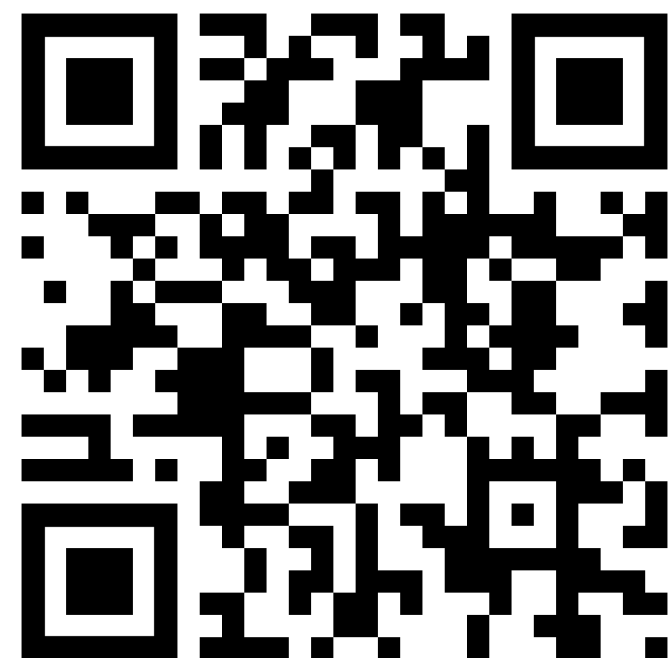




# Шаблонный доклад



<http://github.com/road21/talks>



**Алексей Троицкий**

@ Telegram, github: [road21](#)

# План

1. Ставим задачу
2. Решаем на scala
3. Усложняем задачу
4. Опять решаем на scala
5. Опять усложняем
6. Опять решаем на scala

# Задача

Движок для заполнения форм (например, для процесса подачи заявки)

- фронт получает описание формы которую нужно заполнить пользователю
- В описании формы есть вся необходимая информация для заполнения и валидации
- фронт не знает что именно заполняет пользователь, вся логика реализуется на бэке

### Введите свои данные

Имя \*

Мой ответ

Фамилия \*

Мой ответ

Дата рождения \*

Дата

дд.мм.гггг

Отправить

# Протокол

```
{
  "title" : "Введите свои данные",
  "fields" : [
    { "title" : "Имя",
      "hint" : "Мой ответ",
      "type" : "string" },
    { "title" : "Фамилия",
      "hint" : "Мой ответ",
      "type" : "string" },
    { "title" : "Дата рождения",
      "hint" : "Дата",
      "type" : "date" }
  ],
  "actions" : [
    { "id" : "send",
      "title" : "Отправить",
      "type" : "Send" }
  ]
}
```

## Введите свои данные

Имя \*

Мой ответ

Фамилия \*

Мой ответ

Дата рождения \*

Дата

дд.мм.гггг

Отправить

# Протокол

Введите свои данные

Имя \*

Владимир

Фамилия \*

Ленский

Дата рождения \*

Дата

27.07.1823

Отправить

```
{
  "fields" : [
    { "string" : { "value" : "Владимир" } },
    { "string" : { "value" : "Ленский" } },
    { "date" : { "value" : "1823-07-27" } }
  ],
  "actionId" : "send"
}
```

# Модели

```
case class Form(title: String, fields: List[Field], actions: List[Action])
```

```
case class Field(title: String, hint: String, `type`: FieldType)
```

```
case class Action(id: String, title: String, `type`: ActionType)
```

```
enum FieldType:  
  case string, date
```

```
enum ActionType:  
  case send
```

# Модели

```
case class Form(title: String, fields: List[Field], actions: List[Action])
```

```
case class Field(title: String, hint: String, `type`: FieldType)
```

```
case class Action(id: String, title: String, `type`: ActionType)
```

```
enum FieldType:  
  case string, date
```

```
enum ActionType:  
  case send
```

```
case class FormValue(fields: List[FieldValue], actionId: String)
```

```
enum FieldValue:  
  case string(value: String)  
  case date(value: LocalDate)
```



# Кодеки

```
import io.circe.{Encoder, Decoder}

case class Form(title: String, fields: List[Field], actions: List[Action])

case class Field(title: String, hint: String, `type`: FieldType)

case class Action(id: String, title: String, `type`: ActionType)

enum FieldType:
  case string, date

enum ActionType:
  case send

case class FormValue(fields: List[FieldValue], actionId: String)

enum FieldValue:
  case string(value: String)
  case date(value: LocalDate)
```

# Кодеки

```
import io.circe.{Encoder, Decoder}
case class Form(title: String, fields: List[Field], actions: List[Action]) derives Encoder.AsObject

case class Field(title: String, hint: String, `type`: FieldType) derives Encoder.AsObject

case class Action(id: String, title: String, `type`: ActionType) derives Encoder.AsObject

enum FieldType derives Encoder.AsObject:
  case string, date

enum ActionType derives Encoder.AsObject:
  case send

case class FormValue(fields: List[FieldValue], actionId: String) derives Decoder

enum FieldValue derives Decoder:
  case string(value: String)
  case date(value: LocalDate)
```

