# Pam – Project Details

## Table of Contents

## Overview

This is an application to help create other mobile applications

### Users involved

1. Application developers (AD), which is us! (Professor Jose, Disha Goradia, Madhur Mehta, and Venkata Mallampati)
2. Developer User – DU, the developers who want to use our application to create mobile applications
3. End users – EU, these people will run the application developed by DU

### Brief Description

DUs are going to provide a schema and configuration for their application. These will be stored in MongoDB and be accessed while EU runs the application.

## Features supported:

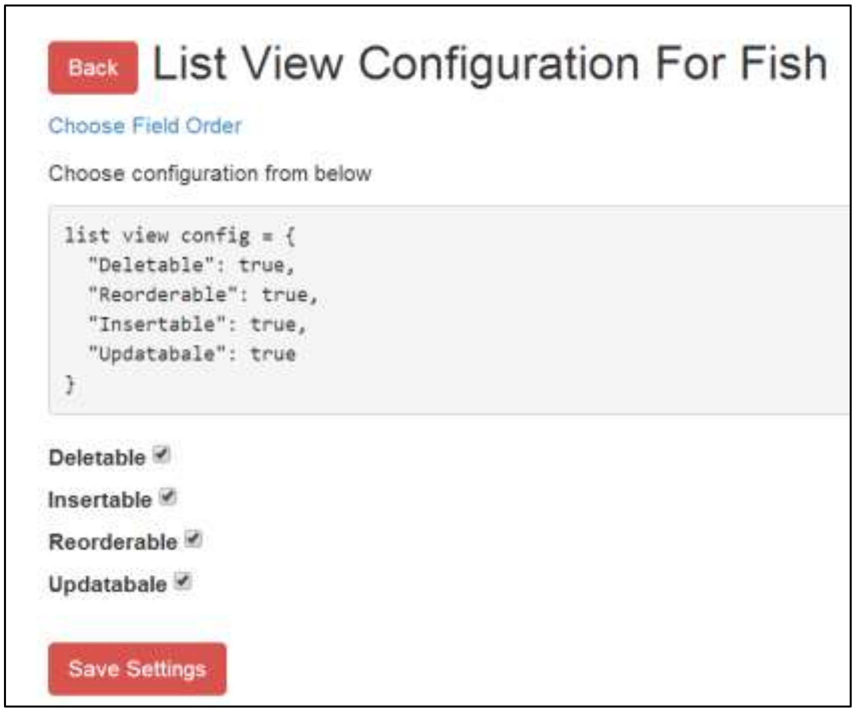
### Features for DU (developer users)

#### List view configuration for each entity participating in the application

a. **Update and save the configuration**

```
        list view config = {
          "Deletable": true/false/undefined,
          "Reorderable":true/false/undefined,
          "Insertable": true/false/undefined,
          "Updatabale": true/false/undefined
}
```

These settings are defining the *view behavior* for an end user while he/she is running the application



List View Configuration For Fish

Choose Field Order

Choose configuration from below

```
list view config = {
  "Deletable": true,
  "Reorderable": true,
  "Insertable": true,
  "Updatabale": true
}
```

Deletable ✓

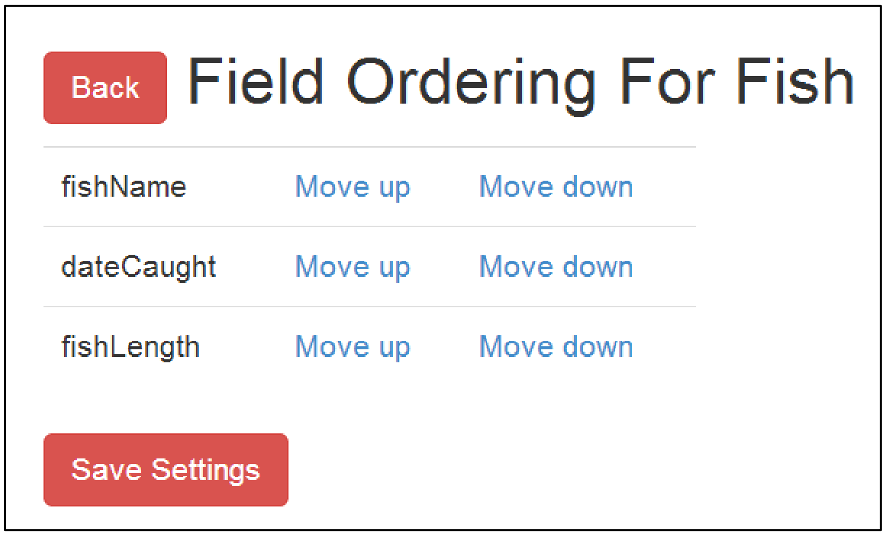
Insertable ✓

Reorderable ✓

Updatabale ✓

**b. Choose field order**

This defines the order in which the fields will be displayed for the entity while the end user runs the app



