Using MongoDB at x.ai

x.ai a personal assistant
who schedules meetings for you

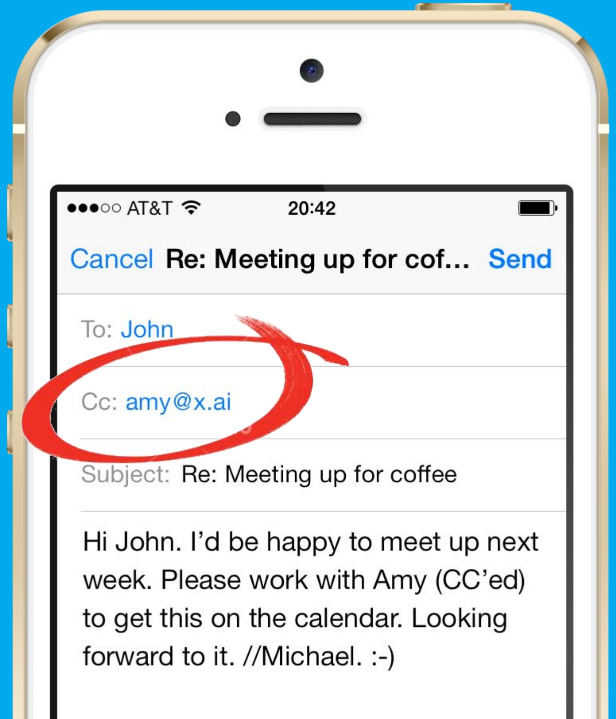
matt casey
@xdotai

MONGO DAY 2015 SV

VISIT X.AI TO JOIN THE WAITLIST

Magically
Schedule Meetings

Just Cc: Amy@x.ai



Re: Meeting up for cof...

