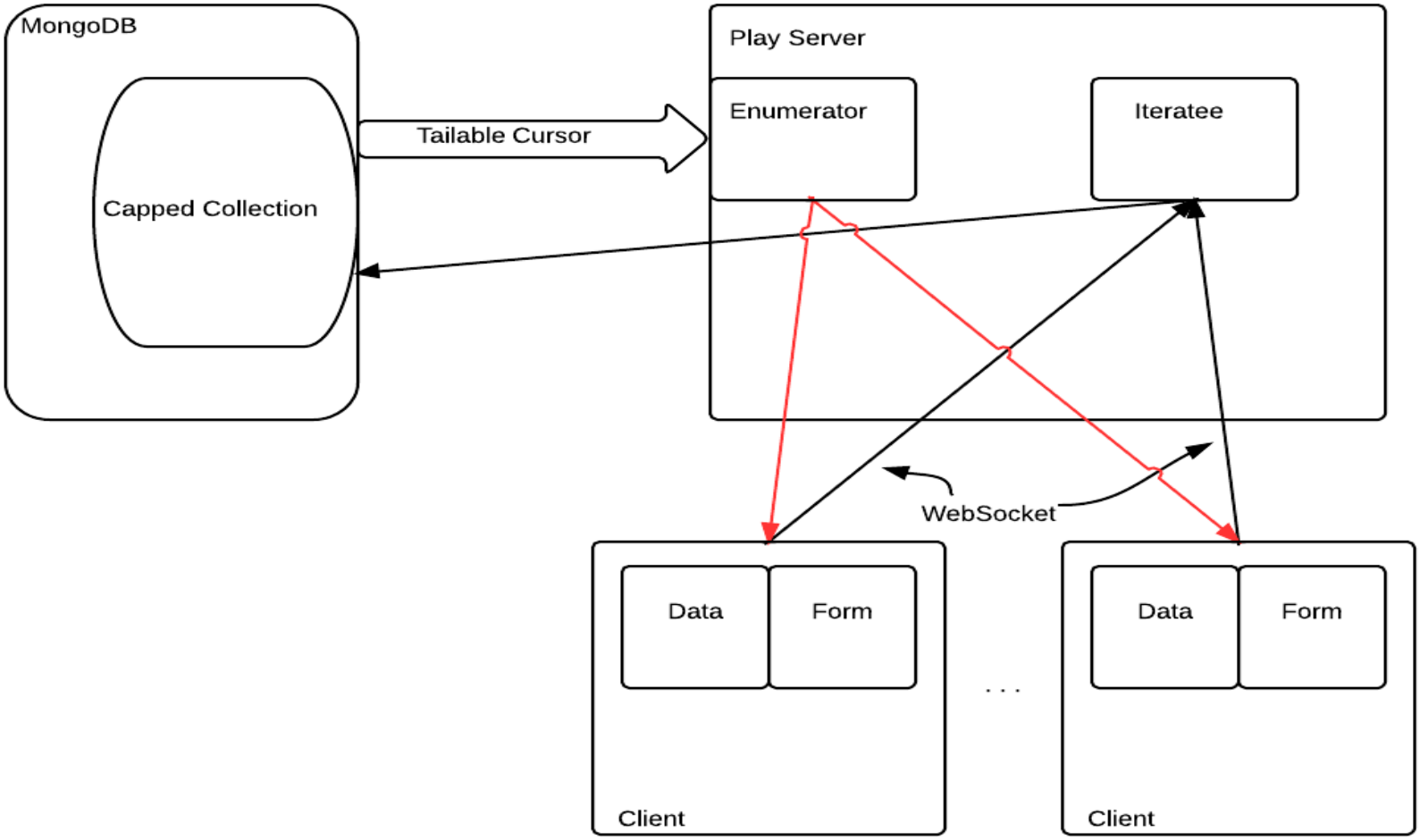
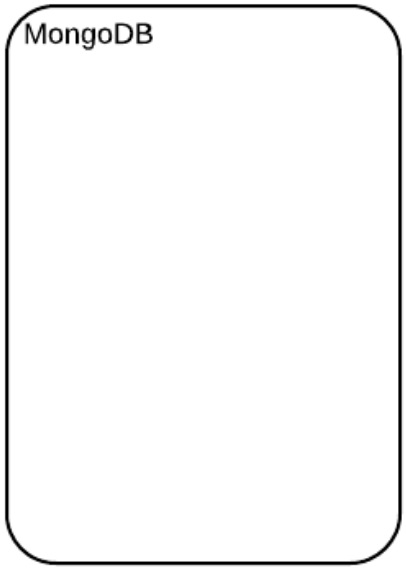
Play 2, ReactiveMongo ve MongoDB ile Web Uygulaması Geliştirme