Field Ordering For Fish

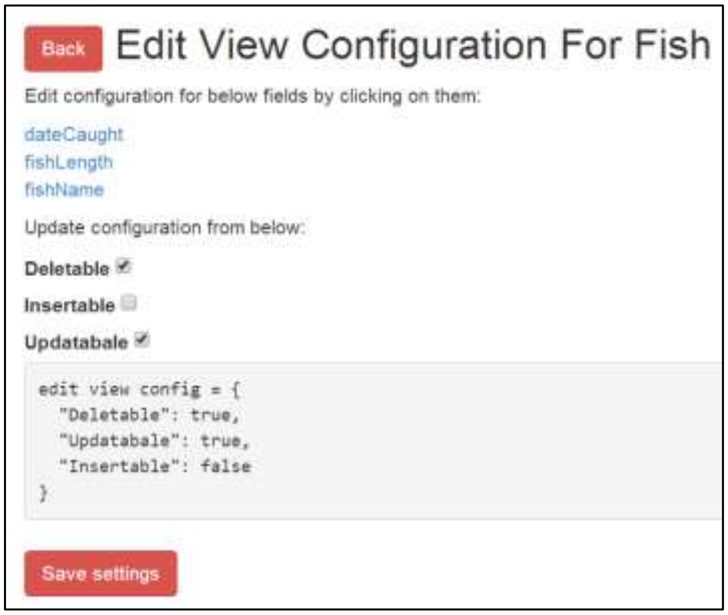
| | | |
|---|---|---|
| fishName | Move up | Move down |
| dateCaught | Move up | Move down |
| fishLength | Move up | Move down |

## Edit view configuration for each entity participating in the application

**a. Update and save the configuration**

```
        edit view config = {
         "Deletable": true/false/undefined,
          "Insertable": true/false/undefined,
}
```

These settings are defining the *edit behavior* for an end user while he/she is running the application to edit a particular document



Edit View Configuration For Fish

Edit configuration for below fields by clicking on them:

dateCaught
fishLength
fishName

Update configuration from below:

Deletable ✓

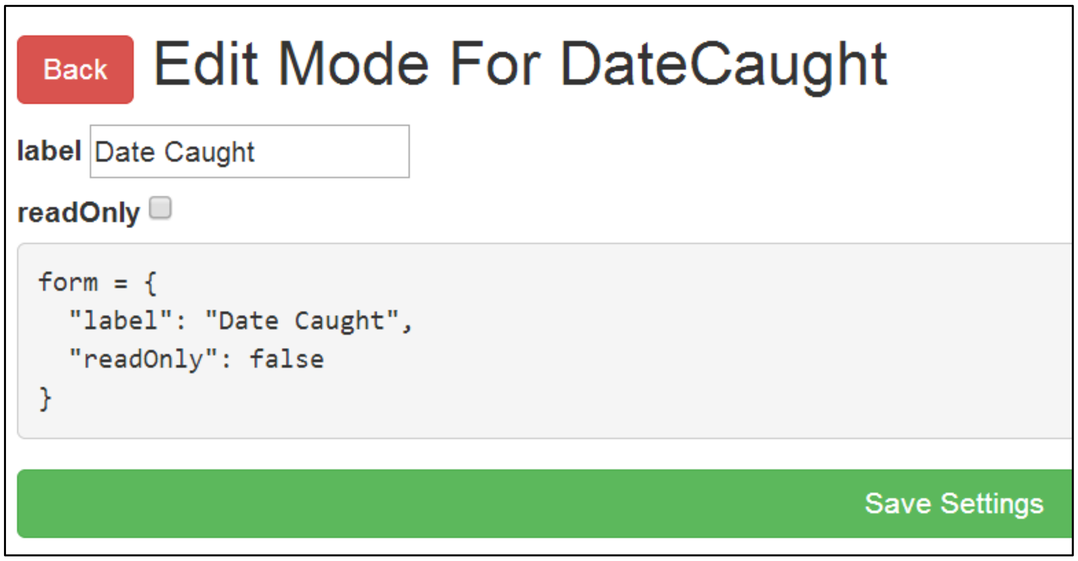
Insertable ☐

Updatabale ✓

```
edit view config = {
  "Deletable": true,
  "Updatabale": true,
  "Insertable": false
}
```

**b. Read only and label settings**

This allows the developer user on decide on a user friendly name for the fields and mark them as read only if the field should not be editable by end user (in the edit mode)