# Кодеки

```
import io.circe.{Encoder, Decoder}
case class Form(title: String, fields: List[Field], actions: List[Action]) derives Encoder.AsObject

case class Field(title: String, hint: String, `type`: FieldType) derives Encoder.AsObject

case class Action(id: String, title: String, `type`: ActionType) derives Encoder.AsObject

enum FieldType derives Encoder.AsObject:
  case string, date
  object Action:
    given Encoder.AsObject[Action] = Encoder.AsObject.derived

enum ActionType derives Encoder.AsObject:
  case send

case class FormValue(fields: List[FieldValue], actionId: String) derives Decoder

enum FieldValue derives Decoder:
  case string(value: String)
  case date(value: LocalDate)
```

# Кодеки

```
import io.circe.{Encoder, Decoder}
case class Form(title: String, fields: List[Field], actions: List[Action]) derives Encoder.AsObject

case class Field(title: String, hint: String, `type`: FieldType) derives Encoder.AsObject

case class Action(id: String, title: String, `type`: ActionType) derives Encoder.AsObject

enum FieldType derives Encoder.AsObject:
  case string, date

enum ActionType derives Encoder.AsObject:
  case send

case class FormValue(fields: List[FieldValue], actionId: String) derives Decoder

enum FieldValue derives Decoder:
  case string(value: String)
  case date(value: LocalDate)
```

# Кодеки

```
import io.circe.{Encoder, Decoder}
case class Form(title: String, fields: List[Field], actions: List[Action]) derives Encoder.AsObject

case class Field(title: String, hint: String, `type`: FieldType) derives Encoder.AsObject

case class Action(id: String, title: String, `type`: ActionType) derives Encoder.AsObject

enum FieldType derives Encoder.AsObject:
  case string, date

enum ActionType derives Encoder.AsObject:
  case send

case class FormValue(fields: List[FieldValue], actionId: String) derives Decoder

enum FieldValue derives Decoder:
  case string(value: String)
  case date(value: LocalDate)
```

```
"type" : {
  "string" : {}
}
```

```
"type" : {
  "send" : {}
}
```

# Кодеки

```
import io.circe.{Encoder, Decoder}
case class Form(title: String, fields: List[Field], actions: List[Action]) derives Encoder.AsObject

case class Field(title: String, hint: String, `type`: FieldType) derives Encoder.AsObject

case class Action(id: String, title: String, `type`: ActionType) derives Encoder.AsObject

enum FieldType:
  case string, date
  object FieldType:
    given Encoder[FieldType] = Encoder[String].contramap(_.toString)

enum ActionType:
  case send
  object ActionType:
    given Encoder[ActionType] = Encoder[String].contramap(_.toString)

case class FormValue(fields: List[FieldValue], actionId: String) derives Decoder

enum FieldValue derives Decoder:
  case string(value: String)
  case date(value: LocalDate)
```

# Локализация

```
enum Lang:  
    case RU, EN
```

# Локализация

```
enum Lang:
```

```
    case RU, EN
```

```
val keys: Map[String, (String, String)] = Map(  
    "name_form_title" -> ("Введите свои данные", "Enter personal info"),  
    "first_name_field_title" -> ("Имя", "First name"),  
    "last_name_field_title" -> ("Фамилия", "Last name"),  
    "name_hint" -> ("Мой ответ", "My answer"),  
    "birth_date_field_title" -> ("Дата рождения", "Date of birth"),  
    "birth_date_hint" -> ("Дата", "Date"),  
    "confirm_action_title" -> ("Подтвердить", "Confirm"),  
)
```



# Локализация

```
enum Lang:
```

```
  case RU, EN
```

```
val keys: Map[String, (String, String)] = Map(  
  "name_form_title" -> ("Введите свои данные", "Enter personal info"),  
  "first_name_field_title" -> ("Имя", "First name"),  
  "last_name_field_title" -> ("Фамилия", "Last name"),  
  "name_hint" -> ("Мой ответ", "My answer"),  
  "birth_date_field_title" -> ("Дата рождения", "Date of birth"),  
  "birth_date_hint" -> ("Дата", "Date"),  
  "confirm_action_title" -> ("Подтвердить", "Confirm"),  
)
```

```
def localize(lang: Lang): String => String = key =>  
  keys.get(key).fold(key) {  
    case (ru, en) =>  
      lang match  
        case Lang.RU => ru  
        case Lang.EN => en  
  }
```

# Локализация

```
enum Lang:
```

```
  case RU, EN
```

```
val keys: Map[String, (String, String)] = Map(  
  "name_form_title" -> ("Введите свои данные", "Enter personal info"),  
  "first_name_field_title" -> ("Имя", "First name"),  
  "last_name_field_title" -> ("Фамилия", "Last name"),  
  "name_hint" -> ("Мой ответ", "My answer"),  
  "birth_date_field_title" -> ("Дата рождения", "Date of birth"),  
  "birth_date_hint" -> ("Дата", "Date"),  
  "confirm_action_title" -> ("Подтвердить", "Confirm"),  
)
```

```
def localize(lang: Lang): String => String = key =>  
  keys.get(key).fold(key) { // не обрабатываем отсутствие перевода  
    case (ru, en) =>  
      lang match  
        case Lang.RU => ru  
        case Lang.EN => en  
  }
```

