Migration d'un algo legacy vers de l'Event Sourcing :

REX d'un refacto de code

Thomas Girard

tgirard12/legacy-to-event-sourcing

tgirard12

The leading freelancing platform in Europe

700,000+ freelancers

70,000+ companies

700 Malters

malt.fr
careers.malt.com

🕰️ Il était une fois

- 🍝 Codebase legacy / vieillissante
- 👩‍🎓 Recrutement et on-boarding
- ⏳ Développement des features très / trop long

Comment (essayer) de résoudre ces problèmes

📄 Le projet V2 from scratch

- 💾 Assurer la migration des datas
- 🚀 Jamais mis en production
- 🪳 Oblige à recréer les mêmes bugs

🫡 Devscout rule

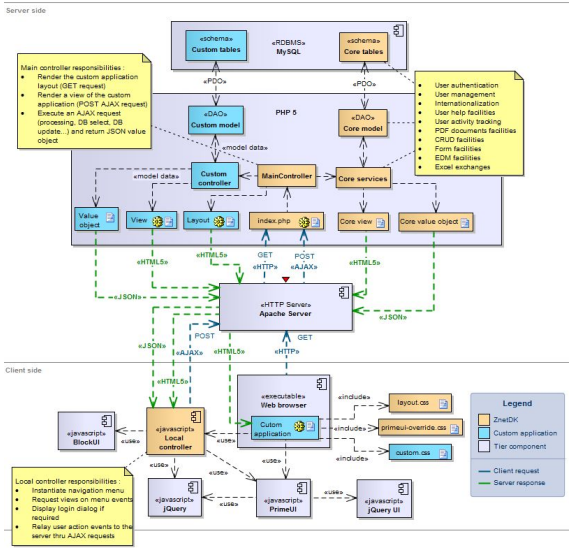
- 🧹 Laisser le code plus propre qu'en arrivant
- ⛰️ Savoir stopper les trop gros refactor
- ♾️ Jamais fini

🎯 Gros refactor ciblés

- 🗓️ Limité à 1-2 semaines
- 🎲 Sujet technique ou fonctionnel
- 🗃️ En lien avec la roadmap produit

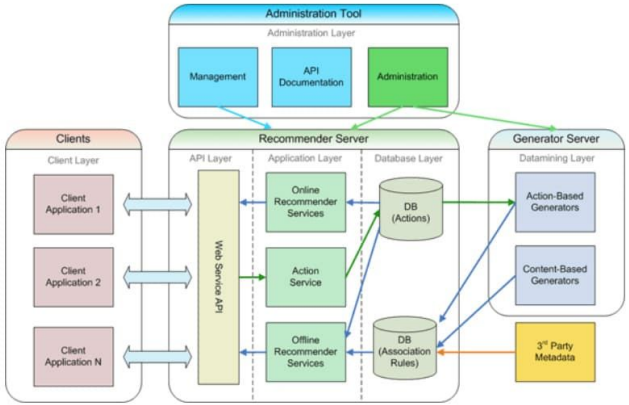
🙄 Comment choisir ?

- ✏️ Faire un état des lieux de l'application
- 🧓 Nécessite de l'expérience
- 🌍 Schéma des composants Infra, Techniques, Métiers, API, WebServices, Services, Algos



🌈 Une vision idéale

- 🗒️ Liste de tous les éléments à modifier
- 🏅 Classement par priorité & complexité & temps
- 🥔 La bonne granularité
- 🤝 Feuille de route