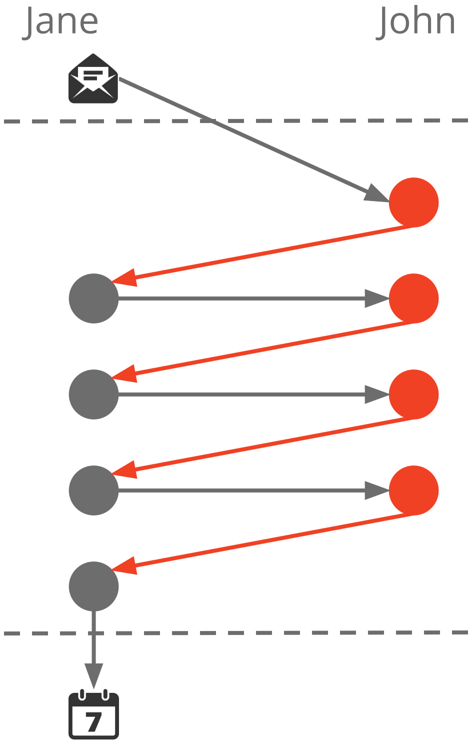
To: John

Cc: amy@x.ai

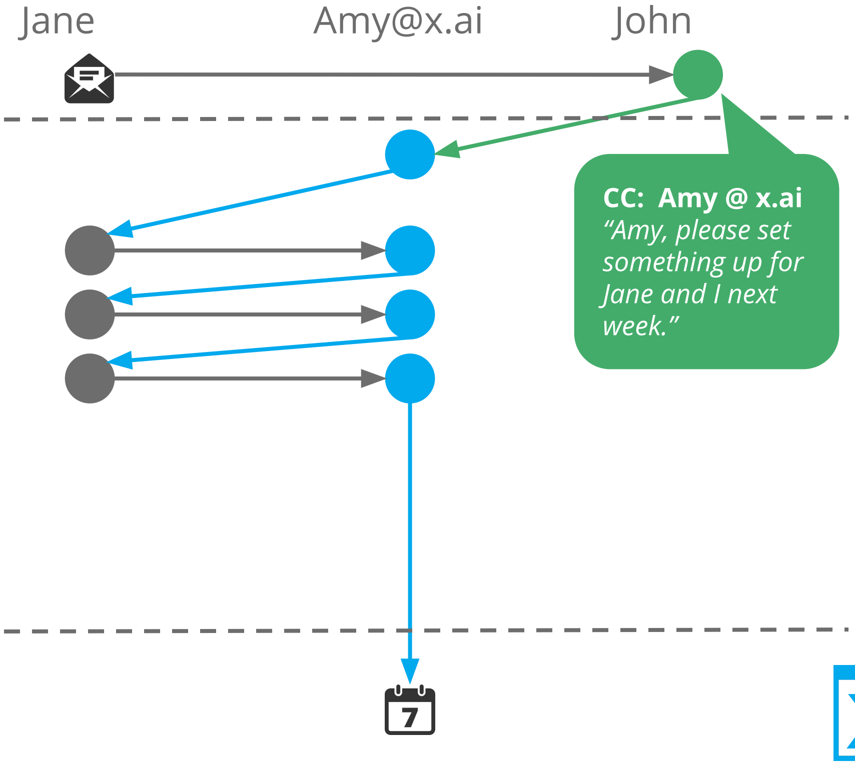
Subject: Re: Meeting up for coffee

Hi John. I’d be happy to meet up next week. Please work with Amy (CC’ed) to get this on the calendar. Looking forward to it. //Michael. :-)

Pain



Solution



CC: Amy @ x.ai
"Amy, please set something up for Jane and I next week."

System characteristics

- Need quick response
- Learning algorithms require large training data set
- # meetings scale linearly with # users
- 1 user meets with N people
- People share meeting, places, times and company
- Social relationships (assistants, coordinators)

Technical challenges

- Natural language understanding with extremely high accuracy
- Natural conversation over email with people
- Complex data relationship
- Optimize for sparse data
- Speed of development and change

Stack

New Relic.