Edit Mode For DateCaught

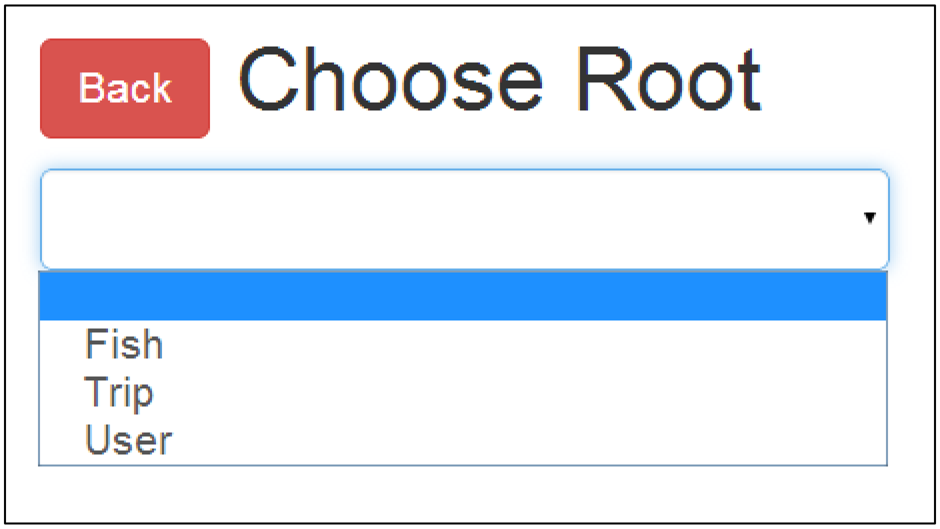
label Date Caught

readOnly ☐

```
form = {
  "label": "Date Caught",
  "readOnly": false
}
```

## Select root entity

For each application, the developer user chooses the root entity for the application, as he is aware of it



### Create services to post/get /update/delete the values

POST/GET/PUT/DELETE services that will be used by the end users while creating/updating/deleting entries. These services are developed by us (ADs). I had to re-implement these for the changed schema and appData structure, to support embedded structure

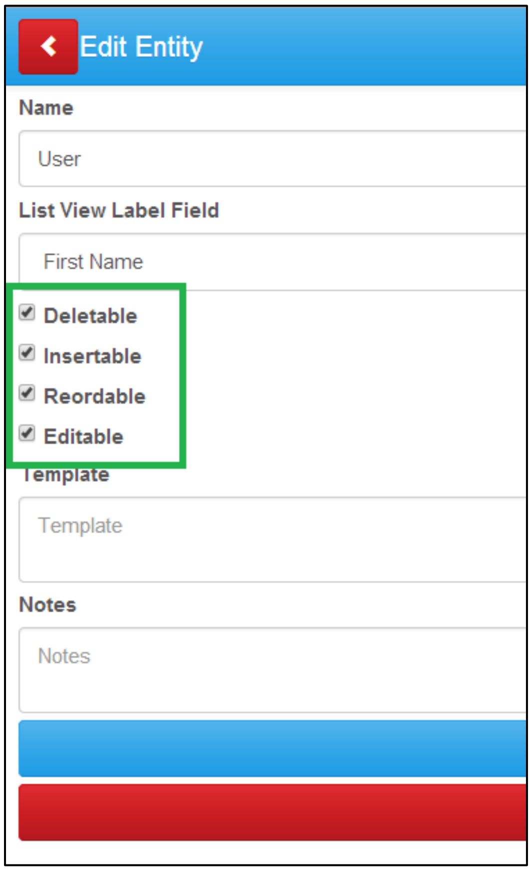
## Features for EU (end users)

1. Run the application
    a. Create objects
    b. View objects
    c. Edit objects
    d. Delete objects

With the above services, we enabled below use cases depending on the list and edit view configurations

