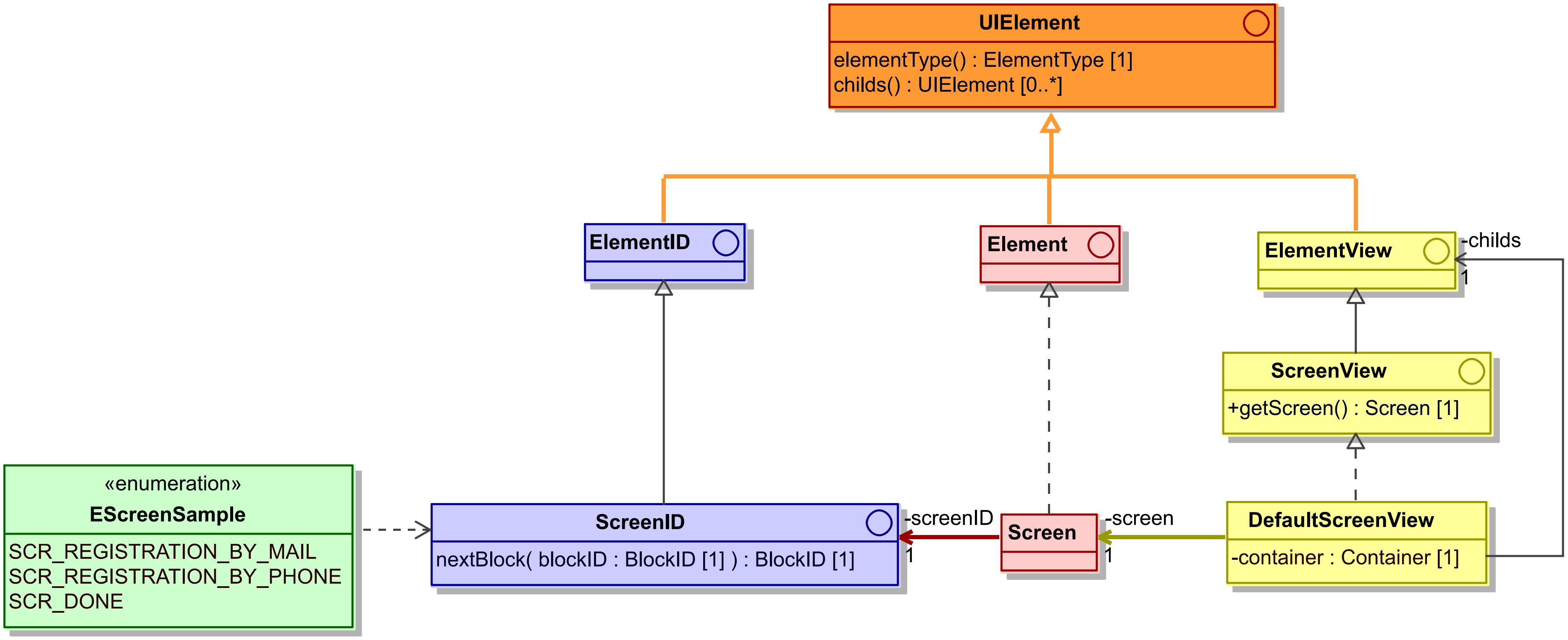
# FieldView – GroupView – BlockView - ScreenView

View of the MVC pattern (runtime)

We instantiate all HTML widgets from a UI model instance, using GWT and twitter bootstrap frameworks.



# From the point of view of a screen


UIElement
elementType() : ElementType [1]
childs() : UIElement [0..*]
ElementID
Element
ElementView
-childs
1
ScreenView
+getScreen() : Screen [1]
«enumeration»
EScreenSample
SCR_REGISTRATION_BY_MAIL
SCR_REGISTRATION_BY_PHONE
SCR_DONE
ScreenID
nextBlock( blockID : BlockID [1] ) : BlockID [1]
-screenID
1
Screen
-screen
1
DefaultScreenView
-container : Container [1]

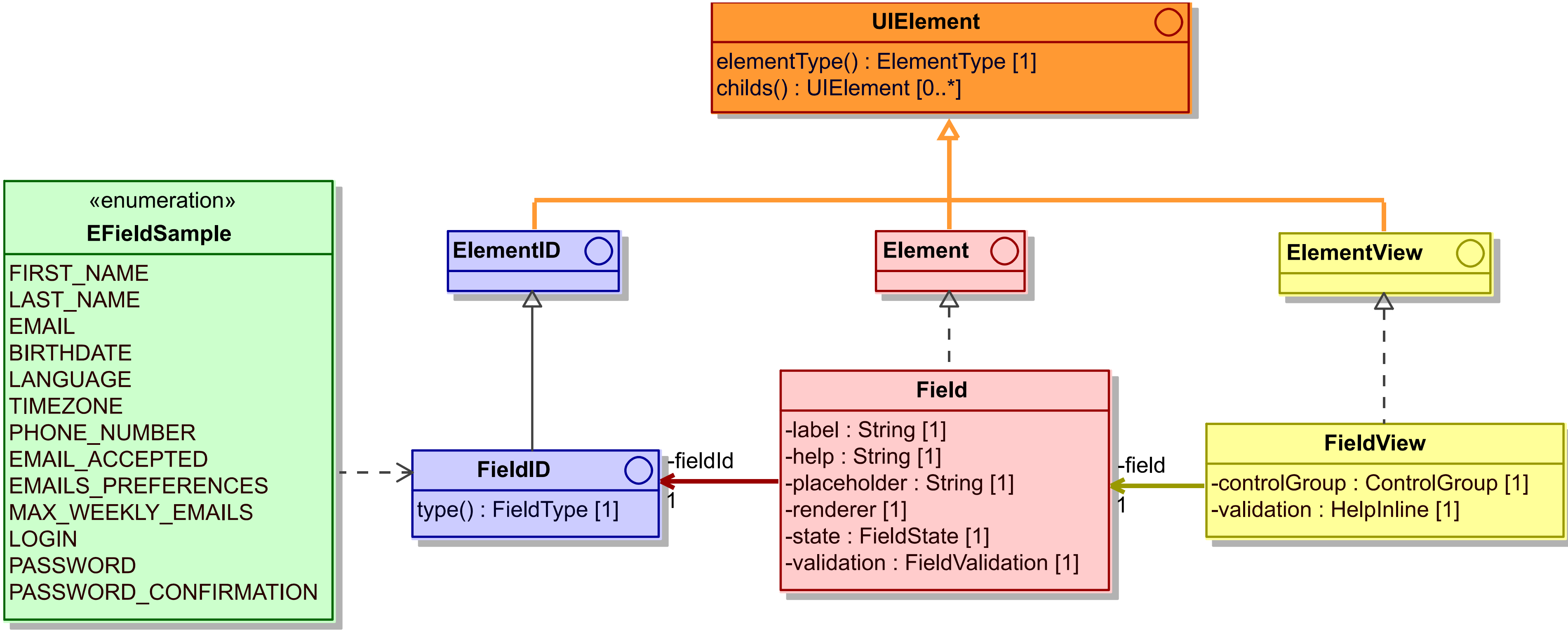
Model

MetaModel

Model Instance (runtime)

View of the MVC pattern (runtime)

# From the point of view of a field



«enumeration»
**EFieldSample**

FIRST_NAME
LAST_NAME
EMAIL
BIRTHDATE
LANGUAGE
TIMEZONE
PHONE_NUMBER
EMAIL_ACCEPTED
EMAILS_PREFERENCES
MAX_WEEKLY_EMAILS
LOGIN
PASSWORD
PASSWORD_CONFIRMATION

**UIElement**

elementType() : ElementType [1]
childs() : UIElement [0..*]