Refactor vers de l'Event Sourcing

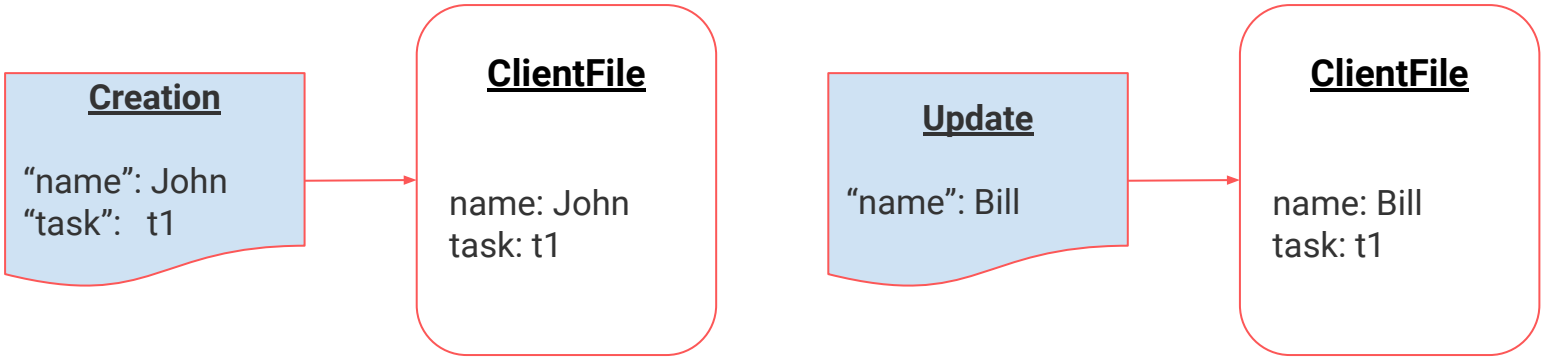
Event sourcing

- **ClientFile**
- **Task**
- **Event**

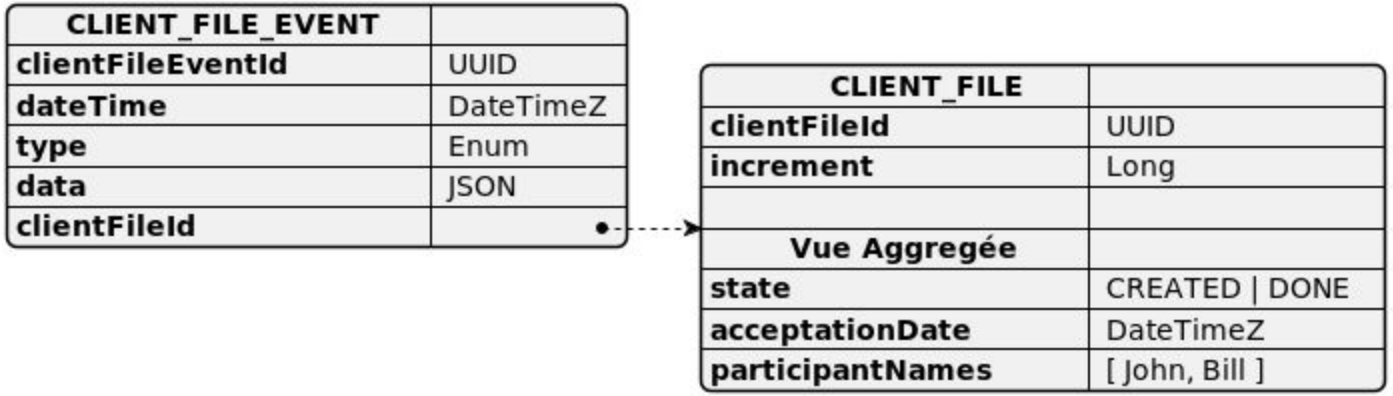
> *Persists the state of a business entity such an Order or a Customer as a sequence of state-changing events*

https://microservices.io/patterns/data/event-sourcing.html

Event sourcing



Base de données



Des Events

```
sealed interface ClientFileEvent {
    val clientFileEventId : UUID
    val clientFileId: UUID
    val eventNumber : Long
    val dateTime: OffsetDateTime
    val operatorId: UUID
    val type: Type

    enum class Type {
        CREATION,
        DOCUMENT_START,
        DOCUMENT,
        SIGNATURE,
        ACCEPTATION,
        REOPEN,
        UPDATE,
    }
}
```

Des Events

```
data class CreationEvent(
    override val clientFileEventId: UUID,
    override val clientFileId: UUID,
    override val dateTime: OffsetDateTime,
    override val operatorId: UUID,

    val participants: Participant,
    val tasks: List<Task>,
) : ClientFileEvent {
    override val eventNumber = 1L
    override val type = ClientFileEvent.Type.CREATION

    data class Participant(
        val participantId: UUID,
        val name: String,
        val email: String,
    )

    data class Task(
        val name: String,
        val type: String,
    )
}
```

Aggrega *ClientFile*

```
data class ClientFile private constructor(
    val clientFileId: UUID,
    val participant: Participant,
    val tasks: List<Task>,
    // ...
) {
    data class Participant(
        val participantId: UUID,
        val email: String,
        var name: String,
    )

    data class Task(
        val taskId: String,
        val type: Type,
        var state: State,
    ) {
        enum class Type { DOCUMENT, SIGNATURE }
        enum class State {
            UNAVAILABLE, TODO, IN_PROGRESS, DONE
        }
    ...
```