## Screenshots for approach 2

### *Configurations*


Edit Entity

Name

User

List View Label Field

First Name

- [x] Deletable
- [x] Insertable
- [x] Reordable
- [x] Editable

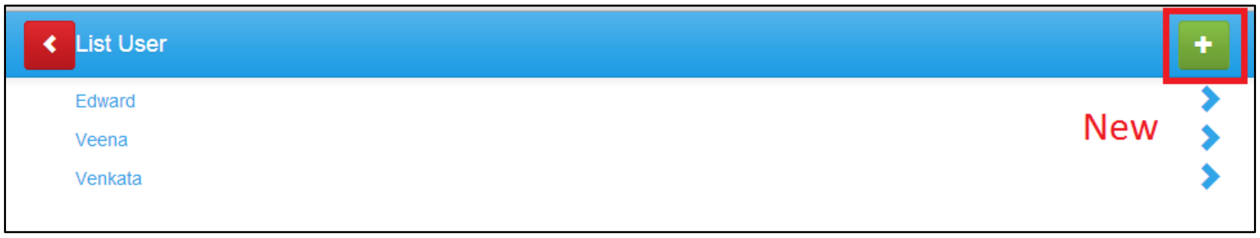
Template

Template

Notes

Notes

### *New*

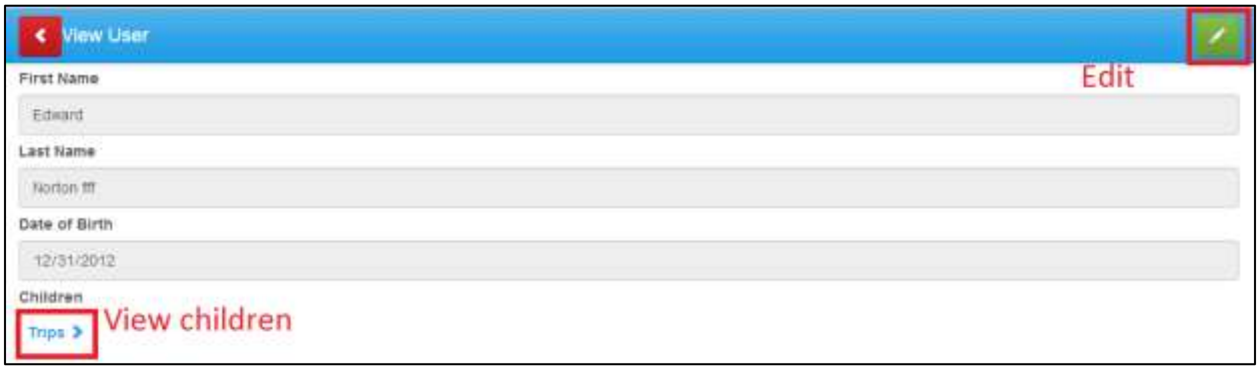


Edward

Veena

Venkata

### *Edit*



First Name

Edward

Last Name

Norton III

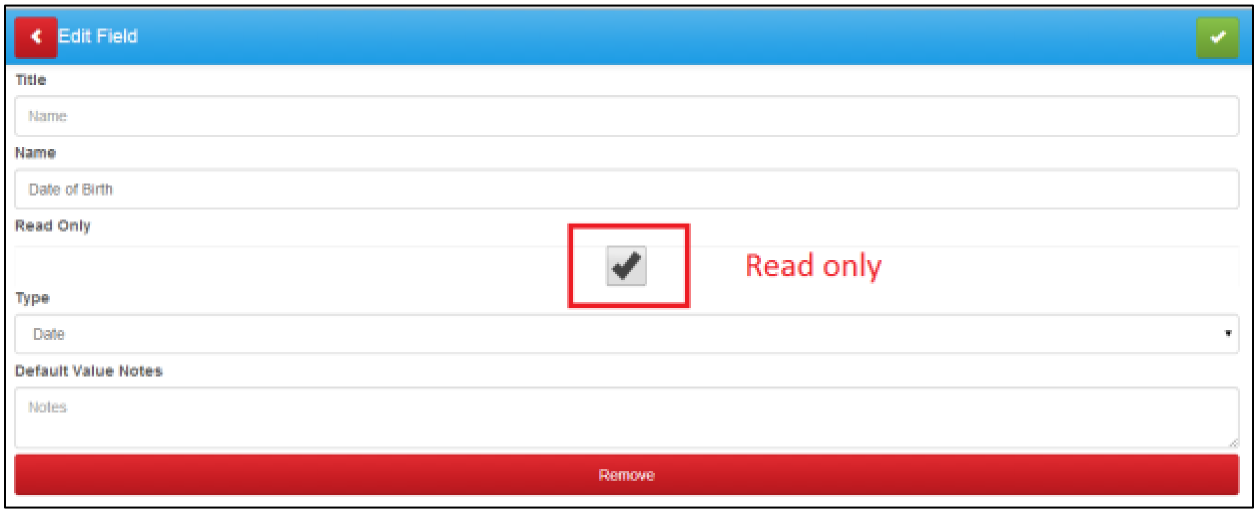
Date of Birth

12/31/2012

Children

Trips

## *Read only fields and Edit View*



Edit Field

Title

Name

Name

Date of Birth

Read Only

✓

Read only

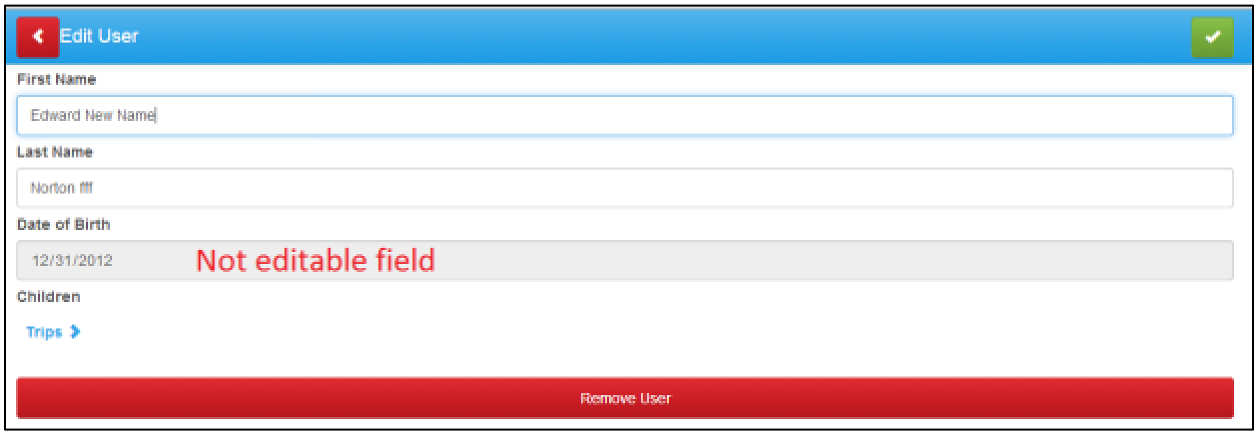
Type

Date