**ElementID**

**Element**

**ElementView**

**FieldID**

type() : FieldType [1]

-fieldId
1

**Field**

-label : String [1]
-help : String [1]
-placeholder : String [1]
-renderer [1]
-state : FieldState [1]
-validation : FieldValidation [1]

-field
1

**FieldView**

-controlGroup : ControlGroup [1]
-validation : HelpInline [1]

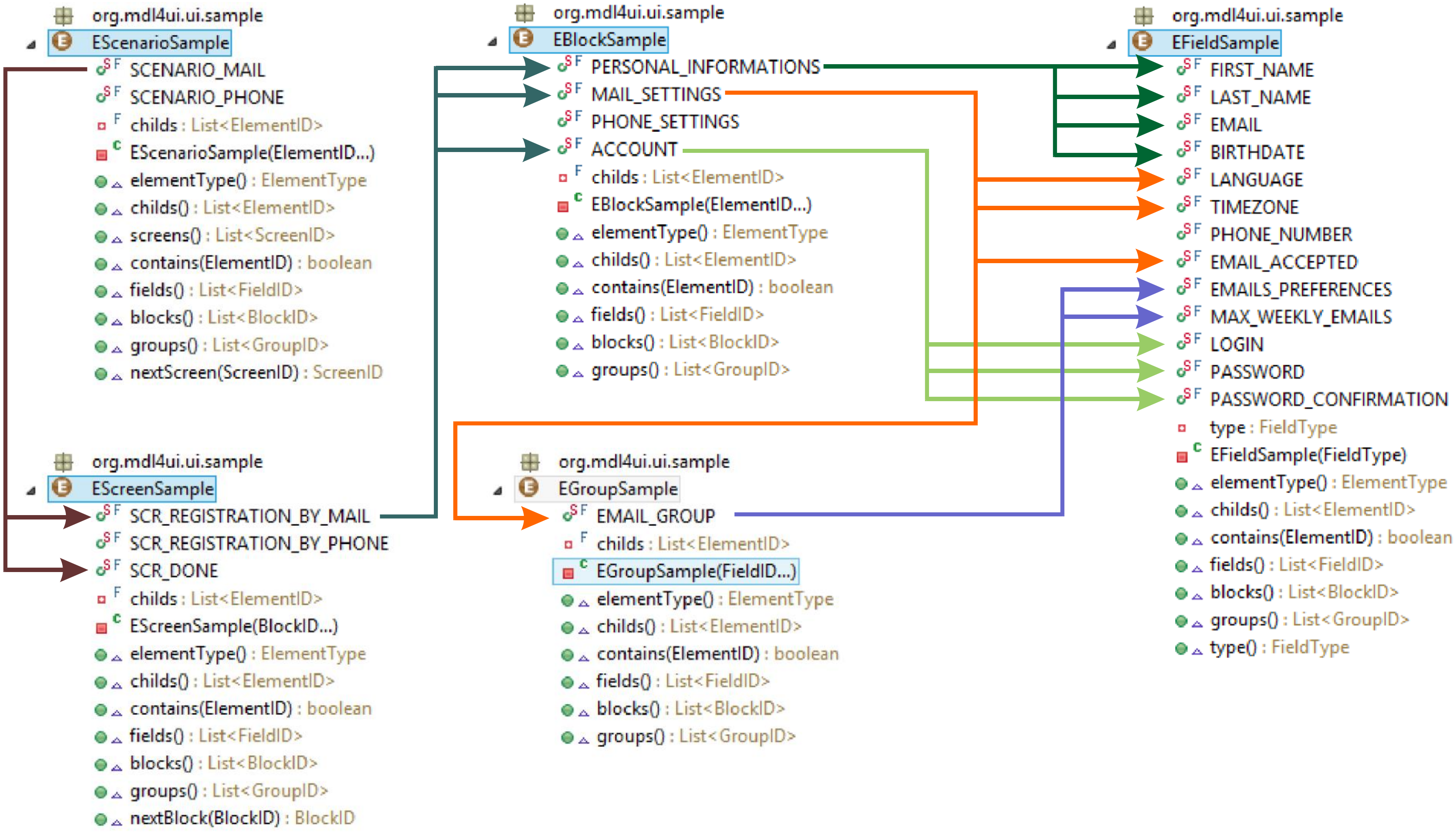
Model

MetaModel

Model Instance (runtime)

View of the MVC pattern (runtime)

# Implementing the model



org.mdl4ui.ui.sample
- EScenarioSample
  - SCENARIO_MAIL
  - SCENARIO_PHONE
  - childs : List<ElementID>
  - EScenarioSample(ElementID...)
  - elementType() : ElementType
  - childs() : List<ElementID>
  - screens() : List<ScreenID>
  - contains(ElementID) : boolean
  - fields() : List<FieldID>
  - blocks() : List<BlockID>
  - groups() : List<GroupID>
  - nextScreen(ScreenID) : ScreenID

org.mdl4ui.ui.sample
- EScreenSample
  - SCR_REGISTRATION_BY_MAIL
  - SCR_REGISTRATION_BY_PHONE
  - SCR_DONE
  - childs : List<ElementID>
  - EScreenSample(BlockID...)
  - elementType() : ElementType
  - childs() : List<ElementID>
  - contains(ElementID) : boolean
  - fields() : List<FieldID>
  - blocks() : List<BlockID>
  - groups() : List<GroupID>
  - nextBlock(BlockID) : BlockID

org.mdl4ui.ui.sample
- EBlockSample
  - PERSONAL_INFORMATIONS
  - MAIL_SETTINGS
  - PHONE_SETTINGS
  - ACCOUNT
  - childs : List<ElementID>
  - EBlockSample(ElementID...)
  - elementType() : ElementType
  - childs() : List<ElementID>
  - contains(ElementID) : boolean
  - fields() : List<FieldID>
  - blocks() : List<BlockID>
  - groups() : List<GroupID>

org.mdl4ui.ui.sample
- EGroupSample
  - EMAIL_GROUP
  - childs : List<ElementID>
  - EGroupSample(FieldID...)
  - elementType() : ElementType
  - childs() : List<ElementID>
  - contains(ElementID) : boolean
  - fields() : List<FieldID>
  - blocks() : List<BlockID>
  - groups() : List<GroupID>

org.mdl4ui.ui.sample
- EFieldSample
  - FIRST_NAME
  - LAST_NAME
  - EMAIL
  - BIRTHDATE
  - LANGUAGE
  - TIMEZONE
  - PHONE_NUMBER
  - EMAIL_ACCEPTED
  - EMAILS_PREFERENCES
  - MAX_WEEKLY_EMAILS
  - LOGIN
  - PASSWORD
  - PASSWORD_CONFIRMATION
  - type : FieldType
  - EFieldSample(FieldType)
  - elementType() : ElementType
  - childs() : List<ElementID>
  - contains(ElementID) : boolean
  - fields() : List<FieldID>
  - blocks() : List<BlockID>
  - groups() : List<GroupID>
  - type() : FieldType

# Why modeling as code ?

- Sorry we are Java developers
- Built-in **continuous integration** for the model
- **No code generation** required to implement the model
- **Modeling concept** understanding is **not required**
- **Modeling stack** is **transparent** for UI development
- **Tooling** is very fast
- **Memory footprint** is very low
- A lot of **consistency** checking is done by the compiler
- More benefits to come in the next slides …