Aggrega *ClientFile*

```
data class ClientFile private constructor(

    // ...

) {
    // companion object
    fun create(ev: CreationEvent, fetcher: MutableFetcher)  =
        ClientFile(
            clientFileId = ev.clientFileId,
            creationDate = ev.dateTime,
            participant = Participant(
                participantId = ev.participant.participantId,
                name = ev.participant.name,
                email = ev.participant.email,
            ),
            tasks = ev.tasks.map { t ->
                Task(
                    taskId = t.name,
                    state = Task.State.UNAVAILABLE,
                    type = Task.Type.valueOf(t.type),
                )
            }
    )
```

Un exemple simple :

AcceptationDate

acceptationDate

Legacy

```
fun getLegacyAcceptanceDate(): OffsetDateTime? {
    val manualAcceptationDate = events
        .filterIsInstance<AcceptationEvent>()
        .firstOrNull()
        ?.dateTime
    val autoAcceptationDate = events
        .filterIsInstance<AutoAcceptationEvent>()
        .firstOrNull()
        ?.dateTime

    return when {
        manualAcceptationDate != null
          && autoAcceptationDate != null -> {
            if (manualAcceptationDate < autoAcceptationDate)
                manualAcceptationDate
            else
                autoAcceptationDate
        }

        manualAcceptationDate != null -> manualAcceptationDate
        autoAcceptationDate != null -> autoAcceptationDate
        else -> null
    }
}
```

acceptationDate

Event Sourcing

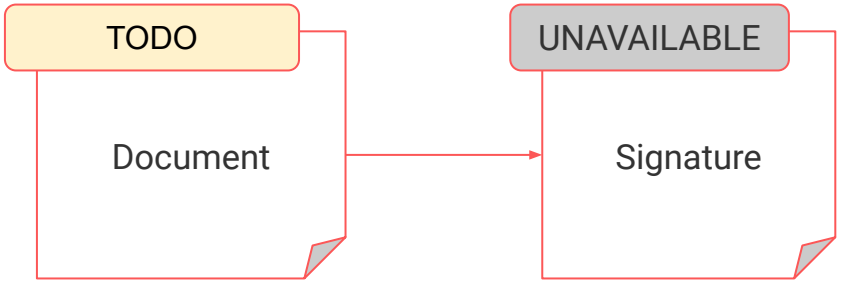
```
data class ClientFile private constructor() {

    //...

    private fun applyAcceptationEventDate(event: ClientFileEvent) {
        when (event) {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
                if (acceptationDate == null)
                    acceptationDate = event.dateTime
            }

            else -> {
            }
        }
    }
}
```