Default Value Notes

Notes

Remove



Edit User

First Name

Edward New Name

Last Name

Norton fff

Date of Birth

12/31/2012

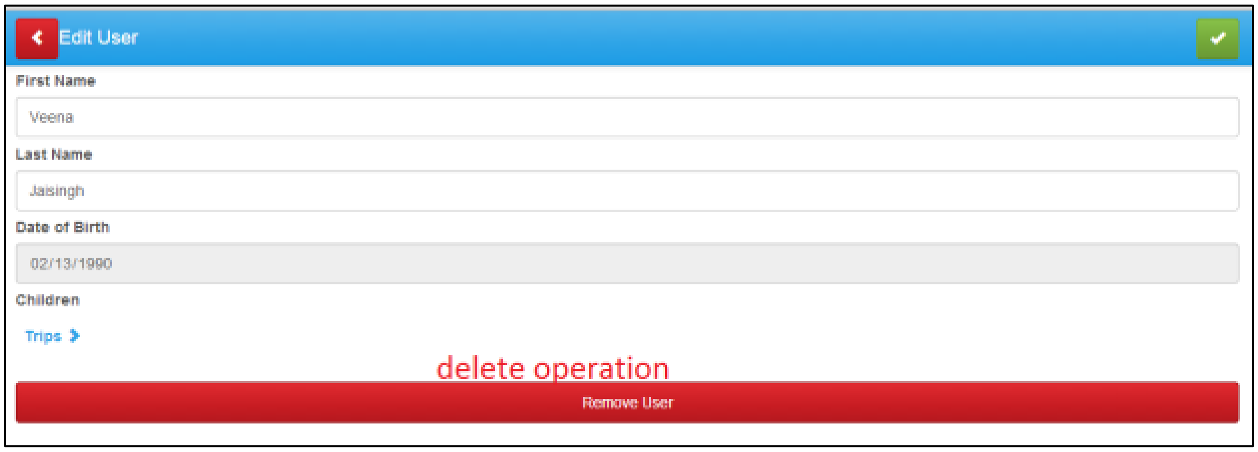
Not editable field

Children

Trips

Remove User

## *Delete entry*



Edit User

First Name

Veena

Last Name

Jaisingh

Date of Birth

02/13/1990

Children

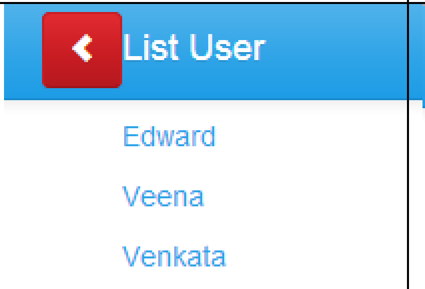
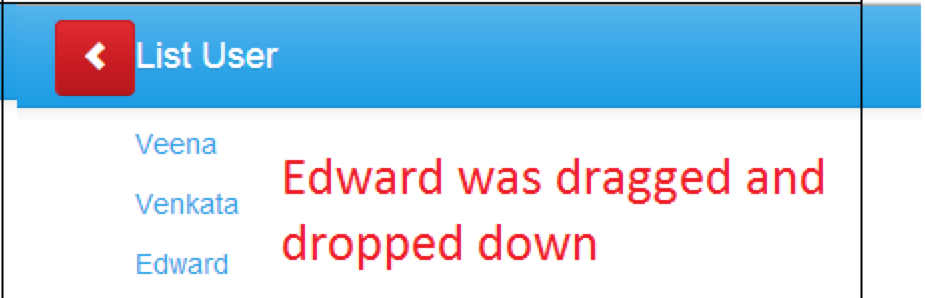
Trips

delete operation

Remove User

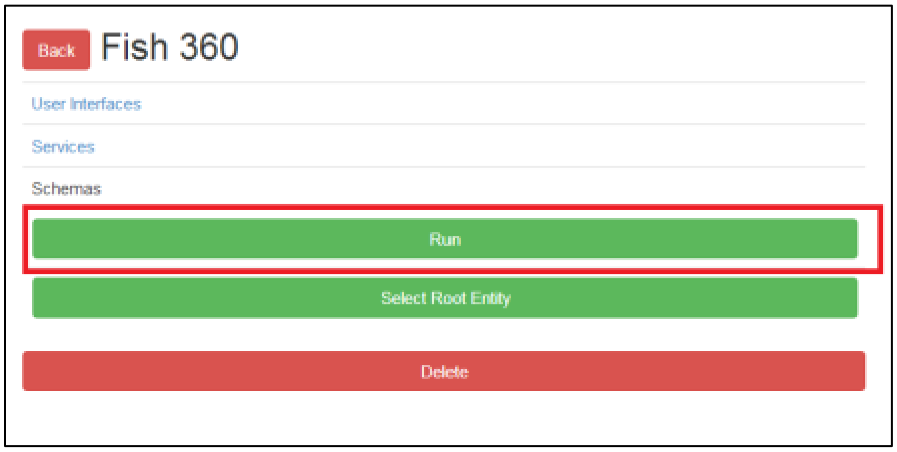
## *Re-orderable*

The rows can be dragged and dropped. The issue here is the order is not saved, that must be stored, and else the feature isn't of any use