# Локализация

```
def localizeForm(lang: Lang): Form => Form = ???
```

# Локализация

```
def localizeField(lang: Lang): Field => Field = ???
```

```
def localizeAction(lang: Lang): Action => Action = ???
```

```
def localizeForm(lang: Lang): Form => Form = ???
```

# Локализация

```
def localizeField(lang: Lang): Field => Field =  
  case Field(title, hint, typ) =>  
    Field(localize(lang)(title), localize(lang)(hint), typ)
```

```
def localizeAction(lang: Lang): Action => Action =  
  case Action(id, title, typ) =>  
    Action(id, localize(lang)(title), typ)
```

```
def localizeForm(lang: Lang): Form => Form = ???
```

# Локализация

```
def localizeField(lang: Lang): Field => Field =  
  case Field(title, hint, typ) =>  
    Field(localize(lang)(title), localize(lang)(hint), typ)
```

```
def localizeAction(lang: Lang): Action => Action =  
  case Action(id, title, typ) =>  
    Action(id, localize(lang)(title), typ)
```

```
def localizeForm(lang: Lang): Form => Form =  
  case Form(title, fields, actions) =>  
    Form(  
      localize(lang)(title),  
      fields.map(localizeField(lang)),  
      actions.map(localizeAction(lang)))  
)
```

# Локализация

```
def localizeField(lang: Lang): Field => Field =  
  case Field(title, hint, typ) =>  
    Field(localize(lang)(title), localize(lang)(hint), typ)
```

```
def localizeAction(lang: Lang): Action => Action =  
  case Action(id, title, typ) =>  
    Action(id, title, typ)
```

```
def localizeForm(lang: Lang): Form => Form =  
  case Form(title, fields, actions) =>  
    Form(  
      localize(lang)(title),  
      fields.map(localizeField(lang)),  
      actions.map(localizeAction(lang)))  
  )
```

# Локализация

```
def localizeField(lang: Lang): Field => Field =  
  case Field(title, hint, typ) =>  
    Field(localize(lang)(title), localize(lang)(hint), typ)
```

```
def localizeAction(lang: Lang): Action => Action =  
  case Action(id, title, typ) =>  
    Action(id, localize(lang)(title), typ)
```

```
def localizeForm(lang: Lang): Form => Form =  
  case Form(title, fields, actions) =>  
    Form(  
      localize(lang)(title),  
      fields.map(localizeField(lang)),  
      actions.map(localizeAction(lang)))  
)
```



# Локализация

```
case class Form(title: Text, fields: List[Field], actions: List[Action])
```

```
case class Field(title: Text, hint: Text, typ: FieldType)
```

```
case class Action(actionId: String, title: Text, typ: ActionType)
```

```
case class FormL(title: LKey, fields: List[FieldL], actions: List[ActionL])
```

```
case class FieldL(title: LKey, hint: LKey, typ: FieldType)
```

```
case class ActionL(actionId: String, title: LKey, typ: ActionType)
```

# Локализация

```
case class Form(title: Text, fields: List[Field], actions: List[Action])
```

```
case class Field(title: Text, hint: Text, typ: FieldType)
```

```
case class Action(actionId: String, title: Text, typ: ActionType)
```

```
case class FormL(title: LKey, fields: List[FieldL], actions: List[ActionL])
```

```
case class FieldL(title: LKey, hint: LKey, typ: FieldType)
```

```
case class ActionL(actionId: String, title: LKey, typ: ActionType)
```

# Локализация

```
case class Form(title: Text, fields: List[Field], actions: List[Action])
```

```
case class Field(title: Text, hint: Text, typ: FieldType)
```

```
case class Action(actionId: String, title: Text, typ: ActionType)
```

```
case class FormL(title: LKey, fields: List[FieldL], actions: List[ActionL])
```

```
case class FieldL(title: LKey, hint: LKey, typ: FieldType)
```

```
case class ActionL(actionId: String, title: LKey, typ: ActionType)
```

```
def localize(lang: Lang): LKey => Text = ???
```

```
def localizeField(lang: Lang): FieldL => Field = ???
```

```
def localizeAction(lang: Lang): ActionL => Action = ???
```

```
def localizeForm(lang: Lang): FormL => Form = ???
```

# New types

```
type Text = String
```

```
type LKey = String
```

# New types

```
type Text = String
```

```
type LKey = String
```

```
def localizeAction(lang: Lang): ActionL => Action =  
  case ActionL(actionId, title, typ) =>  
    Action(actionId, title, typ)
```

# New types

```
type Text <: String
```

```
type LKey <: String
```

# New types

```
type Text = Text.T  
object Text:  
  type T <: String
```

```
type LKey = LKey.K  
object LKey:  
  type K <: String
```

# New types

```
type Text = Text.T
object Text:
  type T <: String
  def apply(str: String): T = str.asInstanceOf[T]
```

```
type LKey = LKey.K
object LKey:
  type K <: String
  def apply(str: String): K = str.asInstanceOf[K]
```