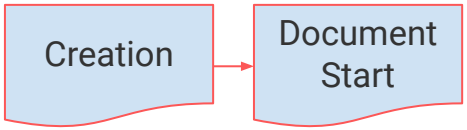
Un exemple plus compliqué

Task.state

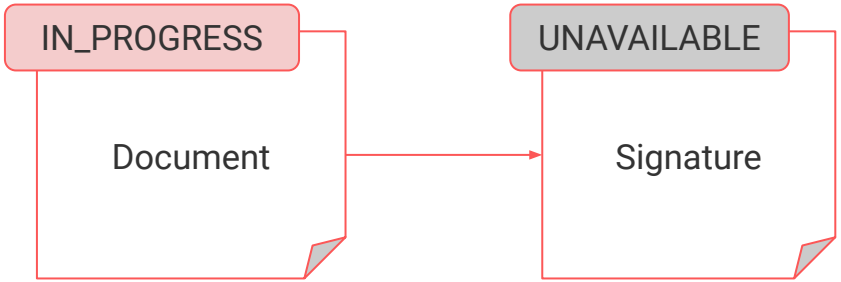
Creation

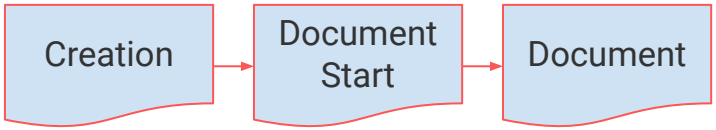
Participant



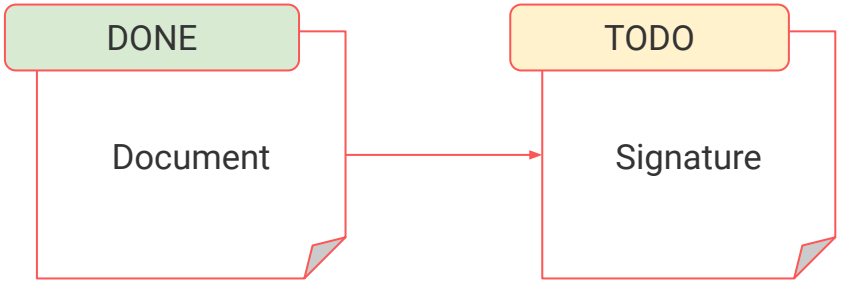


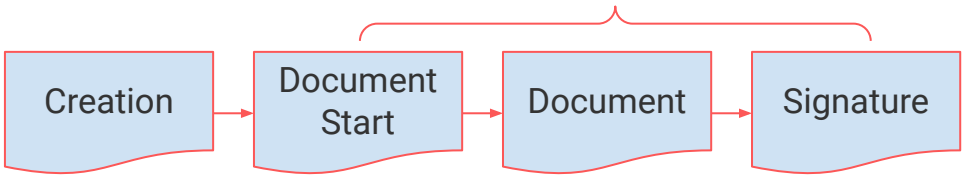
Participant



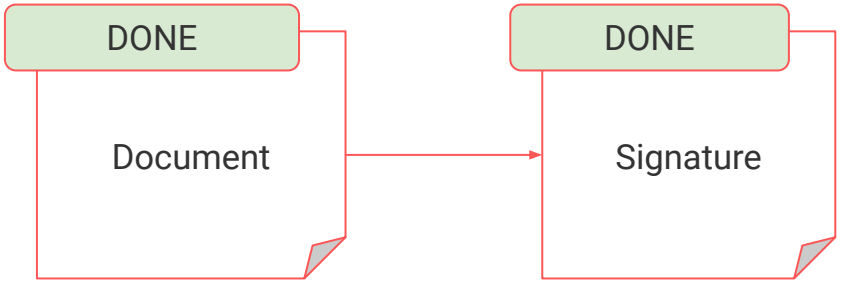