Mehmet Ali Gözaydın

Türkiye Scala Kullanıcıları Topluluğu

Uygulama Yapısı:




MongoDB

MongoDB

Instance oluşturma:

- *$mkdir sampleappdb*
- *$user/bin/mongod --dbpath sampleappdb/*

veya servisi kullanmak için:

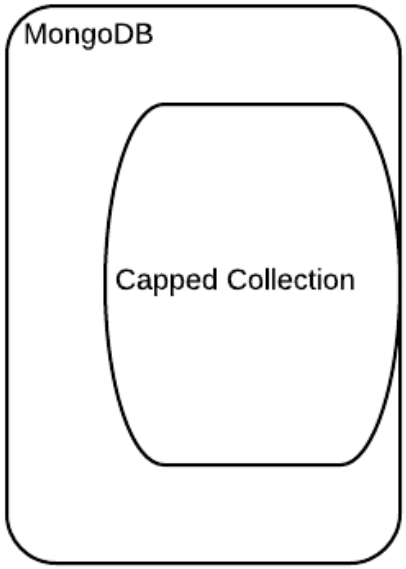
- *$sudo service mongodb start*

Mongo shell'e bağlanma:

- *$mongo*
-

Veritabanı oluşturma:

- *>use sampleappdb*

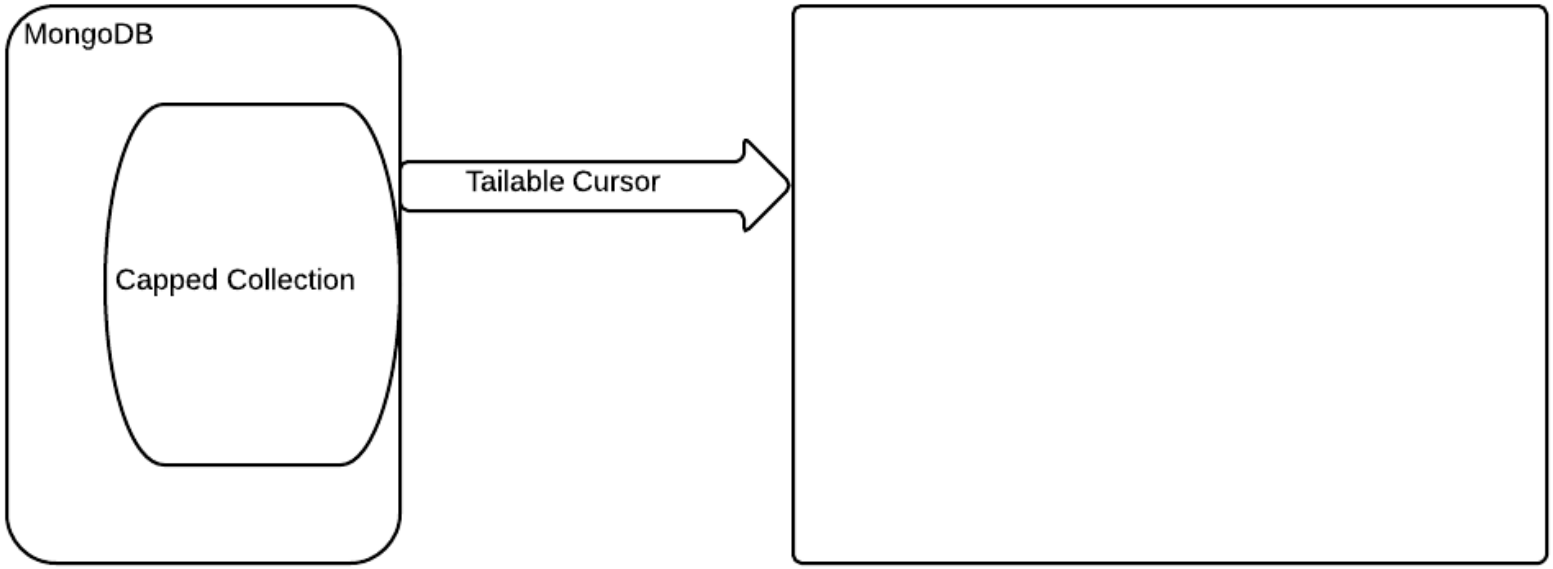

MongoDB
Capped Collection

Capped Collections

- Index alanları yoktur.
- Dokümanlar kayıt sıralarına göre yönetilir.
- Sınırlı bir alanları vardır.
- Dolunca en eski dokümanın üstüne yazar.
- Çok yazma işi olan durumlarda kullanılır(örn: log)