# New types

```
type Text = Text.T
object Text:
  opaque type T <: String = String
  def apply(key: String): T = key
```

```
type LKey = LKey.K
object LKey:
  opaque type K <: String = String
  def apply(key: String): K = key
```

# New types

```
type Text = Text.T
object Text:
  opaque type T <: String = String
  def apply(key: String): T = key
```

```
type LKey = LKey.K
object LKey:
  opaque type K <: String = String
  def apply(key: String): K = key
```

Снаружи

Внутри

# New types

```
type Text = Text.T
object Text extends TextInstances:
  opaque type T <: String = String
  def apply(key: String): T = key

trait TextInstances:
  given Encoder[Text] = Encoder[String].contramap(x => x)

type LKey = LKey.K
object LKey extends LKeyInstances:
  opaque type K <: String = String
  def apply(key: String): K = key

trait LKeyInstances:
  given Encoder[LKey] = Encoder[String].contramap(x => x)
```

# Локализация

```
def localize(lang: Lang): LKey => Text = ...

def localizeAction(lang: Lang): ActionL => Action =
  case ActionL(actionId, title, typ) =>
    Action(actionId, localize(lang)(title), typ)

def localizeField(lang: Lang): FieldL => Field =
  case FieldL(title, hint, typ) =>
    Field(localize(lang)(title), localize(lang)(hint), typ)

def localizeForm(lang: Lang): FormL => Form =
  case FormL(title, fields, actions) =>
    Form(
      localize(lang)(title),
      fields.map(localizeField(lang)),
      actions.map(localizeAction(lang))
    )
```

# Локализация

```
def localize(lang: Lang): LKey => Text = ...

def localizeAction(lang: Lang): ActionL => Action =
  case ActionL(actionId, title, typ) =>
    Action(actionId, localize(lang)(title), typ)

def localizeField(lang: Lang): FieldL => Field =
  case FieldL(title, hint, typ) =>
    Field(localize(lang)(title), localize(lang)(title), typ)

def localizeForm(lang: Lang): FormL => Form =
  case FormL(title, fields, actions) =>
    Form(
      localize(lang)(title),
      fields.map(localizeField(lang)),
      actions.map(localizeAction(lang))
    )
```

# Что получилось

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  typ: FieldType  
)
```

```
case class ActionL(  
  actionId: String,  
  title: LKey,  
  typ: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[FieldL],  
  actions: List[ActionL]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  typ: FieldType  
)
```

```
case class Action(  
  actionId: String,  
  title: Text,  
  typ: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Что получилось

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  typ: FieldType  
)  
  
case class ActionL(  
  actionId: String,  
  title: LKey,  
  typ: ActionType  
)  
  
case class FormL(  
  title: LKey,  
  fields: List[FieldL],  
  actions: List[ActionL]  
)
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
)  
  
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
)  
  
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  typ: FieldType  
)  
  
case class Action(  
  actionId: String,  
  title: Text,  
  typ: ActionType  
)  
  
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Что получилось

```
case class FieldL(  
  title: LKey,  
  hint: LKey, Field[LKey]  
  typ: FieldType  
)
```

```
case class ActionL(  
  actionId: String,  
  title: LKey, Action[LKey]  
  typ: ActionType  
)
```

```
case class FormL(  
  title: LKey, Form[LKey]  
  fields: List[FieldL],  
  actions: List[ActionL]  
)
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
)
```

```
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
)
```

```
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
)
```

**Field[Text]**

**Action[Text]**

**Form[Text]**

```
case class Field(  
  title: Text,  
  hint: Text,  
  typ: FieldType  
)
```

```
case class Action(  
  actionId: String,  
  title: Text,  
  typ: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```



# Field[A]

```
def localize(lang: Lang): LKey => Text = ...
```

```
def localizeForm(lang: Lang): Form[LKey] => Form[Text] =  
  ???
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
)  
  
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
)  
  
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
)
```

# Field[A]

```
def localize(lang: Lang): LKey => Text = ...
```

```
def localizeForm(lang: Lang): Form[LKey] => Form[Text] =  
  _.map(localize(lang))
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
)  
  
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
)  
  
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
)
```

# Field[A]

```
def localize(lang: Lang): LKey => Text = ...
```

```
def localizeForm(lang: Lang): Form[LKey] => Form[Text] =  
  Functor[Form].map(_)(localize(lang))
```

```
trait Functor[F[_]]:  
  def map[A, B](fa: F[A])(f: A => B): F[B]
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
)  
  
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
)  
  
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
)
```

# Field[A]

```
def localize(lang: Lang): LKey => Text = ...
```

```
def localizeForm(lang: Lang): Form[LKey] => Form[Text] =  
  Functor[Form].map(_)(localize(lang))
```

```
trait Functor[F[_]]:  
  def map[A, B](fa: F[A])(f: A => B): F[B]
```

```
import cats.Functor  
import cats.derived.* // kittens
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
) derives Functor  
  
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
) derives Functor  
  
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
) derives Functor
```

# А что там с кодеками?

```
import cats.derived.*
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
) derives Functor
```

```
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
) derives Functor
```

```
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
) derives Functor
```

# А что там с кодеками?