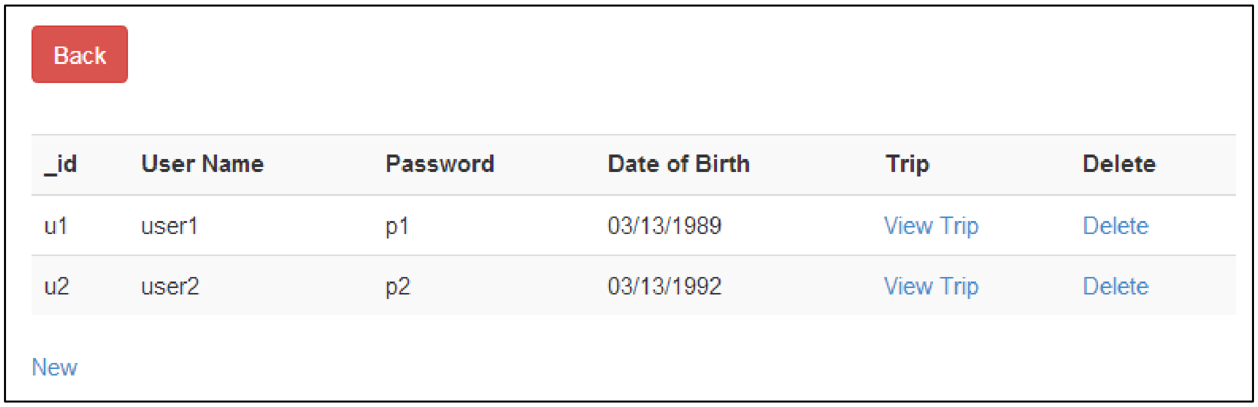
| Before | After dragging and dropping |
|---|---|
|  |  |

## Screenshots for approach 1

Run application:

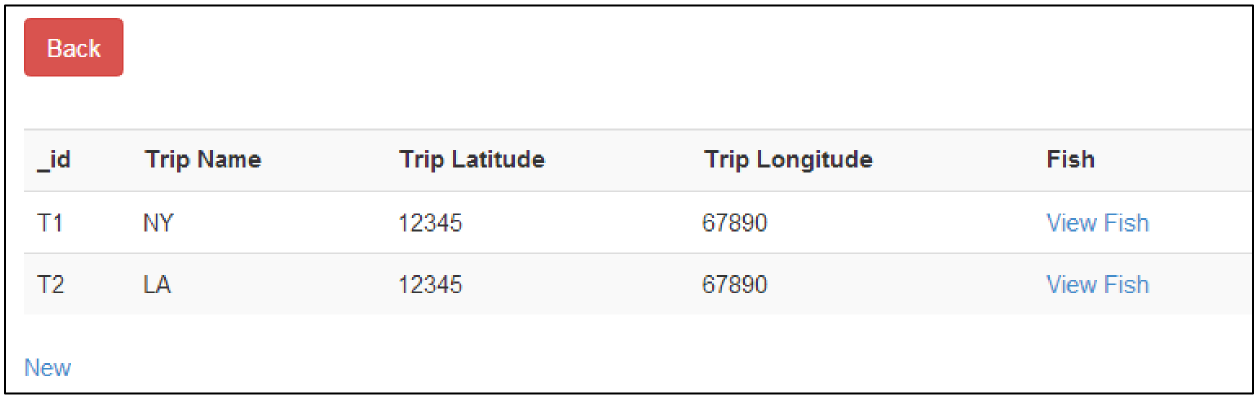


View root results:



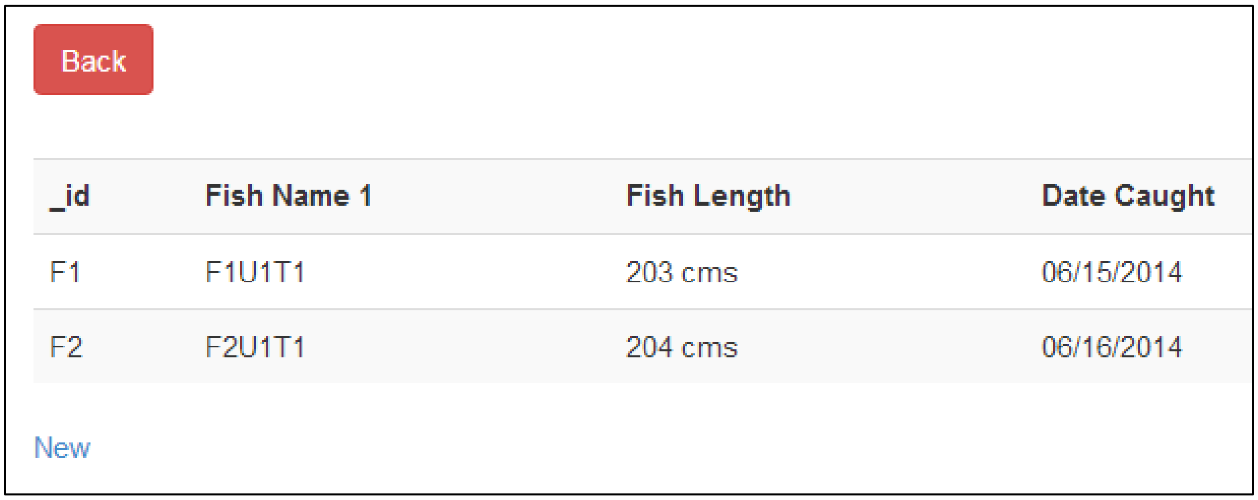
| _id | User Name | Password | Date of Birth | Trip | Delete |
|---|---|---|---|---|---|
| u1 | user1 | p1 | 03/13/1989 | View Trip | Delete |
| u2 | user2 | p2 | 03/13/1992 | View Trip | Delete |