Capped Collection oluşturmak için

- *>db.createCollection("acappedcollection", {capped:true, size:100000})*

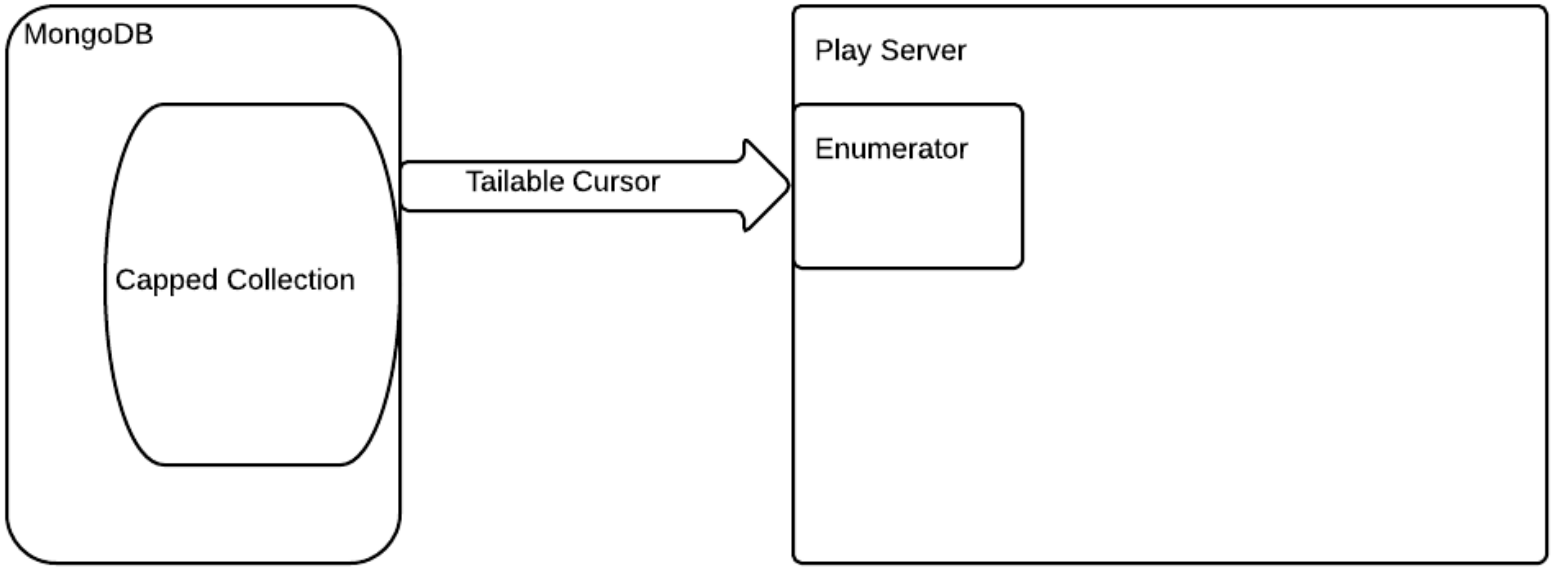

MongoDB
Capped Collection
Tailable Cursor

Tailable Cursors

Cursor:

- Bir sorgunun sonuç setini gösteren pointer.
- Tüm veriler işlendikten sonra otomatik kapanır.

- Sorgu sonuçlandıktan sonra tailable cursor kapanmaz.
- Index bilgisi kullanmaz. Bu yüzden initial sorgu maliyetlidir.
- Fakat sonrasında yeni eklenen dokümanları alması masrafsızdır.


MongoDB
Capped Collection
Tailable Cursor
Play Server
Enumerator

Play2 Framework

- MongoDB sürücüsü ReactiveMongo'dur.
- ReactiveMongo sayesinde tüm veritabanı işlemleri asenkron/non-blocking olarak yürütülür.
- Bu işi Iterateeler ve Future valuelar sayesinde yapar.