```
import cats.derived.*
```

```
case class Field[A](  
  title: A,  
  hint: A,  
  typ: FieldType  
) derives Functor, Encoder.AsObject
```

```
case class Action[A](  
  id: String,  
  title: A,  
  typ: ActionType  
) derives Functor, Encoder.AsObject
```

```
case class Form[A](  
  title: A,  
  fields: List[Field[A]],  
  actions: List[Action[A]]  
) derives Functor, Encoder.AsObject
```

# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
Encoder.AsObject[Form[LKey]]
```

```
[error] 77 | Encoder.AsObject[Form[LKey]] ^
[error]    |
[error]    |No given instance of type
io.circe.Encoder.AsObject[routine.Form[routi
ne.LKey]] was found for parameter instance
of method apply in object AsObject.
```

```
...
[error]    |But no implicit values were
found that match type
io.circe.export.Exported[io.circe.Encoder.As
Object[routine.LKey.T]].
```

# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
Encoder.AsObject[Form[LKey]]
```

```
[error] 77 | Encoder.AsObject[Form[LKey]]
[error]    |                                     ^
[error]    |No given instance of type
io.circe.Encoder.AsObject[routine.Form[routi
ne.LKey]] was found for parameter instance
of method apply in object AsObject.
```

```
...
[error]    |But no implicit values were
found that match type
io.circe.export.Exported[io.circe.Encoder.As
Object[routine.LKey.T]].
```



# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
object Field:
  given [A: Encoder.AsObject]: Encoder.AsObject[Field[A]] =
    Encoder.AsObject.derived[Field[A]]
```

```
Encoder.AsObject[Form[LKey]]
```

```
[error] 77 | Encoder.AsObject[Form[LKey]]
[error]    |                                     ^
[error]    |No given instance of type
io.circe.Encoder.AsObject[routine.Form[routi
ne.LKey]] was found for parameter instance
of method apply in object AsObject.
```

```
...
[error]    |But no implicit values were
found that match type
io.circe.export.Exported[io.circe.Encoder.As
Object[routine.LKey.T]].
```

# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
object Field:
  given [A: Encoder]: Encoder.AsObject[Field[A]] =
    Encoder.AsObject.derived[Field[A]]
```

# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
object Field:
  given [A: Encoder]: Encoder.AsObject[Field[A]] =
    Encoder.AsObject.derived[Field[A]]
```

# А что там с кодеками?

```
import cats.derived.*

case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder.AsObject

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder.AsObject

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder.AsObject
```

```
object Field:
  given [A: Encoder]: Encoder[Field[A]] =
    Encoder.AsObject.derived[Field[A]]
```

# А что там с кодеками?

```
import cats.derived.*
import EncoderDerived.*
case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder

case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder

case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder
```

```
object EncoderDerived:
  extension (e: Encoder.type)
    inline def derived[A](using inline A: Mirror.Of[A]):
Encoder[A] =
  Encoder.AsObject.derived[A]
```

# А что там с кодеками?

```
import cats.derived.*
import EncoderDerived.*
case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder
```

```
case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder
```

```
case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder
```

```
object EncoderDerived:
  extension (e: Encoder.type)
    inline def derived[A](using inline A: Mirror.Of[A]):
Encoder[A] =
  Encoder.AsObject.derived[A]
```

```
[error] -- [E008] Not Found Error
[error]   |) derives Functor, Encoder.AsObject
[error]   |      ^
[error]   |      value derived is not a member of object cats.Functor.
```

# А что там с кодеками?

```
import cats.derived.*
import EncoderDerived.*
case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder
```

```
case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder
```

```
case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder
```

```
object EncoderDerived:
  extension (e: Encoder.type)
    inline def derived[A](using inline A: Mirror.Of[A]):
Encoder[A] =
  Encoder.AsObject.derived[A]
```

```
[error] -- [E008] Not Found Error
[error]   |) derives Functor, Encoder.AsObject
[error]   |      ^
[error]   |      value derived is not a member of object cats.Functor.
```

```
extension (x: cats.Functor.type)
  inline def derived[F[_]]: cats.Functor[F] = ...
```

# А что там с кодеками?

```
import cats.derived.*
import EncoderDerived.*
case class Field[A](
  title: A,
  hint: A,
  typ: FieldType
) derives Functor, Encoder
```

```
case class Action[A](
  id: String,
  title: A,
  typ: ActionType
) derives Functor, Encoder
```

```
case class Form[A](
  title: A,
  fields: List[Field[A]],
  actions: List[Action[A]]
) derives Functor, Encoder
```

```
object EncoderDerived:
  implicit class EncoderSyntax(e: Encoder.type):
    inline def derived[A](using inline A: Mirror.Of[A]):
Encoder[A] =
  Encoder.AsObject.derived[A]
```



# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]](lang: Lang)(using loc: Localize[F]): Field[LKey] => F[Field[Text]] =  
  temp =>  
    Functor[Field].map(temp)(loc.localize(lang, _)): Field[F[Text]]
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]](lang: Lang)(using loc: Localizer[F]): Field[LKey] => F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]](lang: Lang)(using loc: Localizer[F]): Field[LKey] => F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))  
  