View Trips:



| _id | Trip Name | Trip Latitude | Trip Longitude | Fish |
|---|---|---|---|---|
| T1 | NY | 12345 | 67890 | View Fish |
| T2 | LA | 12345 | 67890 | View Fish |

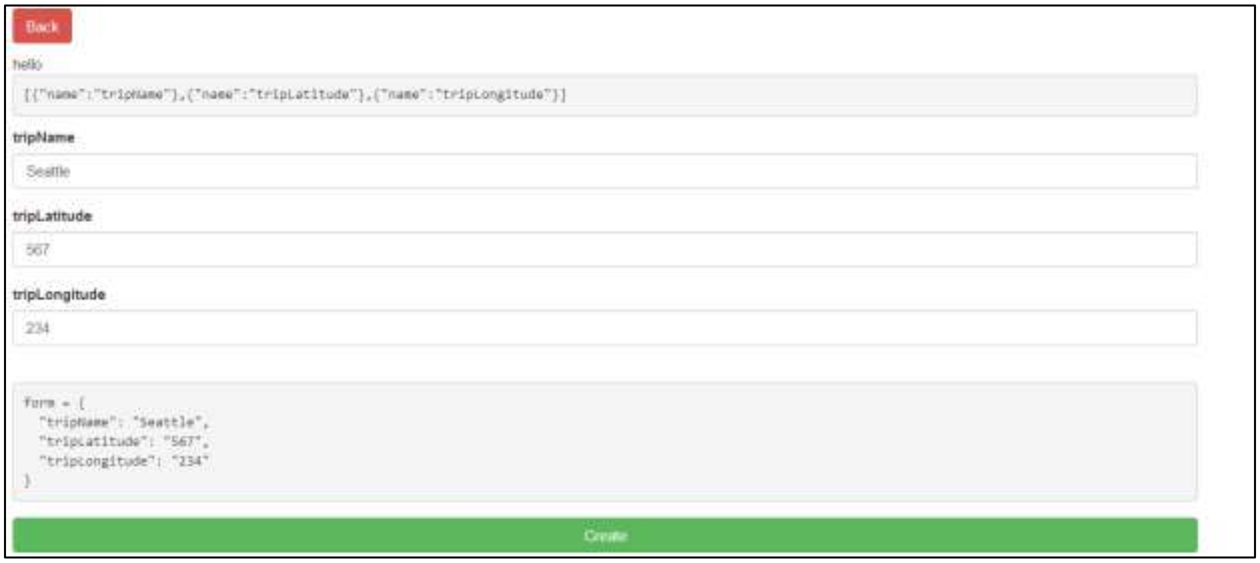
View Fish:



| _id | Fish Name 1 | Fish Length | Date Caught |
|---|---|---|---|
| F1 | F1U1T1 | 203 cms | 06/15/2014 |
| F2 | F2U1T1 | 204 cms | 06/16/2014 |

View works recursively without hardcoding the query; it can access the embedded arrays for reading!

Create new trip for user u1. This adds a new trip to user 1 (works without hardcoding the que ry). But inserting a fish for trip t1 and user u1 fails! Due to non-support from MongoDB.



hello

```
[{"name":"tripName"},{"name":"tripLatitude"},{"name":"tripLongitude"}]
```

tripName: Seattle

tripLatitude: 567

tripLongitude: 234

```
form = {
  "tripName": "Seattle",
  "tripLatitude": "567",
  "tripLongitude": "234"
}
```

## Development details

### Set-up and learning

1. Install MongoDB
2. Install Node.Js
3. Install Angular.Js
4. Learn the above technologies
    a. Saw YouTube videos
    b. Examples on their official sites gave decent understanding
5. Setting up the base code (provided by professor)
6. Understanding the requirement and the base code

### Design discussions

Designs changed w.r.t the implementation for below:

#### Applications

This collection stores the details of the applications created by the developers. Each application has a unique object id generated by MongoDB. This id is used by 'masterApplicationsSchema' as a foreign key reference for the application

#### Schema

Schema for an application is provided by the DU and is stored in MongoDB

#### App Data

The data inserted/edited/deleted by the end users

## Approaches

### *Normalized schema and embedded app data*

#### Schema collection description

1. The schema for all entities for each application is stored in a collection called 'masterApplicationsSchema'
2. This schema is normalized in the sense that parent entity has a property called 'one -to-many' to store reference to the child entity
3. I would prefer explaining the structure w.r.t **Fish 360 application** (Each user has multiple trips and on each trip he/she catches fishes)
4. Each entity (document) has below properties:
   1. **_schema**
      a. _schema Body: the value represents all the fields of this entity and the corresponding type