Iteratees

Özetle:

Enumerator (produce data) → Enumeratee (map data) → Iteratee (consume data)

- Veritabanı sorgusu bir Enumerator olabilir.
- Iteratee Enumeratorda veri beklerken bloklanmaz.
- Enumerator'un bir stream olabilmesi sayesinde tailable cursorları bağlayabiliyoruz.

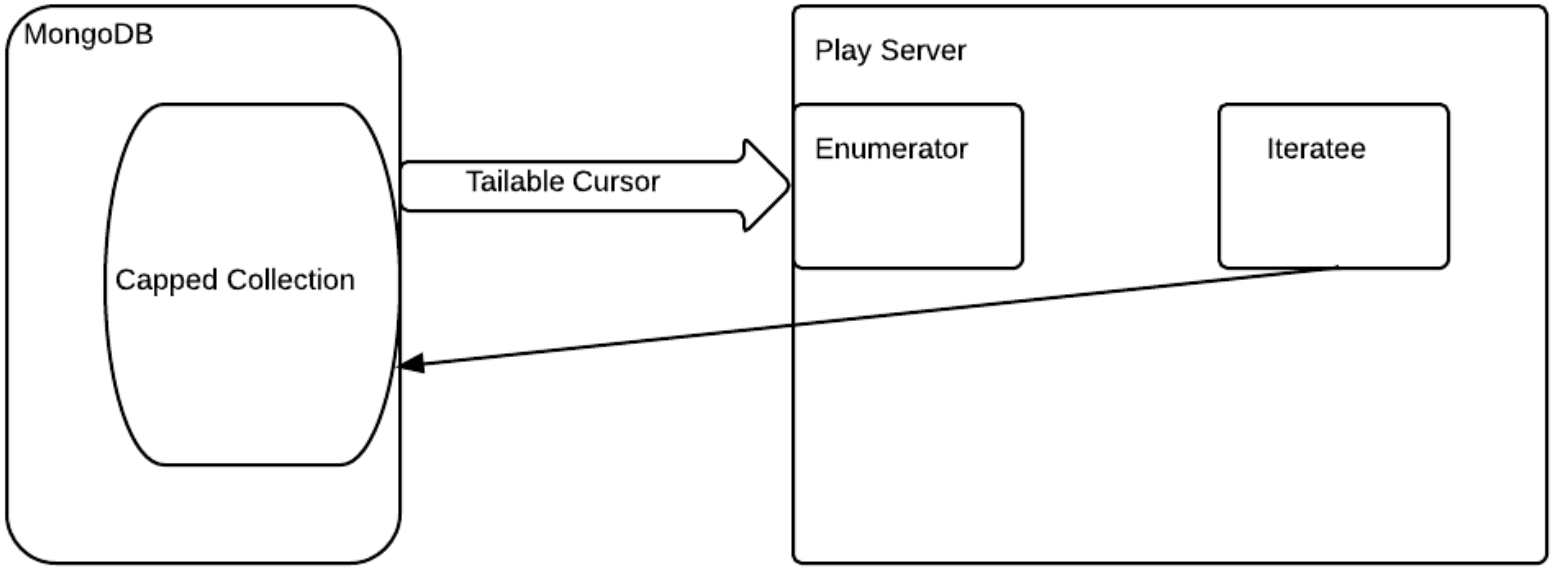
Future

- Gelecekte elimizde olacağını varsaydığımız veri.

```
lazy val futureCollection :Future[Collection] = {
    val db = ReactiveMongoPlugin.db
    val collection = db.collection("acappedcollection")
    Future(collection)
}
```