Context

Quick demo

Modeling approach

**Dependency Model**

Field features

Extensions

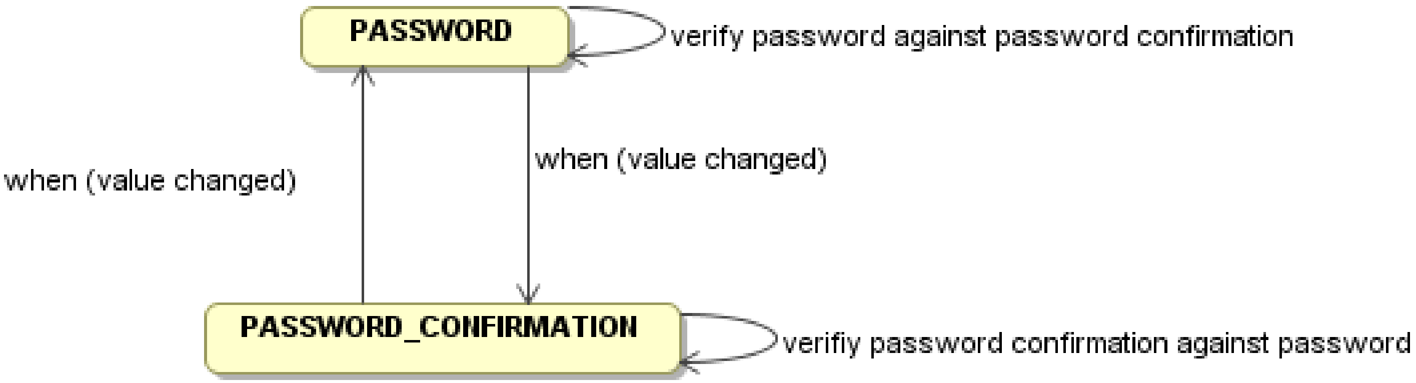
Live coding demo

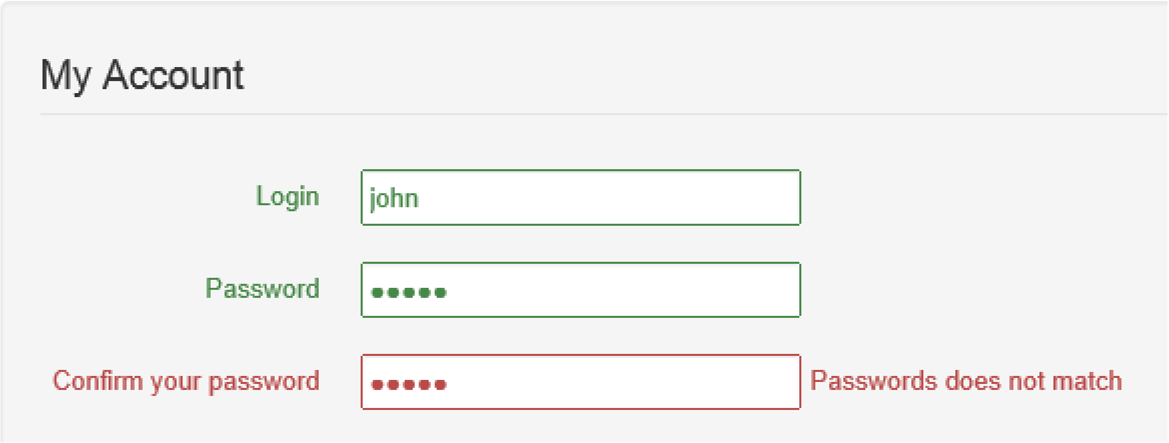
Back to LesFurets.com

# We need a dependency graph

- Implementing business rules involves **triggering the behaviors** using a dependency model
- **No semantics** on the dependency
- Fields receive **dependency events** with **source attribute**
- Each **field implements various features** to react to dependency events
    - Visibility of the fields
    - Value range definition
    - Reset of value
    - Validation of value

# Validation dependency


PASSWORD
verify password against password confirmation
when (value changed)
when (value changed)
PASSWORD_CONFIRMATION
verifiy password confirmation against password



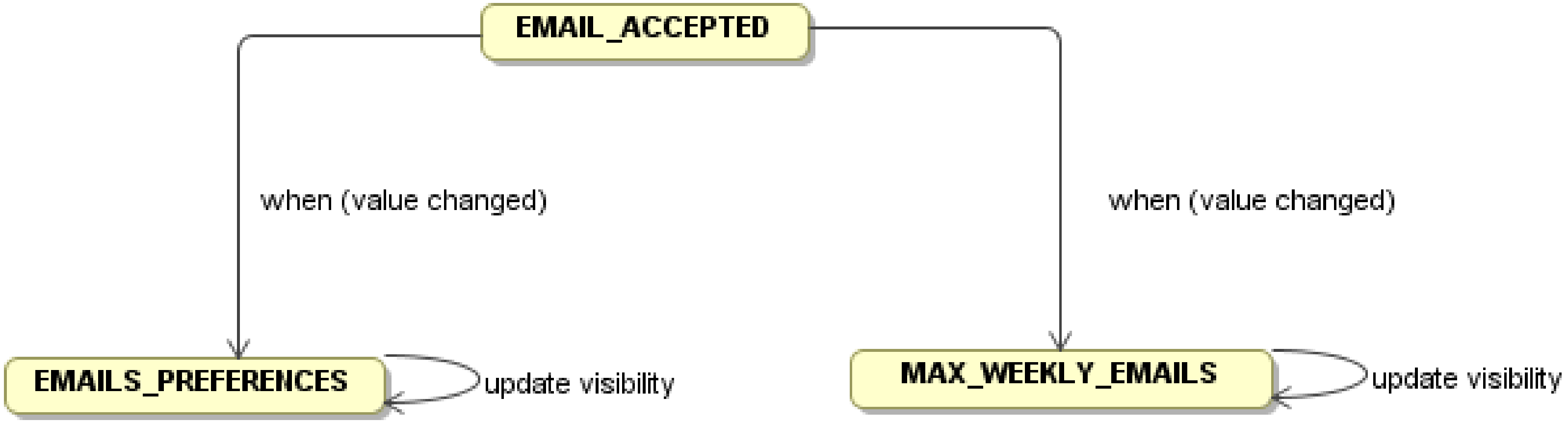
My Account

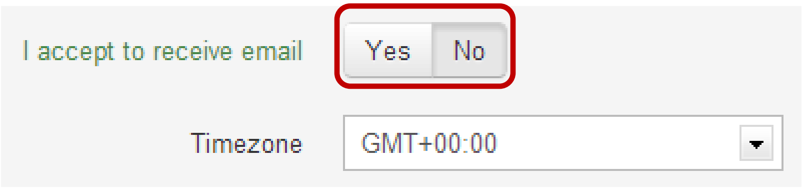
Login: john

Password: ●●●●●

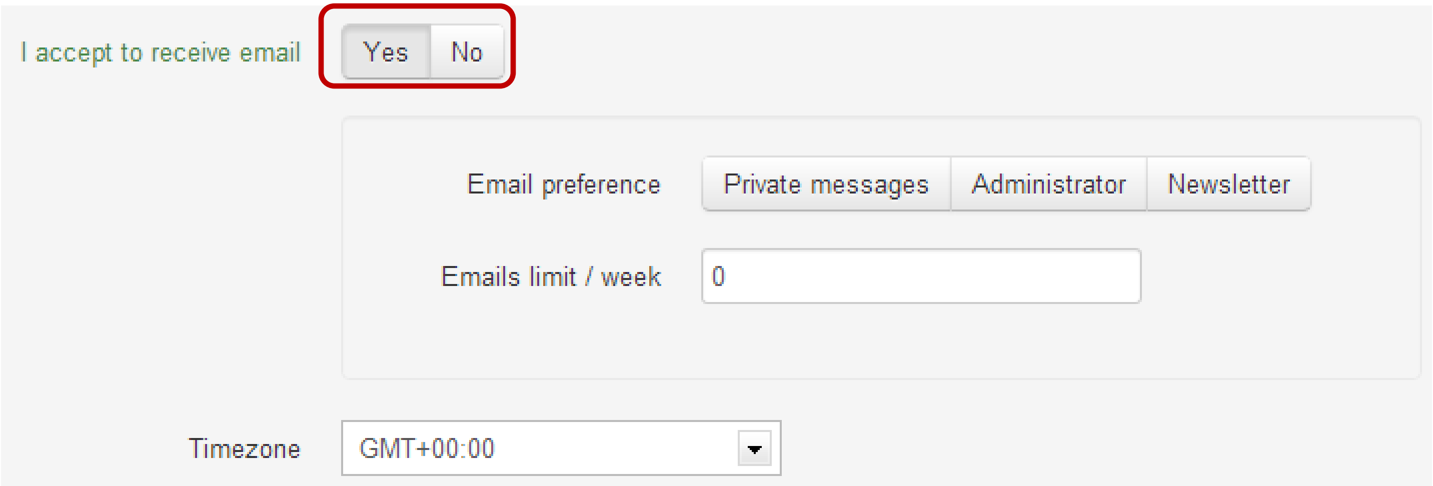
Confirm your password: ●●●●● Passwords does not match