Scala

node JS™

mongoDB

libratoᵀᴹ

amazon web services

Spark

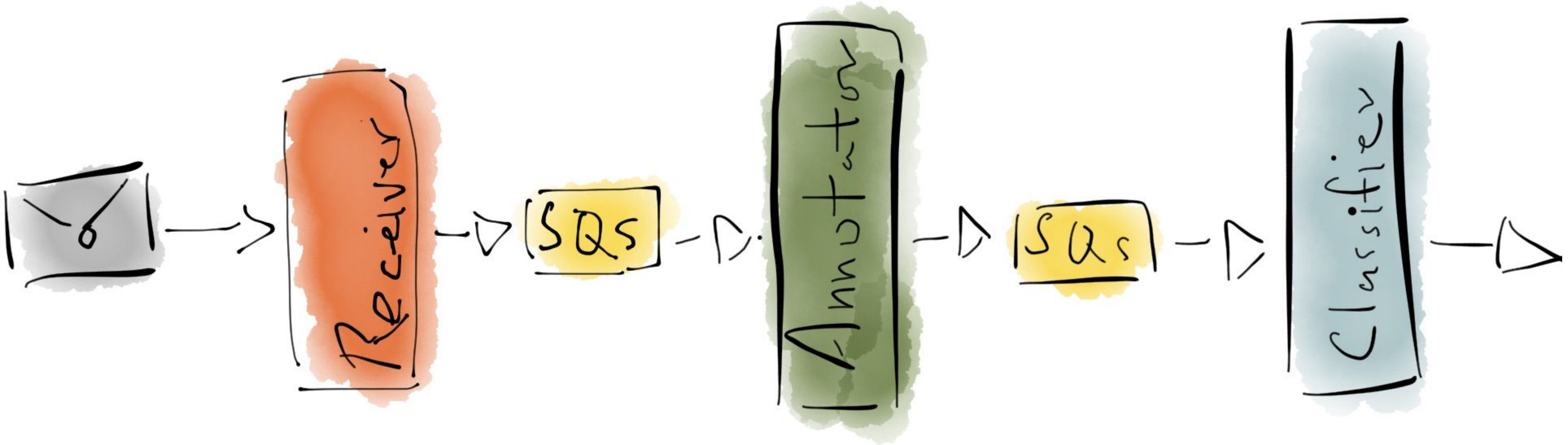
circleci

ANGULARJS by Google

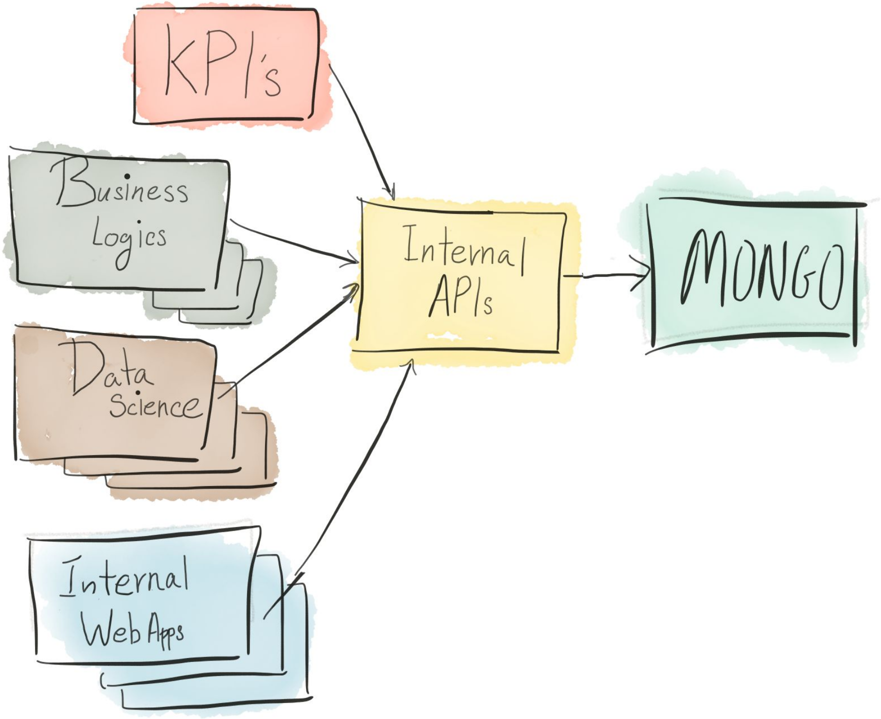
loggly

X.

Queue Based Architecture



Data Access



- Schema Standardization
- Mongoose
- Supported by Elastic Load Balancer

Data Models: Let’s Get Started

Here’s a few tips:

- Mongoose.js ODM for CRUD services
- mongoose-express-restify to provide a REST API
- MongoDB MMS for monitoring and backups
- Keep old data up-to-date with schema changes
- Don’t over-design, use models that are easy to change

How Do We Model Time?

Use Case:

- Capture times and constraints from an email for scheduling a meeting
- Data is created, saved, and read by a human

“Let’s meet the first week of February next year”

```
time: {
     start: new Date(2017, 1, 1),
     end: new Date(2017, 1, 2)
}
```

Try Again: Our Second Model

Use Case:

- A nested model captures the original intent
- Makes it possible for machines to detect and respond

```
timeV2: {
     within: {
          reference: {
               weekOfYear: 4,
               year: 2017
          }
     }
}
```

Still, some room for improvement

- Storing data closer to text makes it easier to improve accuracy
- But is harder to interpret and validate without tools

```
timeV3: [
     { timeId: id0, text: ‘first week’, weekOfMonth: 1 },
     { timeId: id0, text: ‘February’, month: 2 },
     { timeId: id0, text: ‘next year’, year: 2017 }
]
```