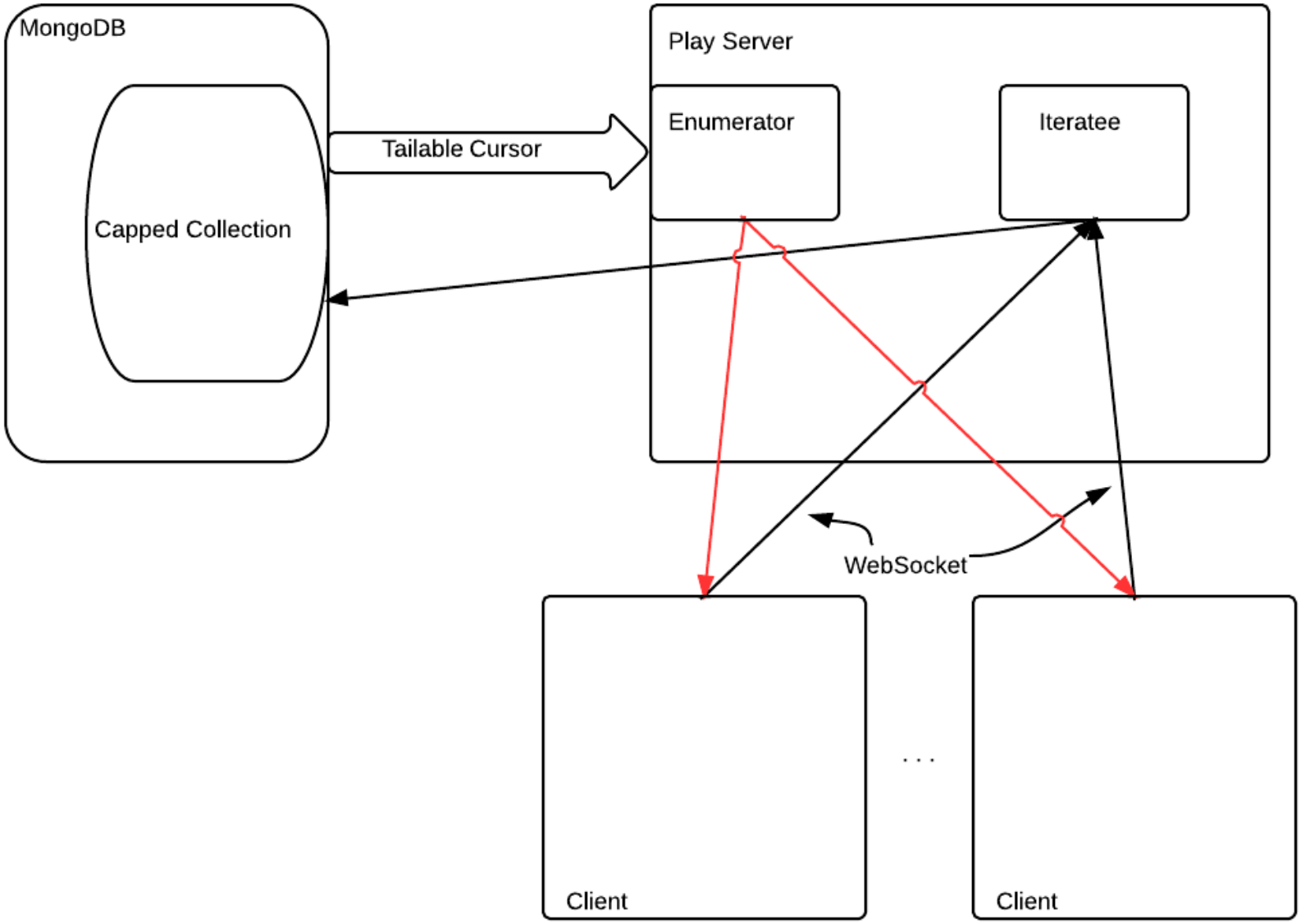
Enumerator

```
// veritabanına girdi oluştuğunda
val out = {
  val futureEnumerator = futureCollection.map {collection
=>
    val cursor =
        collection.find(Json.obj(),
        QueryOpts().tailable.awaitData)
        cursor.enumerate
  }
Enumerator.flatten(futureEnumerator)
}
```


MongoDB
Capped Collection
Tailable Cursor
Play Server
Enumerator
Iteratee

Iteratee

```
val in = Iteratee.flatten(
      futureCollection.map(
        collection => Iteratee.foreach[JsValue] { json =>
          println("received " + json)
          collection.insert(json)
        }
      )
    )
```


MongoDB
Capped Collection
Tailable Cursor
Play Server
Enumerator
Iteratee
WebSocket
Client
. . .
Client

Websocket

```
def watchCollection = WebSocket.using[JsValue] { request
=>

    // WS'den gelen mesajı veritabanına yazar
    val in = ...

    // veritabanına girdi oluştuğunda
    val out = ...
    // We're done!
    (in, out)
  }
```