(F[Text], F[Text], FieldType) => F[Field[Text]]
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]](lang: Lang)(using loc: Localizer[F]): Field[LKey] => F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))  
  
(F[Text], F[Text], FieldType) => F[Field[Text]]  
  
(F[Text], F[Text]) => F[(Text, Text)]  
(F[(Text, Text)], FieldType) => F[Field[Text]]
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]](lang: Lang)(using loc: Localizer[F]): Field[LKey] => F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))
```

```
(F[Text], F[Text], FieldType) => F[Field[Text]]
```

```
(F[Text], F[Text]) => F[(Text, Text)]
```

```
(F[(Text, Text)], FieldType) => F[Field[Text]] // Functor
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Field[LKey] =>  
  F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))
```

```
(F[Text], F[Text], FieldType) => F[Field[Text]]  
  
(F[Text], F[Text]) => F[(Text, Text)] // Applicative  
(F[(Text, Text)], FieldType) => F[Field[Text]] // Functor
```

```
trait Applicative[F[_]] extends Functor[F]:  
  def ap[A, B](ff: F[A => B])(fa: F[A]): F[B]  
  def pure[A](x: A): F[A]  
  
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)] =  
    ap(map(fa)(a => (b: B) => (a, b)))(fb)
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Field[LKey] =>  
  F[Field[Text]] =  
  temp =>  
    val Field(title: F[Text], hint: F[Text], typ: FieldType) =  
      Functor[Field].map(temp)(loc.localize(lang, _))  
    Applicative[F].product(title, hint).map((t, h) => Field(t, h, typ))
```

```
trait Applicative[F[_]] extends Functor[F]:  
  def ap[A, B](ff: F[A => B])(fa: F[A]): F[B]  
  def pure[A](x: A): F[A]  
  
  def product[A, B](fa: F[A], fb: F[B]): F[(A, B)] =  
    ap(map(fa)(a => (b: B) => (a, b)))(fb)
```



# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Field[LKey] =>  
  F[Field[Text]] =  
    temp =>  
      val fieldF: Field[F[Text]] = Functor[Form].map(temp)(loc.localize(lang, _))
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Field[LKey] =>  
  F[Field[Text]] =  
  temp =>  
    val fieldF: Field[F[Text]] = Functor[Form].map(temp)(loc.localize(lang, _))  
    Traverse[Field].sequence(fieldF)
```

```
trait Traverse[F[_]] extends Functor[F]:  
  def traverse[G[_]: Applicative, A, B](fa: F[A])(f: A => G[B]): G[F[B]]  
  
  def sequence[G[_]: Applicative, A](fga: F[G[A]]): G[F[A]] = traverse(fga)(ga => ga)
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeField[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Field[LKey] =>  
  F[Field[Text]] =  
    Traverse[Field].traverse(_)(loc.localize(lang, _))
```

```
trait Traverse[F[_]] extends Functor[F]:  
  def traverse[G[_]: Applicative, A, B](fa: F[A])(f: A => G[B]): G[F[B]]  
  
  def sequence[G[_]: Applicative, A](fga: F[G[A]]): G[F[A]] = traverse(fga)(ga => ga)
```

# Локализация с эффектом

```
trait Localizer[F[_]]:  
  def localize(lang: Lang, key: LKey): F[Text]  
  
def localizeForm[F[_]: Applicative](lang: Lang)(using loc: Localizer[F]): Form[LKey] =>  
  F[Form[Text]] =  
    Traverse[Form].traverse(_)(loc.localize(lang, _))  
  
import cats.derived.* // kittens  
  
case class Field[A](title: A, hint: A, typ: FieldType)  
  derives Traverse, Encoder  
  
case class Action[A](id: String, title: A, typ: ActionType)  
  derives Traverse, Encoder  
  
case class Form[A](title: A, fields: List[Field[A]], actions: List[Action[A]])  
  derives Traverse, Encoder
```

# Шаблоны

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  typ: FieldType  
)
```

```
case class ActionL(  
  actionId: String,  
  title: LKey,  
  typ: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[FieldL],  
  actions: List[ActionL]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  typ: FieldType  
)
```

```
case class Action(  
  actionId: String,  
  title: Text,  
  typ: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Шаблоны

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp],  
  `type`: FieldType  
)
```

```
enum UserProp:  
  case birthDate, firstName, lastName
```

```
enum FieldValue derives Decoder:  
  case string(value: String)  
  case date(value: LocalDate)
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Шаблоны

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp],  
  `type`: FieldType  
)
```

```
enum UserProp:  
  case birthDate, firstName, lastName
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
enum FieldValue derives Decoder:  
  case string(value: String)  
  case date(value: LocalDate)
```

```
case class UserInfo(  
  firstName: Option[String],  
  lastName: Option[String],  
  birthDate: Option[LocalDate]  
)
```

```
def propToValue(userInfo: UserInfo):  
  Option[UserProp] => Option[FieldValue]  
= ...
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp],  
  `type`: FieldType  
)
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```



# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp]  
  `type`: FieldType  
)  
  
Field[LKey, Option[UserProp]]
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
Action[LKey, Option[UserProp]]
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
Form[LKey, Option[UserProp]]
```