# Visibility dependency



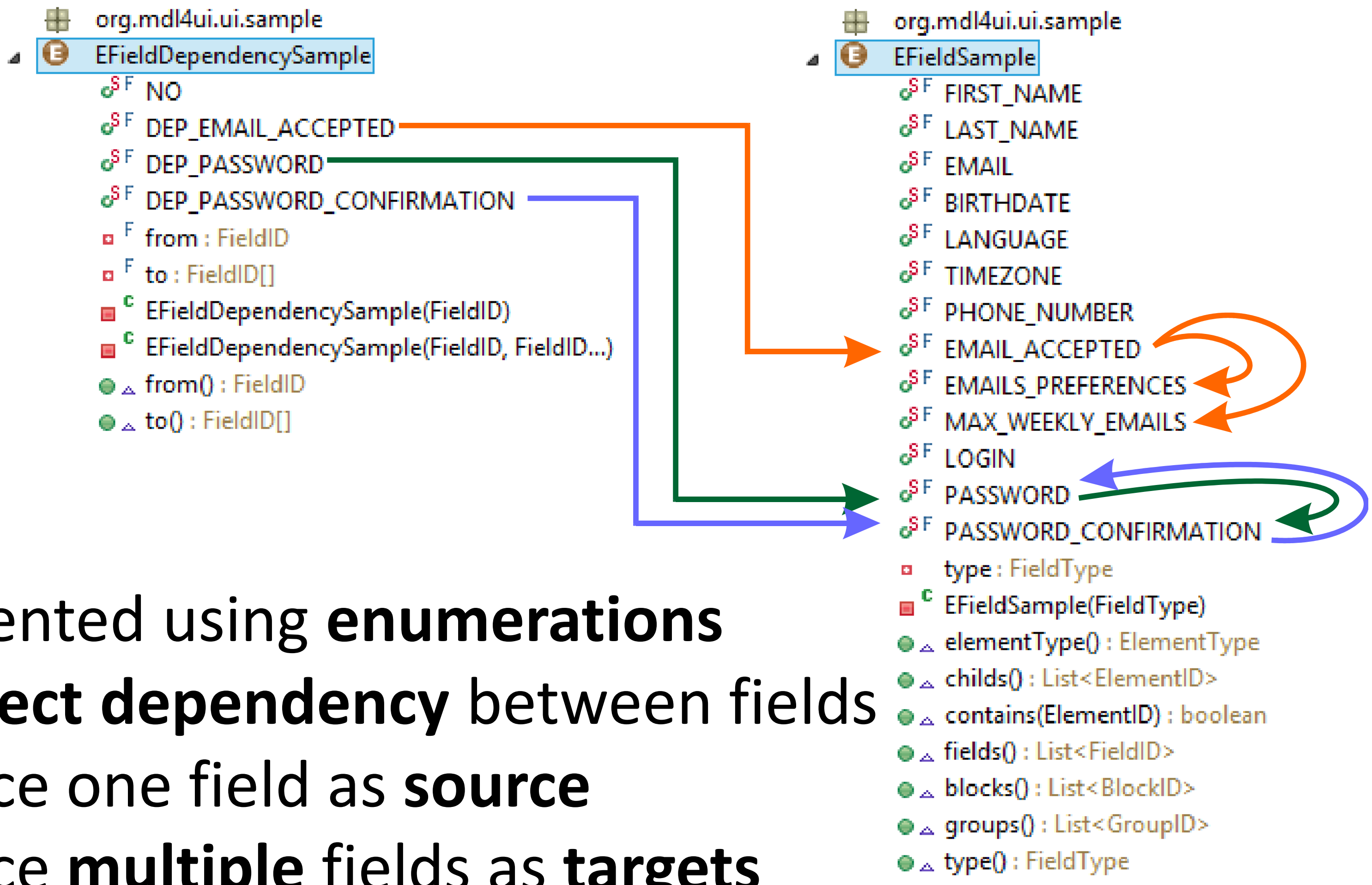


| | |
|---|---|
| I accept to receive email | Yes [✓ No] |
| Timezone | GMT+00:00 |



| | |
|---|---|
| I accept to receive email | [✓ Yes] No |
| Email preference | Private messages / Administrator / Newsletter |
| Emails limit / week | 0 |
| Timezone | GMT+00:00 |

# In code declarative dependencies modeling



org.mdl4ui.ui.sample
EFieldDependencySample
NO
DEP_EMAIL_ACCEPTED
DEP_PASSWORD
DEP_PASSWORD_CONFIRMATION
from : FieldID
to : FieldID[]
EFieldDependencySample(FieldID)
EFieldDependencySample(FieldID, FieldID...)
from() : FieldID
to() : FieldID[]

org.mdl4ui.ui.sample
EFieldSample
FIRST_NAME
LAST_NAME
EMAIL
BIRTHDATE
LANGUAGE
TIMEZONE
PHONE_NUMBER
EMAIL_ACCEPTED
EMAILS_PREFERENCES
MAX_WEEKLY_EMAILS
LOGIN
PASSWORD
PASSWORD_CONFIRMATION
type : FieldType
EFieldSample(FieldType)
elementType() : ElementType
childs() : List<ElementID>
contains(ElementID) : boolean
fields() : List<FieldID>
blocks() : List<BlockID>
groups() : List<GroupID>
type() : FieldType

- Implemented using **enumerations**
- **Only direct dependency** between fields
- Reference one field as **source**
- Reference **multiple** fields as **targets**

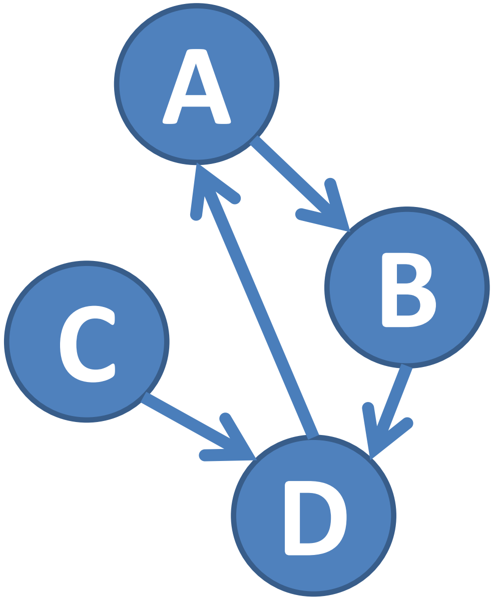
# Dependencies processing

Declared dependencies

A —> B
B —> D
D —> A
C —> D

Hand written code

Dependencies graph



Underlying model

Deep dependencies resolved

A —> B,D
B —> D,A
D —> A,B
C —> D,A,B

Generated code

# Deep dependency, dependency cycle, graph validation

- **Cycle declaration** between fields is **allowed**
- Deep dependencies are statically resolved
  - For each field the **deep dependencies** are **generated** during the compilation
  - Model declared in **EFieldDependency[Sample]**
  - Deep dependency  are generated in **EFieldDeepDependency[Sample]**
  - Dependency **order** is **not guaranteed**
- **No runtime infinite loop**