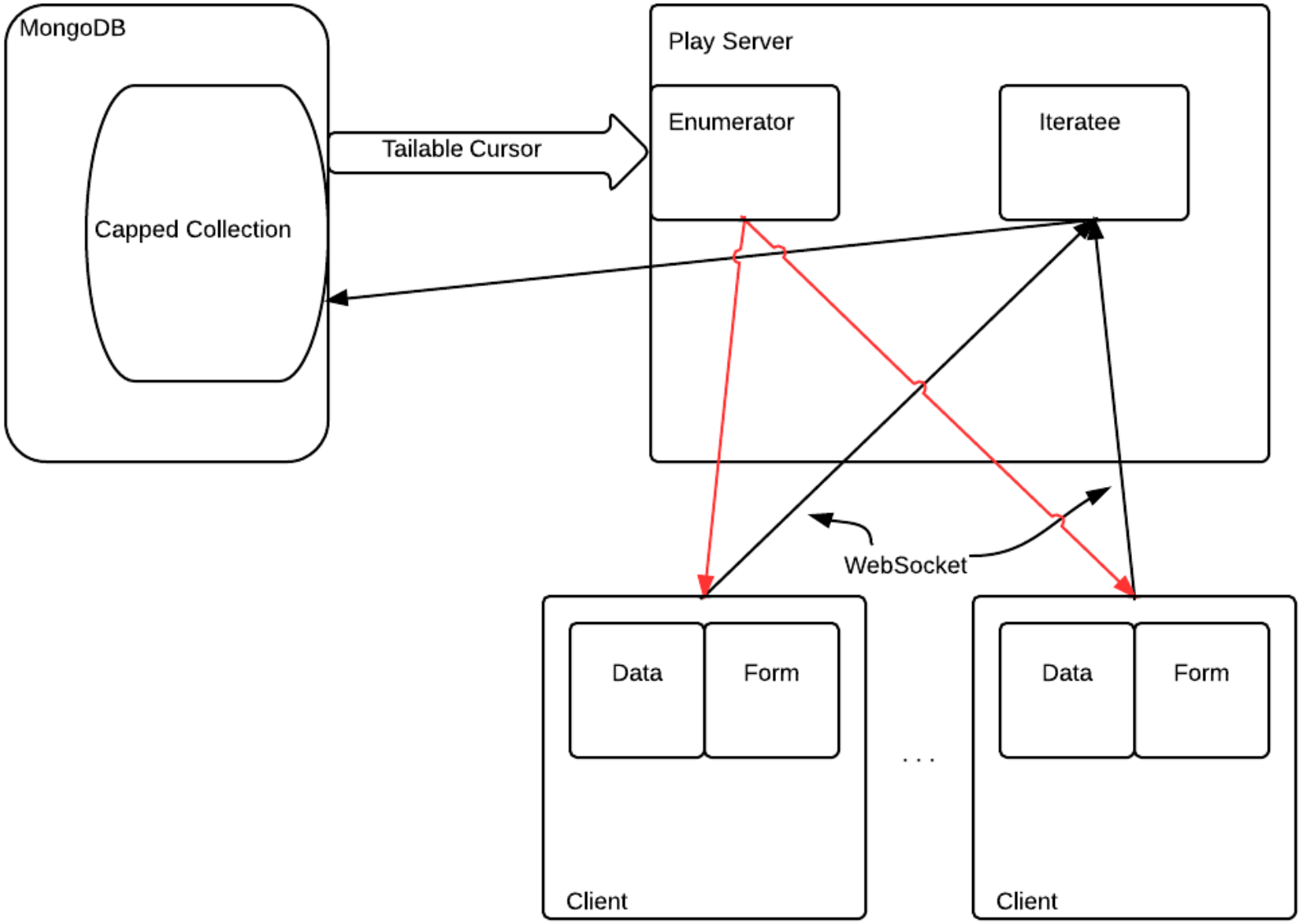
Websocket

Route:

```
GET   /watchCollection   controllers.Application.watchCollection
```

Uri:

```
ws://localhost:9000/watchCollection
```


MongoDB
Capped Collection
Tailable Cursor
Play Server
Enumerator
Iteratee
WebSocket
Data
Form
Client
Data
Form
Client

Bağlantılar

http://www.playframework.com/
http://www.mongodb.org/
http://reactivemongo.org/

Uygulama:
https://github.com/kubudi/oylg2013-demoApp

İletişim:

- kubudik@4Primes.com
- @kubudik
- github.com/kubudi
- kubudi.me