```
case class Field[A, B](  
  title: A,  
  hint: A,  
  default: B,  
  `type`: FieldType  
)
```

```
case class Action[A, B](  
  id: String,  
  title: A,  
  `type`: ActionType  
)
```

```
case class Form[A, B](  
  title: A,  
  fields: List[Field[A, B]],  
  actions: List[Action[A, B]]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
Field[Text, Option[FieldValue]]
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
Action[Text, Option[FieldValue]]
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
Form[Text, Option[FieldValue]]
```

# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp]  
  `type`: FieldType  
)
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
case class Field[F[_]](  
  title: F[Localize],  
  hint: F[Localize],  
  default: F[Context],  
  `type`: FieldType  
)
```

```
case class Action[F[_]](  
  id: String,  
  title: F[Localize],  
  `type`: ActionType  
)
```

```
case class Form[F[_]](  
  title: F[Localize],  
  fields: List[Field[F]],  
  actions: List[Action[F]]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp]  
  `type`: FieldType  
)
```

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

```
type Localize  
type Context
```

```
case class Field[F[_]](  
  title: F[Localize],  
  hint: F[Localize],  
  default: F[Context],  
  `type`: FieldType  
)
```

```
case class Action[F[_]](  
  id: String,  
  title: F[Localize],  
  `type`: ActionType  
)
```

```
case class Form[F[_]](  
  title: F[Localize],  
  fields: List[Field[F]],  
  actions: List[Action[F]]  
)
```

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp]  
  `type`: FieldType  
)
```

Field[Template]

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

Action[Template]

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

Form[Template]

```
type Localize  
type Context
```

```
type Template[x] = ???  
type Rendered[x] = ???
```

```
case class Field[F[_]](  
  title: F[Localize],  
  hint: F[Localize],  
  default: F[Context],  
  `type`: FieldType  
)
```

Field[Rendered]

```
case class Action[F[_]](  
  id: String,  
  title: F[Localize],  
  `type`: ActionType  
)
```

Action[Rendered]

```
case class Form[F[_]](  
  title: F[Localize],  
  fields: List[Field[F]],  
  actions: List[Action[F]]  
)
```

Form[Rendered]

```
case class Field(  
  title: Text,  
  hint: Text,  
  default: Option[FieldValue],  
  `type`: FieldType  
)
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Можно ли обобщить?

```
case class FieldL(  
  title: LKey,  
  hint: LKey,  
  default: Option[UserProp]  
  `type`: FieldType  
)
```

Field[Template]

```
case class ActionL(  
  id: String,  
  title: LKey,  
  `type`: ActionType  
)
```

Action[Template]

```
case class FormL(  
  title: LKey,  
  fields: List[Field],  
  actions: List[Action]  
)
```

Form[Template]

```
type Localize  
type Context
```

```
type Template[x] = ???  
type Rendered[x] = ???
```

```
case class Field[F[_]](  
  title: F[Localize],  
  hint: F[Localize],  
  default: F[Context],  
  `type`: FieldType  
)
```

```
case class Action[F[_]](  
  id: String,  
  title: F[Localize],  
  `type`: ActionType  
)
```

```
case class Form[F[_]](  
  title: F[Localize],  
  fields: List[Field[F]],  
  actions: List[Action[F]]  
)
```

```
case class Field(  
  title: Text,
```

```
Template[Localize] = LKey  
Template[Context] = Option[UserProp]  
  
Rendered[Localize] = Text  
Rendered[Context] = Option[FieldValue]
```

```
case class Action(  
  id: String,  
  title: Text,  
  `type`: ActionType  
)
```

```
case class Form(  
  title: Text,  
  fields: List[Field],  
  actions: List[Action]  
)
```

# Form[F[\_]]

```
type Localize  
type Context
```

```
type Template[x] = ???  
type Rendered[x] = ???
```

```
Template[Localize] = LKey  
Template[Context] = Option[UserProp]  
  
Rendered[Localize] = Text  
Rendered[Context] = Option[FieldValue]
```

# GADT

```
type Localize  
type Context
```

```
enum Template[A]:  
  case L(key: LKey) extends Template[Localize]  
  case C(default: Option[UserProp]) extends Template[Context]
```

```
enum Rendered[A]:  
  case L(text: Text) extends Rendered[Localize]  
  case C(default: Option[FieldValue]) extends Rendered[Context]
```

```
Template[Localize] = LKey  
Template[Context] = Option[UserProp]  
  
Rendered[Localize] = Text  
Rendered[Context] = Option[FieldValue]
```

# GADT

```
type Localize
type Context
```

```
enum Template[A]:
  case L(key: LKey) extends Template[Localize]
  case C(default: Option[UserProp]) extends Template[Context]
```

```
enum Rendered[A]:
  case L(text: Text) extends Rendered[Localize]
  case C(default: Option[FieldValue]) extends Rendered[Context]
```

```
def localize(lang: Lang): LKey => Text = ...
```

```
def propToValue(user: UserInfo): Option[UserProp] => Option[FieldValue] = ...
```

```
def render[A](lang: Lang, user: UserInfo): Template[A] => Rendered[A] =
  case Template.L(key) => Rendered.L(localize(lang)(key))
  case Template.C(default) => Rendered.C(propToValue(user)(default))
```

```
Template[Localize] = LKey
Template[Context] = Option[UserProp]
```

```
Rendered[Localize] = Text
Rendered[Context] = Option[FieldValue]
```