Participant

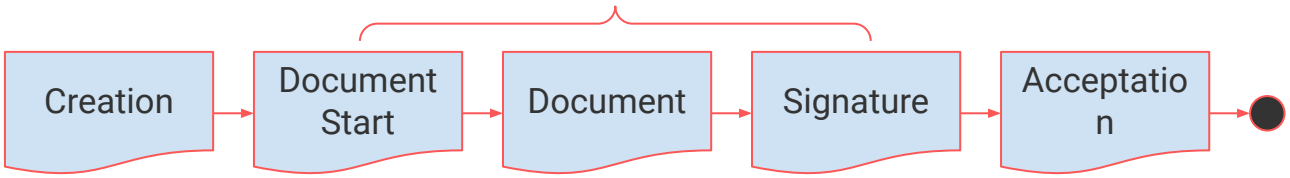




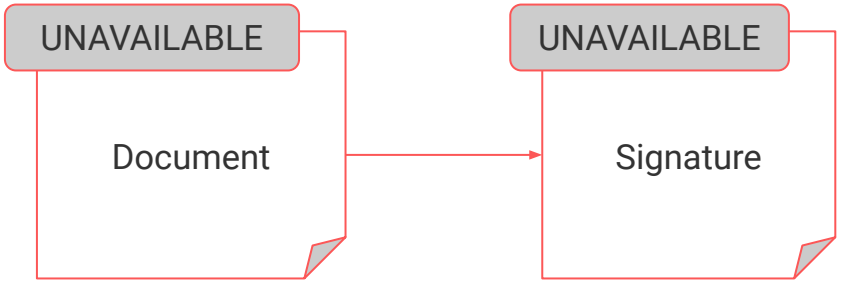
Participant



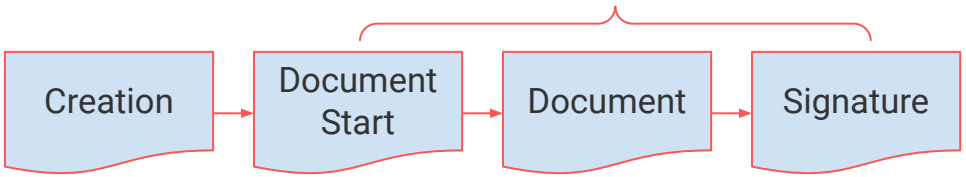
Opérateur



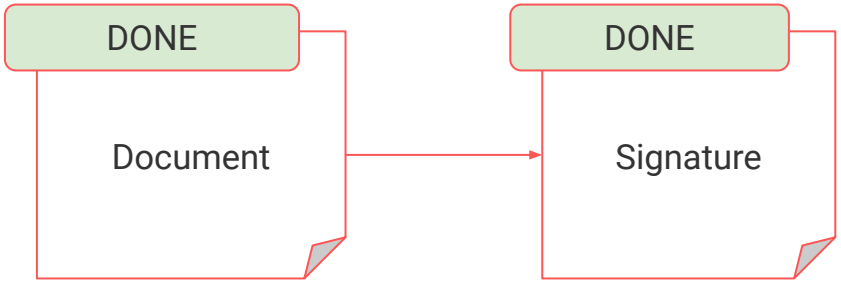
Participant



Opérateur

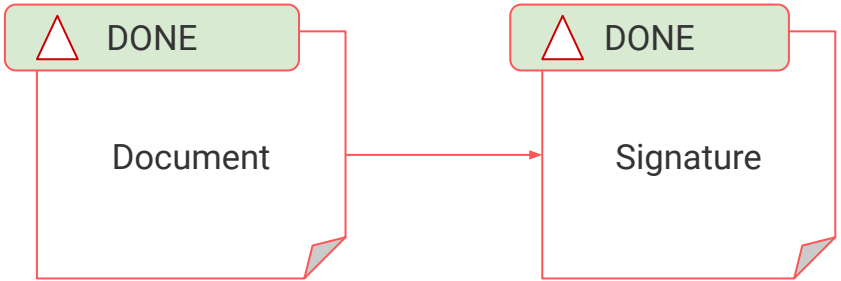


Participant

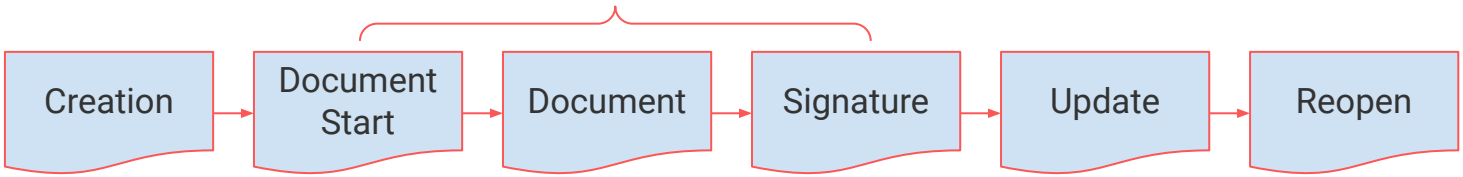