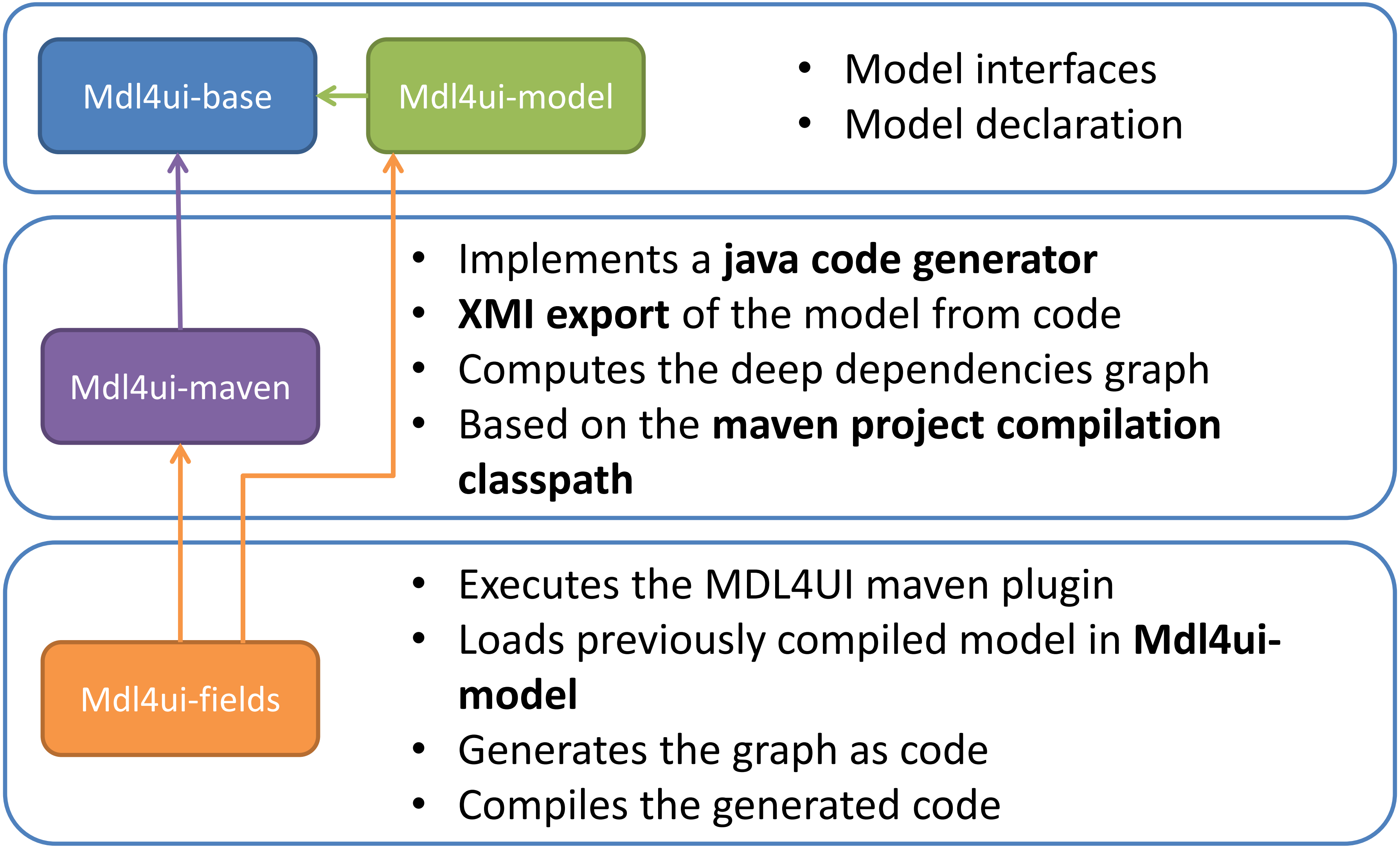
# Simple dependency API

```
public interface FieldDependencyFactory {

   FieldID[] get(FieldID fieldId);
}
```

- Implementation is **generated** by our maven plugin
- Graph is **built from** the **FieldDependency** declaration
- Deep dependencies are **statically resolved** for each field
  - Look at the implementation EFieldDeepDependency[Sample]
- **No runtime computation** of the dependencies
- **Safe** and **efficient**

# Code generation using a maven custom plugin



- Model interfaces
- Model declaration

- Implements a **java code generator**
- **XMI export** of the model from code
- Computes the deep dependencies graph
- Based on the **maven project compilation classpath**

- Executes the MDL4UI maven plugin
- Loads previously compiled model in **Mdl4ui-model**
- Generates the graph as code
- Compiles the generated code

# Walkthrough the model in a maven plugin

```
void lookOver(ElementID parentId) {
   for (ElementID childId : parentId.childs()) {
      if (childId.elementType() == GROUP ||
          childId.elementType() == BLOCK)
         lookOver(childId);
      else
         System.out.println("field :" + childId);
   }
}
```

- Simple tree API to explore the structure
- Easy use of recursive algorithms

# Maven plugin declaration

```
<plugin>
  <groupId>org.mdl4ui</groupId>
  <artifactId>mdl4ui-maven</artifactId>
  <executions>
    <execution>
      <id>generate-model</id>
      <phase>process-classes</phase>
      <goals>
        <goal>generateModel</goal>
      </goals>
      <configuration>
        <screenClasses>
          <screenClasse>org.mdl4ui.ui.sample.EScreenSample</screenClasse>
        </screenClasses>
      </configuration>
    </execution>
    ….
```

- Model instance is available in the maven project **classpath** through the **maven dependencies**
- We **load** the model from the **screens elements**

Context

Quick demo

Modeling approach

Dependency Model

**Field features**

Extensions

Live coding demo

Back to LesFurets.com

# Goal and inspiration

- UI logic is often synonym of **spaghetti code**
- **Decoupling UI and logic** is often difficult to implement
- **Slicing the logic** in tiny pieces of code is the key for :
  - Testability
  - Governance
- Inspiration
  - MVC (client side)
  - JavaBean
  - BeanValidation
  - Injection, CDI, Guice, Dagger
- **Browser** runtime using JavaScript is a heavy **constraint**
  - Inversion of control is difficult to implement

# Features provided by fields

- FieldInitializer
  - Initialize **default value** and **range**
- FieldEditor
  - MVC pattern to **sync the model** during form completion
  - **Validation** during form completion
  - Reset after **visibility changes**
- FieldBehaviour
  - **Visibility** update
  - **Dependency** update
- Labeling
  - **Attached** widget **labels**, **help** messages, **place holders**

# FieldInitializer API

```
public interface FieldInitializer {

   void init(Field field,
                FieldEvent event);
}
```

- Initialize the field during the **bootstrap** of the application

# FieldEditor API

```
public interface FieldEditor {

   void updateFromContext(Field field,  WizardContext context,
                                                          FieldEvent fieldEvent);

   void updateContext(Field field, WizardContext context,
                                              FieldEvent fieldEvent);

   void reset(Field field, WizardContext context,
                         FieldEvent fieldEvent);

   FieldValidation validate(Field field, WizardContext context,
                                                      FieldEvent fieldEvent);
}
```