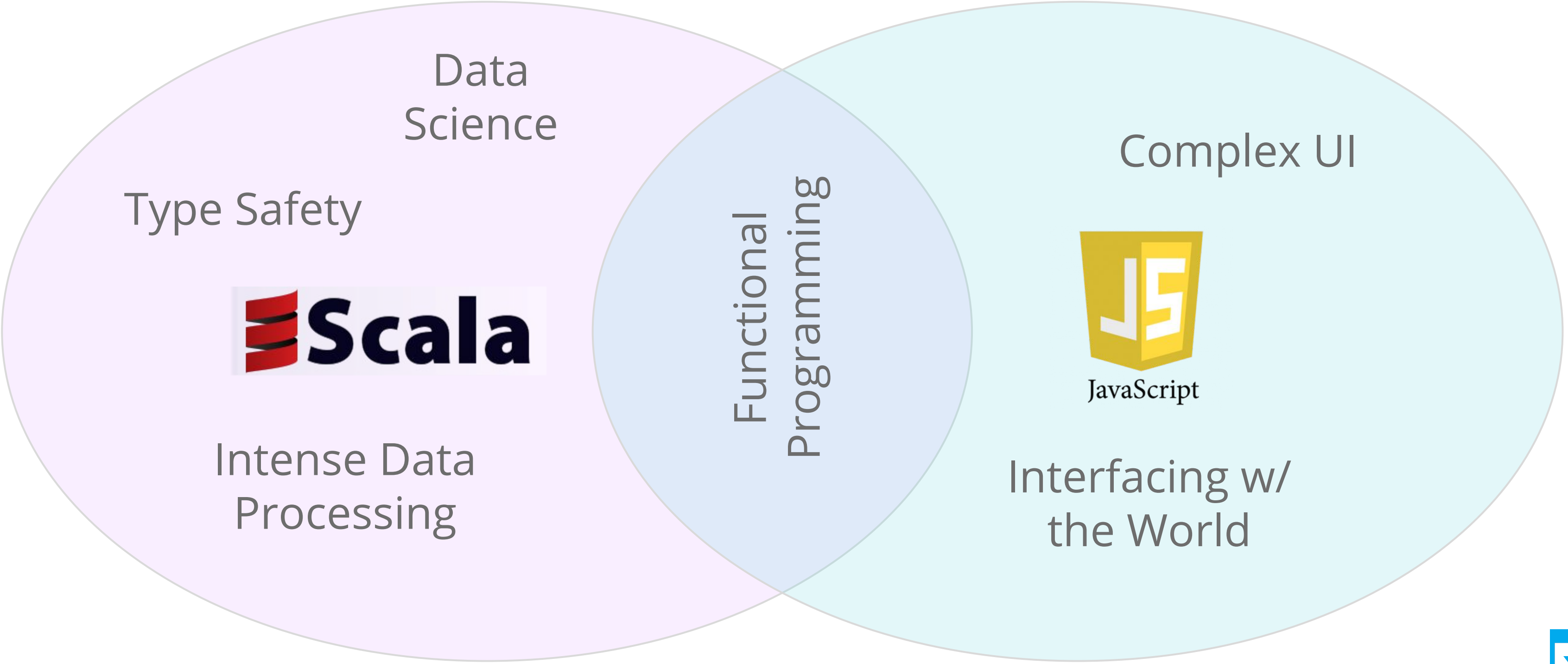
Edge cases dealing with time

- Scheduling phone calls for people in different timezones
- Dealing with partial information
  - ***“5pm is good”*** or ***“next week”***
- Irrelevant times
  - ***“Can’t wait to see you next month. Can we talk tomorrow?”***
- Need context for timezones: natural language vs tz database
  - Identify: **“IST,” “CST,” “EST”**
  - ***“I’m -5 from Dave”*** or ***“3pm London time”***
  - Etc/GMT+3 is not what you think

Going from human to machine

- Machine detects and predicts predicts times
- Human work shifts from feeding new information to correcting and re-affirming
- Focus shifts to generating more information and ensuring product quality

Scala & JS - Happily Ever After



Data Science

Type Safety

Intense Data Processing

Functional Programming

Complex UI

Interfacing w/ the World

Mongoose.js

object modeling and querying library for mongodb with node.js.