Opérateur

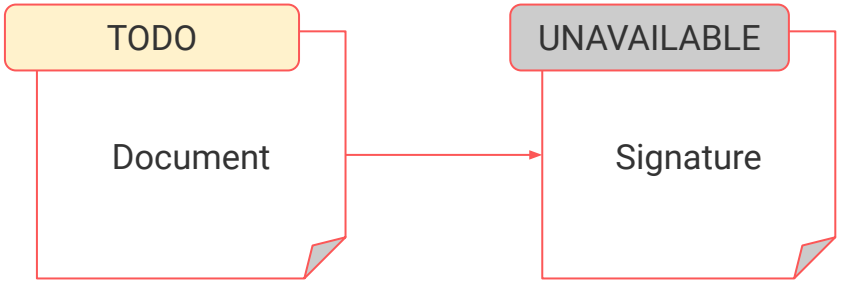
Participant

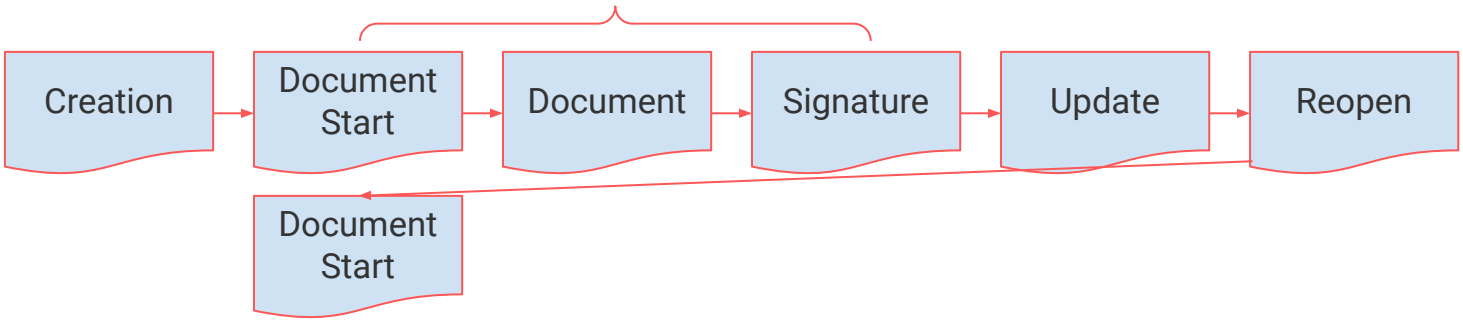


Opérateur

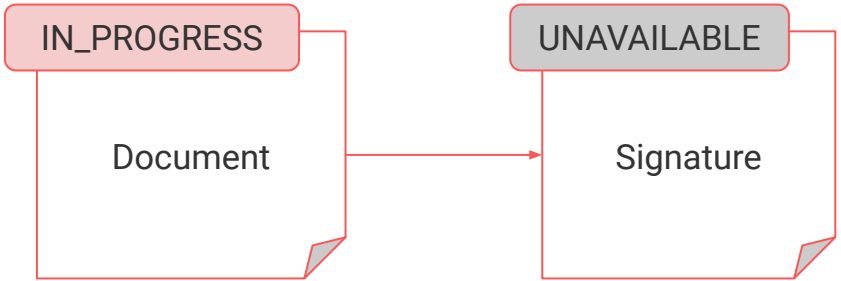


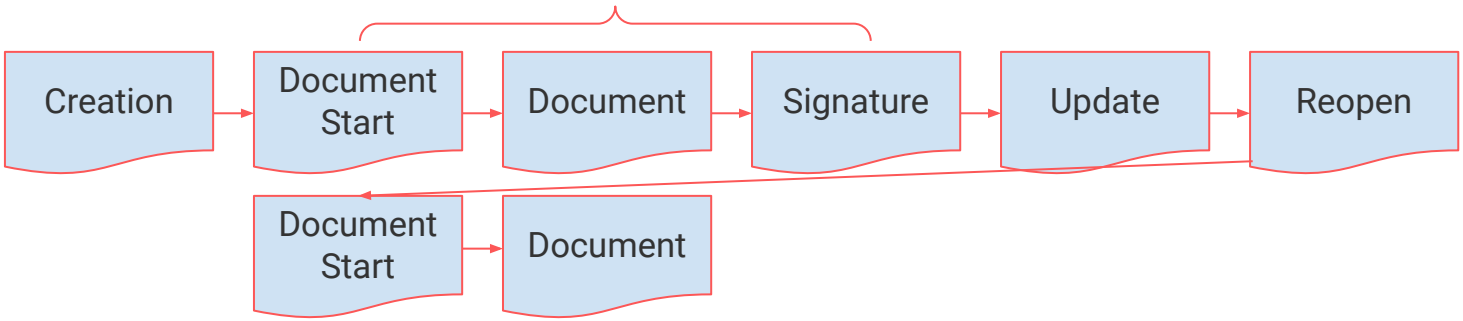
Participant



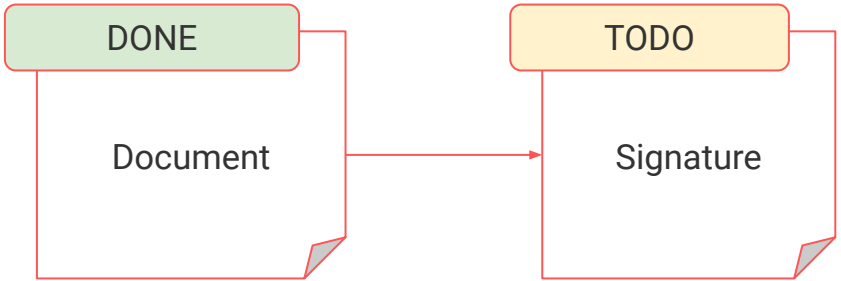


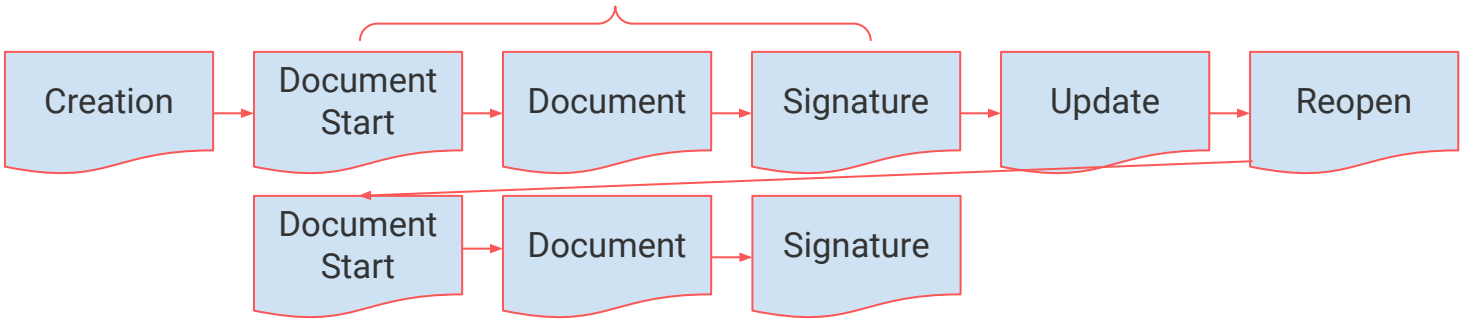
Participant