- WizardContext is the **entry point** of the domain model for the MVC pattern
- **updateFromContext** and **updateContext** **read and update** the domain model of the MVC pattern
- **reset** is called after a **field** is **hidden** or a **value change** from a **dependency**

# FieldBehaviour API

```
public interface FieldBehaviour {

   boolean isVisible(FieldID fieldId,
                     WizardContext context,
                     FieldEvent fieldEvent);

   void updateValue(Field field,
                     WizardContext context,
                     FieldEvent event);
}
```

- **isVisible** returns the visibility **following** the value of the domain **model**
- **updateValue** is **triggered** by the **dependency management**

# Declaring a feature of a field

```
@InjectSampleBehaviour(
      @OnField({ EFieldSample.EMAILS_PREFERENCES,
                         EFieldSample.MAX_WEEKLY_EMAILS }))
public class AcceptEmailsBehaviour extends DefaultBehaviour {

   @Override
   public boolean isVisible(FieldID fieldId, WizardContext context,
                                          FieldEvent fieldEvent) {
      SampleContext sampleContext = (SampleContext) context;
      Boolean acceptEmail = sampleContext.getUserAccount().isAcceptEmail();
      return acceptEmail != null && acceptEmail;
   }
}
```

# Injecting the field features with annotations and meta annotation

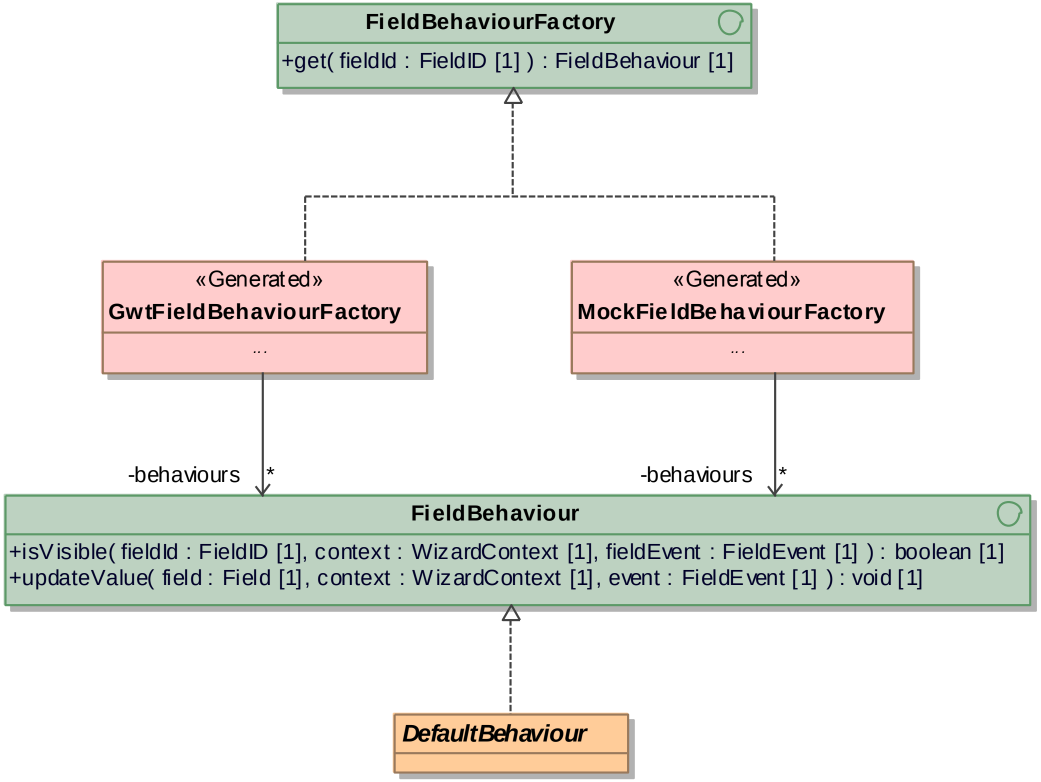
| Meta annotation | Custom annotation | Injected resource |
|---|---|---|
| **@InjectInit** | **@InjectSampleInit**<br>Reference one or more EFieldSample | Any class implementing **FieldInitializer** |
| **@InjectEditor** | **@InjectSampleEditor**<br>Reference one or more EFieldSample | Any class implementing **FieldEditor** |
| **@InjectBehaviour** | **@InjectSampleBehaviour**<br>Reference one or more EFieldSample | Any class implementing **FieldBehaviour** |
| **@InjectLabel** | **@InjectSampleLabel**<br>Reference one or more EField, EGroup, EBlock and EScreen[Sample] | Any **interface method** without parameter returning a String |
| **@InjectHelp** | **@InjectSampleHelp**<br>Reference one or more EField, EGroup, EBlock and EScreen[Sample] | Any **interface method** without parameter returning a String |
| **@InjectPlaceHolder** | **@InjectSamplePlaceHolder**<br>Reference one or more EField, EGroup, EBlock and EScreen[Sample] | Any **interface method** without parameter returning a String |

# The plumbing using APT

We use **Annotation Processing Tool** to bind together the various field features and the fields

- APT is a **standard tooling** packaged with the JDK since Java 6
- Allows to **generate source code** and resources in the source path of the compiler during the early stage of the **compilation process**
- Source code processing based on **javax.lang.model** API
- Code processing is **triggered by annotation**
- No built-in code generator
  - Use basic template mechanism to simplify source code generation

# Generated pattern to glue things together



**FieldBehaviourFactory**
+get( fieldId : FieldID [1] ) : FieldBehaviour [1]

«Generated»
**GwtFieldBehaviourFactory**
...

«Generated»
**MockFieldBehaviourFactory**
...

-behaviours *

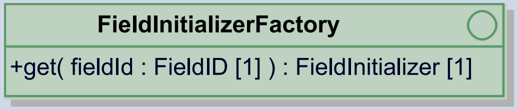
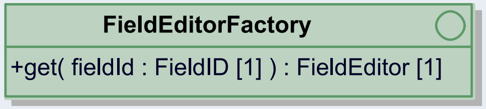
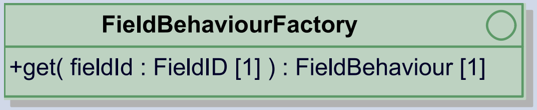
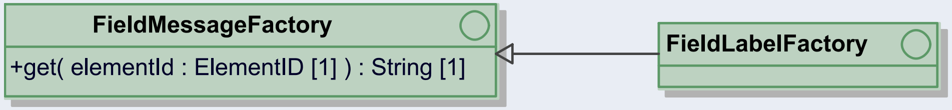
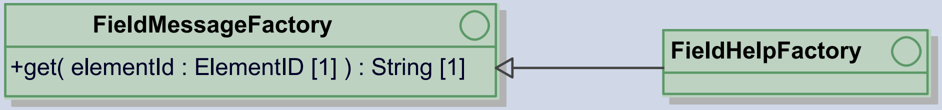
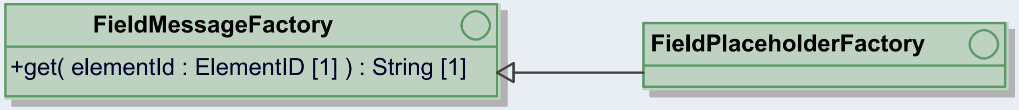
-behaviours *

**FieldBehaviour**
+isVisible( fieldId : FieldID [1], context : WizardContext [1], fieldEvent : FieldEvent [1] ) : boolean [1]
+updateValue( field : Field [1], context : WizardContext [1], event : FieldEvent [1] ) : void [1]