👍

- ODM
- Easy to set up, convenient
- Schema enforcement
- Use with caution: getters, setters, virtual fields
- Client side joins across collections and databases
- Nested documents with automatic _id reference

👎

- Applies to only one language
- `populate` mutates the object
- Functional Programming expects domain “events” vs objects
- Impossible to ‘clone’ instances
- using schema requires db connection
- No read validation - errors on updates

Mongo & Scala

Mongo

```
{
  "messageId" : "<somerandommessage@xxxx>",
  "classifierLabels" : [
    {
      "classifier" : "MeetingClassifier",
      "class" : "A"
    }
  ],
  "featureVector" : [
    {
      "featureType" : "TaggedOneGram",
      "featureKeys" : [
        {
          "featureKey" : "work",
          "featureValue" : 3
        },
        {
          "featureKey" : "that",
          "featureValue" : 1
        },
        ...
      ]
    },
    ...
  ]
}
```

Scala

```
case class Features (
       messageId: String,
       classes: List[Classes],
       featureVector: List[FeatureVectors]
)


case class Classes (
       classifier: String,
       class: String
)

case class FeatureVectors(
       featureType: String,
       featureKeys: List[FeatureKeys]
)

case class FeatureKey(
       featureKey: String,
       featureValue: 1
)
```

Schemaless vs. Typesafe

- Discrepancy between models in Scala versus stored data in mongo.
  - MongoDB stores free form nested structures
  - Scala relies on strict, type safed models for data
- Models in flux
  - Same format for both data extraction and decision making layers
  - We are continuously learning new edge case about scheduling meetings

Updating schemas in production

Step #1: Design new schema

- Avoid mutating the meaning of a field, add a new one intead

```
timeEntities => timeEntitiesV2
```

- Don’t stop writing to the old field until other processes are updated

Step #2: Migrate old documents, if possible, or use a ‘blacklisted’ field for training

Step #3: Transition all processes that read from schema to look for the new model

Design Tips:

- Avoid saving state to the db, keep the data as close to reality as possible.
- Separate collections for each service
- Future-proof your model, a list of objects is more extendable than a list of strings:

```
actions: [‘SENT’, ‘DELIVERED’]
actions: [{ timestamp, action: ‘SENT’ }, { timestamp, action ‘DELIVERED’ }]
```

Lessons Learned

- Let the data tell the story, keep track of what was said or happened
- Avoid foreign keys to “moving” documents
- Be cautious of too much buy-in to your tools
- Write tests from Day One
- Make backups!

matt @ human.x.ai

cto & founder

25 Broadway. 9th Floor
New York, 10004 NY

E: hello@human.x.ai
T: @xdotai

Visit x.ai to join the waitlist

Embedding vs. Referencing

Embedding

```
{
  host :
  {
    name : {.....},
    nicknames : [String],
    phones : [{Type: String}]
    primaryEmail : String,
    secondaryEmails : [String],
    title : String,
    signatures: [String],
    …...
  },
  travelTime : String,
  status : String,
  timezone : String,
  duration : Number,
  …...
}
```

Referencing

```
{
  host : Participant,
  travelTime : String,
  status : String,
  timezone : String,
  duration : Number,
  …...
}

Participant
  {
    name : {.....},
    nicknames : [String],
    phones : [{Type: String}]
    primaryEmail : String,
    secondaryEmails : [String],
    title : String,
    signatures: [String],
    …...
  },
```

Considerations

- Query patterns
- Access to embedded doc
- # references to a doc
- Application level join
- 1-way or 2-way referencing

Unsolved Schema issues

- Keeping schema changes in sync with Mongoose and Scala
- No libraries or tests to make sure the contracts don’t break
  - ProtoBuffers or Avro?
  - discriminator key
- Managing a ‘context collection’ where multiple copies of production are stored per email
- Deploying schema changes sometimes requires re-training a model
- Schema validation from Scala supercedes Mongoose