# Match types

```
type Localize  
type Context
```

```
type Template[x] = x match  
  case Localize => LKey  
  case Context => Option[UserProp]
```

```
type Rendered[x] = x match  
  case Localize => Text  
  case Context => Option[FieldValue]
```

```
Template[Localize] = LKey  
Template[Context] = Option[UserProp]
```

```
Rendered[Localize] = Text  
Rendered[Context] = Option[FieldValue]
```

# Match types

```
type Localize  
type Context
```

```
type Template[x <: Localize | Context] = x match  
  case Localize => LKey  
  case Context => Option[UserProp]
```

```
type Rendered[x <: Localize | Context] = x match  
  case Localize => Text  
  case Context => Option[FieldValue]
```

```
Template[Localize] = LKey  
Template[Context] = Option[UserProp]
```

```
Rendered[Localize] = Text  
Rendered[Context] = Option[FieldValue]
```

# Match types

```
type Localize
type Context
```

```
type Template[x] = x match
  case Localize => LKey
  case Context => Option[UserProp]
```

```
type Rendered[x] = x match
  case Localize => Text
  case Context => Option[FieldValue]
```

```
def render[A](lang: Lang, user: UserInfo): Template[A] => Rendered[A] =
  case key: LKey => localize(lang)(key).asInstanceOf[Rendered[A]]
  case prop: Option[UserProp] => propToValue(user)(prop).asInstanceOf[Rendered[A]]
```

```
Template[Localize] = LKey
Template[Context] = Option[UserProp]
```

```
Rendered[Localize] = Text
Rendered[Context] = Option[FieldValue]
```

# Render Form[F[\_]]

```
def render[A](lang: Lang, user: UserInfo): Template[A] => Rendered[A] =  
  case Template.L(key) => Rendered.L(localize(lang)(key))  
  case Template.C(default) => Rendered.C(propToValue(user)(default))  
  
def renderForm(lang: Lang, user: UserInfo): Form[Template] => Form[Rendered] = ???
```

# Render Form[F[\_]]

```
def render[A](lang: Lang, user: UserInfo): Template[A] => Rendered[A] =  
  case Template.L(key) => Rendered.L(localize(lang)(key))  
  case Template.C(default) => Rendered.C(propToValue(user)(default))  
  
def renderForm(lang: Lang, user: UserInfo): Form[Template] => Form[Rendered] =  
  Functork[Form].mapK(_)([A] => (t: Template[A]) => render[A](lang, user)(t))
```

```
trait FunctorK[U[_[_]]]:  
  def mapK[F[_], G[_]](af: U[F])(fk: [A] => F[A] => G[A]): U[G]
```

# Render Form[F[\_]]

```
def render[A](lang: Lang, user: UserInfo): Template[A] => Rendered[A] =  
  case Template.L(key) => Rendered.L(localize(lang)(key))  
  case Template.C(default) => Rendered.C(propToValue(user)(default))  
  
def renderForm(lang: Lang, user: UserInfo): Form[Template] => Form[Rendered] =  
  FunctorK[Form].mapK(_)([A] => (t: Template[A]) => render[A](lang, user)(t))
```

```
trait FunctorK[U[_[_]]]:  
  def mapK[F[_], G[_]](af: U[F])(fk: [A] => F[A] => G[A]): U[G]
```

cats-tagless

# Render с эффектом

```
def renderF[A, F[_]: Functor](lang: Lang, user: UserInfo): Template[A] => F[Rendered[A]] =  
  case Template.L(key) => localizeF[F](lang).apply(key).map(Rendered.L(_))  
  case Template.C(default) => propToValueF[F](user).apply(default).map(Rendered.C(_))
```

```
def renderFormF[F[_]](lang: Lang, user: UserInfo): Form[Template] => F[Form[Rendered]] =  
  ???
```

# Render с эффектом

```
def renderF[A, F[_]: Functor](lang: Lang, user: UserInfo): Template[A] => F[Rendered[A]] =  
  case Template.L(key) => localizeF[F](lang).apply(key).map(Rendered.L(_))  
  case Template.C(default) => propToValueF[F](user).apply(default).map(Rendered.C(_))  
  
def renderFormF[F[_]: Applicative](lang: Lang, user: UserInfo): Form[Template] => F[Form[Rendered]] =  
  TraverseK[Form].traverseK(_)([A] => (t: Template[A]) => renderF[A](lang, user).apply(t))
```

```
trait TraverseK[U[_[_]]] extends FunctorK[U]:  
  def traverseK[A[_], B[_], F[_] : Applicative](ca: U[A])(f: [X] => X => F[B[X]]): F[U[B]]
```



# Итоги



Абстрактный код может быть  
безопаснее и короче



Повышается сложность кода  
Иногда не хватает тулинга

# Итоги



## Код со scala 3 стал чище



# Писать идиоматично везде пока не получается



**Спасибо!**