***DefaultBehaviour***

- Code generation is triggered by **@InjectBehaviour**
- **APT processor** is executed during the compilation of **Mdl4ui-field** project
- We perform some **validations**, like detecting duplicated injections
- We use a **factory pattern** returning the right instance for each field
- An implementation for **GWT client runtime** purpose
- A **mock implementation GWT less** for unit testing purpose

# Replicate the factory pattern for each feald feature

| Meta annotation | Feature factory |
|---|---|
| **@InjectInit** |  **FieldInitializerFactory**<br>+get( fieldId : FieldID [1] ) : FieldInitializer [1] |
| **@InjectEditor** |  **FieldEditorFactory**<br>+get( fieldId : FieldID [1] ) : FieldEditor [1] |
| **@InjectBehaviour** |  **FieldBehaviourFactory**<br>+get( fieldId : FieldID [1] ) : FieldBehaviour [1] |
| **@InjectLabel** |  **FieldMessageFactory**<br>+get( elementId : ElementID [1] ) : String [1]<br>**FieldLabelFactory** |
| **@InjectHelp** |  **FieldMessageFactory**<br>+get( elementId : ElementID [1] ) : String [1]<br>**FieldHelpFactory** |
| **@InjectPlaceHolder** |  **FieldMessageFactory**<br>+get( elementId : ElementID [1] ) : String [1]<br>**FieldPlaceholderFactory** |

Context

Quick demo

Modeling approach

Dependency Model

Field features

**Extensions**

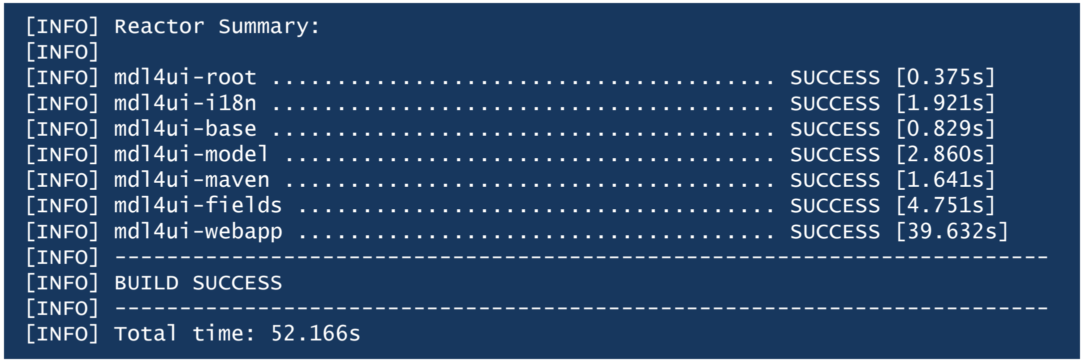
Live coding demo

Back to LesFurets.com

# Content of MDL4UI

- **Mdl4ui-I18n** : foundation framework for text resource injection, containing APT processors and annotations
- **Mdl4ui-base**: foundation framework for the UI model interfaces, containing APT processors and annotations
- **Mdl4ui-model**: model instance for our code sample, including fields and dependencies
- **Mdl4ui-maven**: maven plugin part of the foundation framework that generate and check the dependency graph between the fields, export the model in XMI
- **Mdl4ui-fields**: business rules, validation and field editors (MVC pattern) for our sample

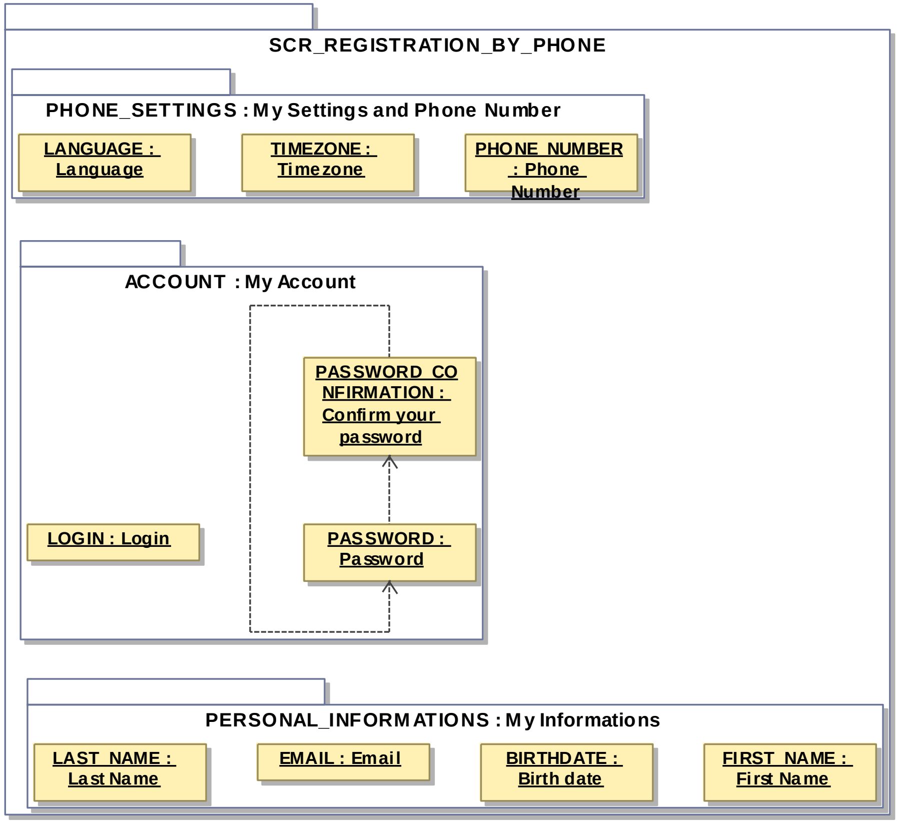
- **Mdl4ui-webapp**: the web application that assembles the code, compiles various resources with GWT and adds styling



```
[INFO] Reactor Summary:
[INFO]
[INFO] mdl4ui-root ....................................... SUCCESS [0.375s]
[INFO] mdl4ui-i18n ....................................... SUCCESS [1.921s]
[INFO] mdl4ui-base ....................................... SUCCESS [0.829s]
[INFO] mdl4ui-model ...................................... SUCCESS [2.860s]
[INFO] mdl4ui-maven ...................................... SUCCESS [1.641s]
[INFO] mdl4ui-fields ..................................... SUCCESS [4.751s]
[INFO] mdl4ui-webapp ..................................... SUCCESS [39.632s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 52.166s
```

# Generate UML to understand the model

- Use to **document** the model and specify **evolution**
- **Visualize** the dependency graph
- Artifact Generated during the **continuous integration**



SCR_REGISTRATION_BY_PHONE

PHONE_SETTINGS : My Settings and Phone Number

LANGUAGE : Language

TIMEZONE : Timezone

PHONE_NUMBER : Phone Number

ACCOUNT : My Account

PASSWORD_CONFIRMATION : Confirm your password

LOGIN : Login

PASSWORD : Password

PERSONAL_INFORMATIONS : My Informations

LAST_NAME : Last Name

EMAIL : Email

BIRTHDATE : Birth date

FIRST_NAME : First Name

| MDL4UI | UML |
|---|---|
| ScreenID | Package |
| BlockID | Package |
| GroupID | Package |
| FieldID | Instance specification |
| FieldLabel | Class |
| FieldDependency | Dependency |

# Field tracking

- Use field features to track
  - Field inputs
  - Field validation errors
  - Screen and block navigation

- Use of tracking results
  - Find common user profiles
  - Improve forms for faster input
  - Find ergonomic issues

# AB testing and shuffling the fields

- Define two versions of a webpage (A and B)
- Split traffic amongst those versions
- Determine which one was more successful
  - Validate any new design
  - Improve the conversion rate