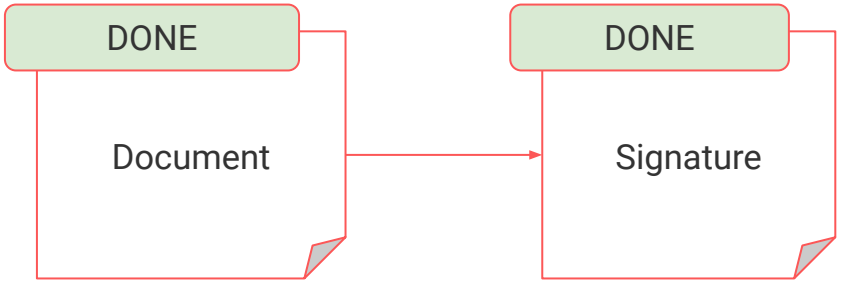


Participant

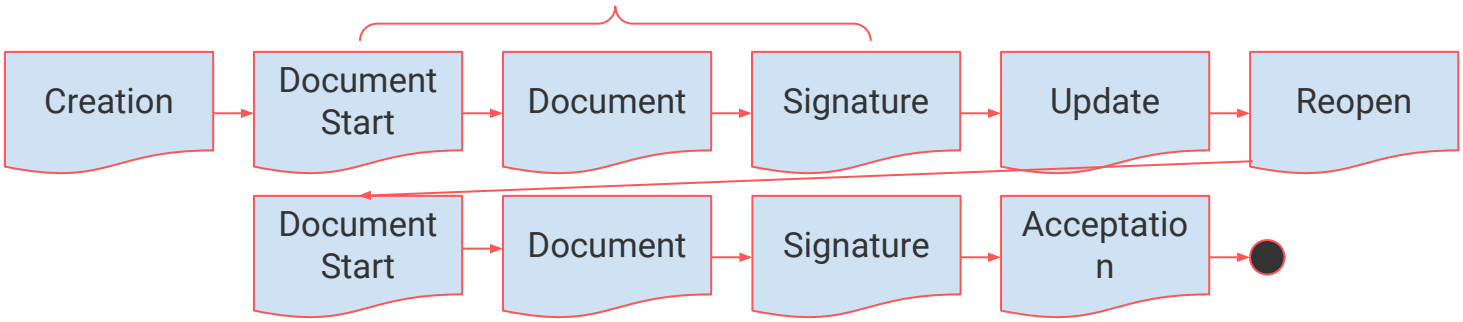




Participant

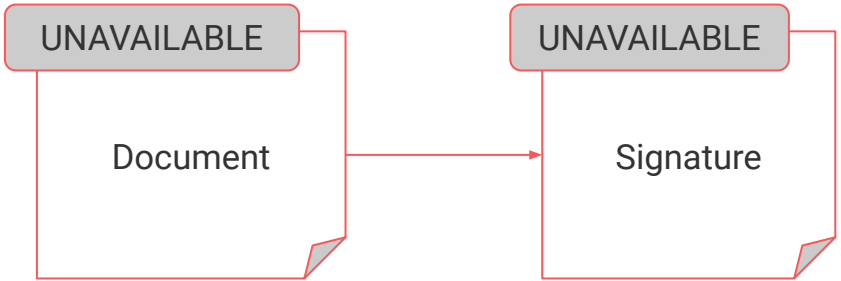


Opérateur



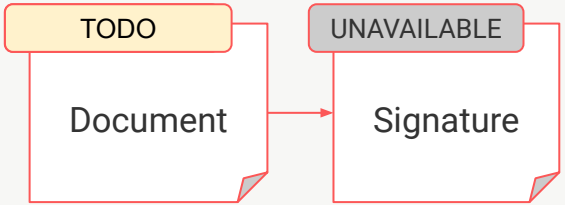
Participant

Opérateur



task.state

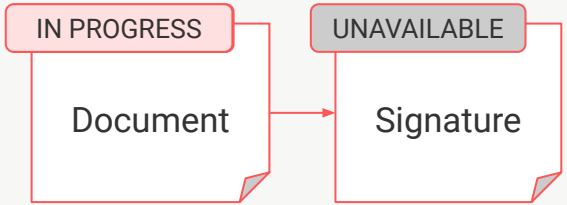
Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {

                tasks
                    .forEach { it.state = Task.State.UNAVAILABLE }
                tasks
                    .documentTasks()
                    .forEach { it.state = Task.State.TODO }
            }
            is DocumentStartEvent -> {
            is DocumentEvent -> {
            is SignatureEvent -> {
            is ReopenEvent -> {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
        }
    }
}
```

task.state

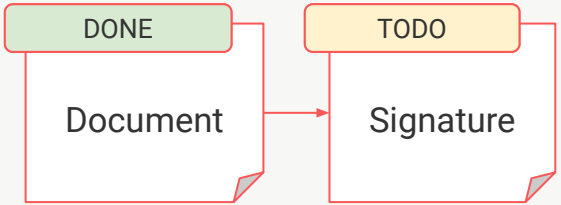
Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {
            is DocumentStartEvent -> {

                taskById(event.taskId)
                    ?.state = Task.State.IN_PROGRESS
            }
            is DocumentEvent -> {
            is SignatureEvent -> {
            is ReopenEvent -> {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
        }
    }
}
```

task.state

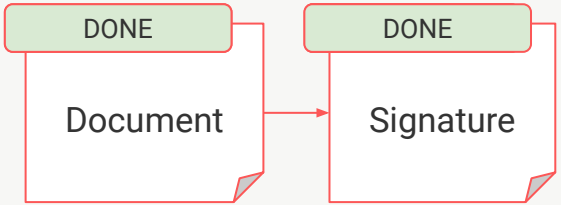
Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {
            is DocumentStartEvent -> {
            is DocumentEvent -> {

                taskById(event.taskId)
                    ?.state = Task.State.DONE

                tasks
                    .signatureTasks()
                    .forEach { it.state = Task.State.TODO }
            }
            is SignatureEvent -> {
            is ReopenEvent -> {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
        }
    }
}
```

task.state

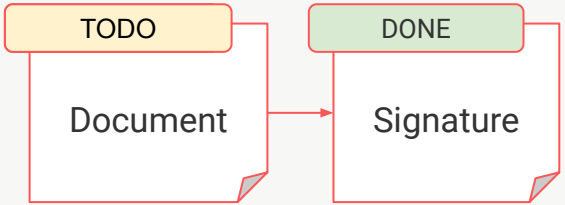
Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {
            is DocumentStartEvent -> {
            is DocumentEvent -> {
            is SignatureEvent -> {

                taskById(event.taskId)
                    ?.state = Task.State.DONE
            }
            is ReopenEvent -> {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
        }
    }
}
```

task.state

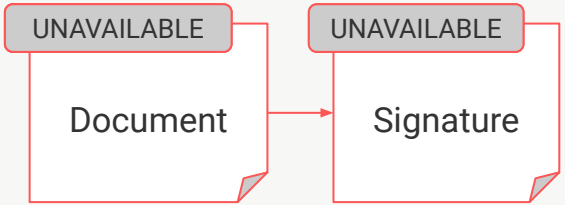
Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {
            is DocumentStartEvent -> {
            is DocumentEvent -> {
            is SignatureEvent -> {
            is ReopenEvent -> {

                val toReopened = event
                    .reopenedTasksId
                    .mapNotNull { taskById(it) }
                when {
                    toReopened.documentTasks().isNotEmpty() -> {
                        toReopened
                            .documentTasks()
                            .forEach { it.state = Task.State.TODO }
                    }
                    else -> // ...
            }
            is AcceptationEvent,
            is AutoAcceptationEvent -> {
```

task.state

Event Sourcing




```
data class ClientFile private constructor() {
    //...

    private fun applyTaskState(event: ClientFileEvent) {
        when (event) {
            is CreationEvent -> {
            is DocumentStartEvent -> {
            is DocumentEvent -> {
            is SignatureEvent -> {
            is ReopenEvent -> {
            is AcceptationEvent,
            is AutoAcceptationEvent -> {

                tasks
                    .forEach { it.state = Task.State.UNAVAILABLE }
            }
        }
    }
}
```

🟢 Tests et non régression

- 📝 Logger tous les changements
- 🪙 Golden Master
- 🔄 Feature flag

🚀 Mise en production

- 🔄 Nouvelles features, clients
- ⏱️ Prendre son temps mais aller vite
- 📈 Suivi de l’adoption
- 🧹 Supprimer la V1

👍 Quelques conseils

✅ Des Events complets et simples

✅ Rendre les events private

⚠️ Protéger les accès concurrents

❌ Re-parcourir les events dans le moteur Event Sourcing

❌ Mixer les états finaux et évènementiel

😬 Evolution des Events

- ✌️ Compatibilité
- 🪓 Migration

malt

Feedback

tgirard12/legacy-to-event-sourcing