- How it can be done?
  - Define new fields and new FieldBehaviour
  - Define two different scenarios

# Unit testing

- Need to **test fields** using **regression** tests:
    - validation rules
    - field visibility update
    - domain model read & update
    - domain model reset

- Generated **mock factories** allow to execute features implementation **without** a web application **container** (GWT)

# Unit testing

```
@Test
public void dependencies() {
  FieldDependencyFactory dependencyFactory
        = new FieldDependencySampleFactory();

  Collection<FieldID> dependencies =
    Arrays.asList(dependencyFactory.get(EMAIL_ACCEPTED));

  assertEquals(2, dependencies.size());
  assertTrue(dependencies.contains(EMAILS_PREFERENCES));
  assertTrue(dependencies.contains(MAX_WEEKLY_EMAILS));
}
```

# Unit testing

```
@Test
public void visibility() {
    FieldDependencyFactory dependencyFactory = new FieldDependencySampleFactory();
    MockFieldBehaviourFactory behaviourFactory = new MockFieldBehaviourFactory();

    SampleContext context = new SampleContext();

    for (FieldID dependency : dependencyFactory.get(EMAIL_ACCEPTED)) {
        FieldBehaviour behaviour = behaviourFactory.get(dependency);

        context.getUserAccount().setAcceptEmail(false);
        assertFalse(behaviour.isVisible(dependency, context, null));

        context.getUserAccount().setAcceptEmail(true);
        assertTrue(behaviour.isVisible(dependency, context, null));
    }
}
```

# Selenium and integration testing

- Selenium is a **test automation** framework for **web applications**
    - sends commands to a browser
    - retrieves results (parsing the DOM)

- Supports:
    - **Java**, Ruby, Python, C#, etc.
    - **Firefox**, Chrome, IE, iOS & Android browsers, etc.

# Selenium and integration testing

- Generation of page object classes
  - **representing a screen or a block** with selenium framework
  - exposing methods to **manipulate each fields**

- Make testing **easier**
  - **hide** selenium framework complexity
  - **minimize** the test maintenance effort

# Selenium and integration testing

```
@Test
  public void testRegistration() {
    RegistrationByMailScreen registrationScreen
                              = new RegistrationByMailScreen(getDriver());
    registrationScreen.assertDisplayed();
    registrationScreen.getPersonalInformations()//
                .assertDisplayed()//
                .setFirstName("John")//
                .setLastName("Doe")//
                .setBirthdate(new DateMidnight(1980, 1, 1))//
                .setEmail("john@doe.com")//
                .submit();
    registrationScreen.getMailSettings().assertDisplayed;
  }
```

«enumeration»
**EScreenSample**

SCR_REGISTRATION_BY_MAIL
SCR_REGISTRATION_BY_PHONE
SCR_DONE

«enumeration»
**EBlockSample**

PERSONAL_INFORMATIONS
MAIL_SETTINGS
PHONE_SETTINGS
ACCOUNT

Context

Quick demo

Modeling approach

Dependency Model

Field features

Extensions

**Live coding demo**

Back to LesFurets.com

Context

Quick demo

Modeling approach

Dependency Model

Field features

Extensions

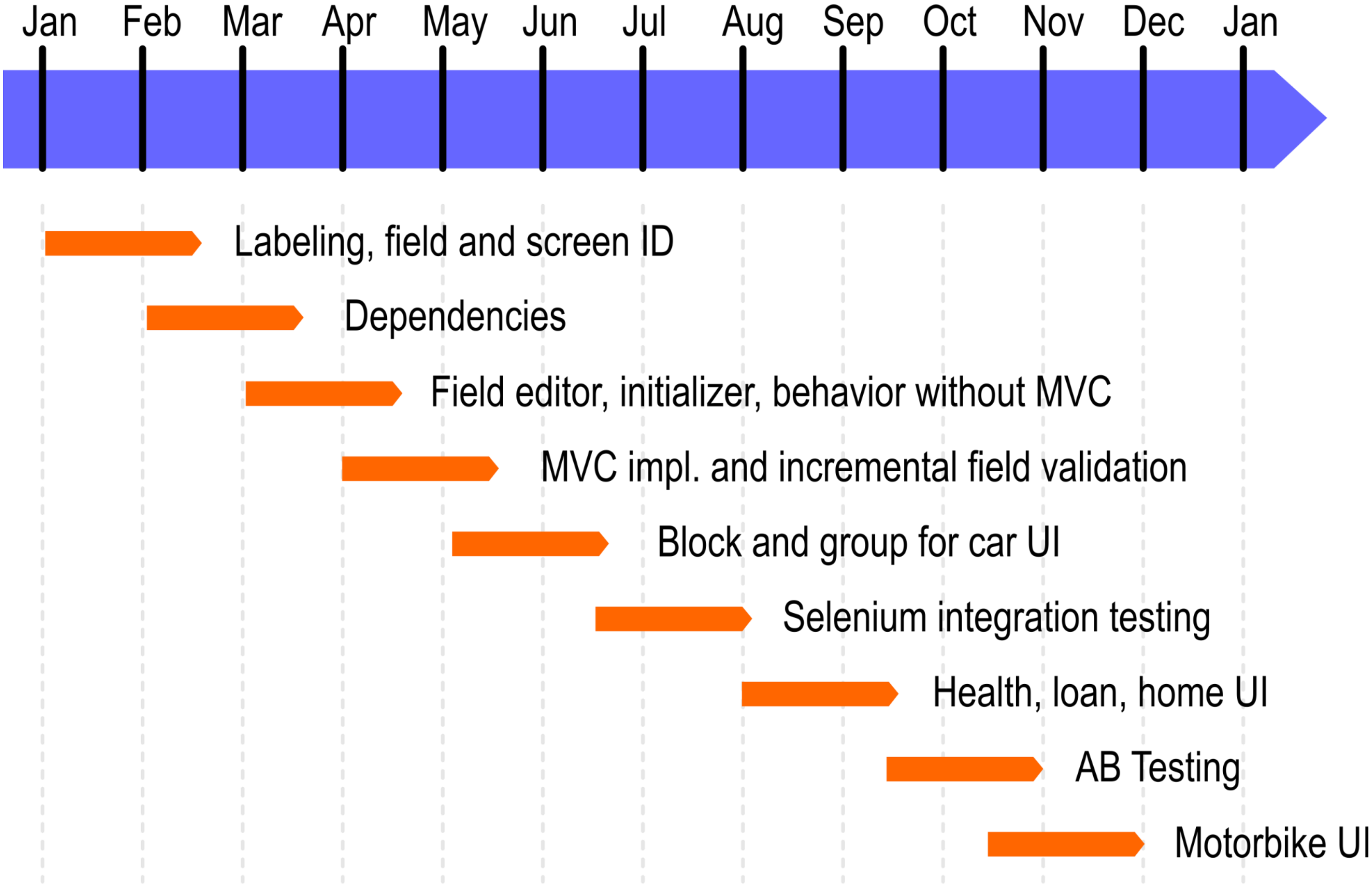
Live coding demo

**Back to LesFurets.com**

# Refactoring and Agile practice

- Opportunity based
- 12 iterations with production deployment
- 1 year of step by step refactoring
- Test coverage from 10% to 50% (in progress)
- Automated testing on more than 400 fields in 5 complex forms

# Project implementation timeline



Labeling, field and screen ID

Dependencies

Field editor, initializer, behavior without MVC

MVC impl. and incremental field validation

Block and group for car UI

Selenium integration testing

Health, loan, home UI

AB Testing

Motorbike UI

# Under investigation

- Multi Variable Testing
- Machine learning algorithm on field tracking
- Dynamic shuffling of the fields order
- Adaptive path for the forms completion

# Enjoy MDL4UI