         **Example:**

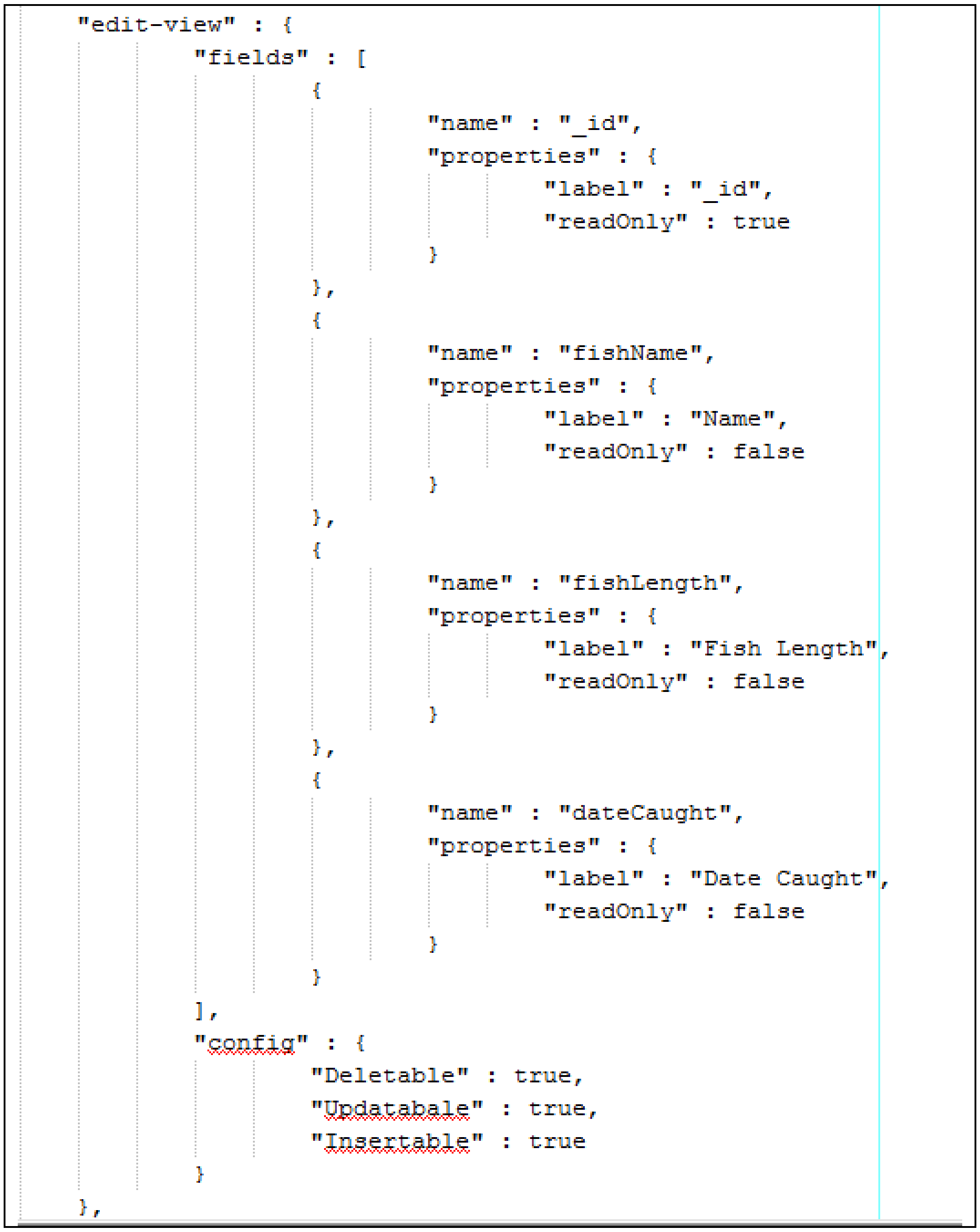
```
{
     "fishName" : "Text",
     "fishLength" : "Text",
    "dateCaught" : "Date"
}
```

      **b. meta-data**

         i. **list-view**: the value is the various configuration possible while viewing the data while running the application (Refer List view configuration for each entity participating in the application –)

```
{
     "Deletable" : true,
     "Reorderable" : true,
     "Insertable" : true,
     "Updatabale": true
},
```

         ii. **edit-view:** the value is the various configuration possible while editing data while running the application (Refer Edit view configuration for each entity participating in the application –)



```
    "edit-view" : {
            "fields" : [
                    {
                            "name" : "_id",
                            "properties" : {
                                    "label" : "_id",
                                    "readOnly" : true
                            }
                    },
                    {
                            "name" : "fishName",
                            "properties" : {
                                    "label" : "Name",
                                    "readOnly" : false
                            }
                    },
                    {
                            "name" : "fishLength",
                            "properties" : {
                                    "label" : "Fish Length",
                                    "readOnly" : false
                            }
                    },
                    {
                            "name" : "dateCaught",
                            "properties" : {
                                    "label" : "Date Caught",
                                    "readOnly" : false
                            }
                    }
            ],
            "config" : {
                    "Deletable" : true,
                    "Updatabale" : true,
                    "Insertable" : true
            }
    },
```

iii. **order-of-fields**: Stores the order in which the fields are to be displayed

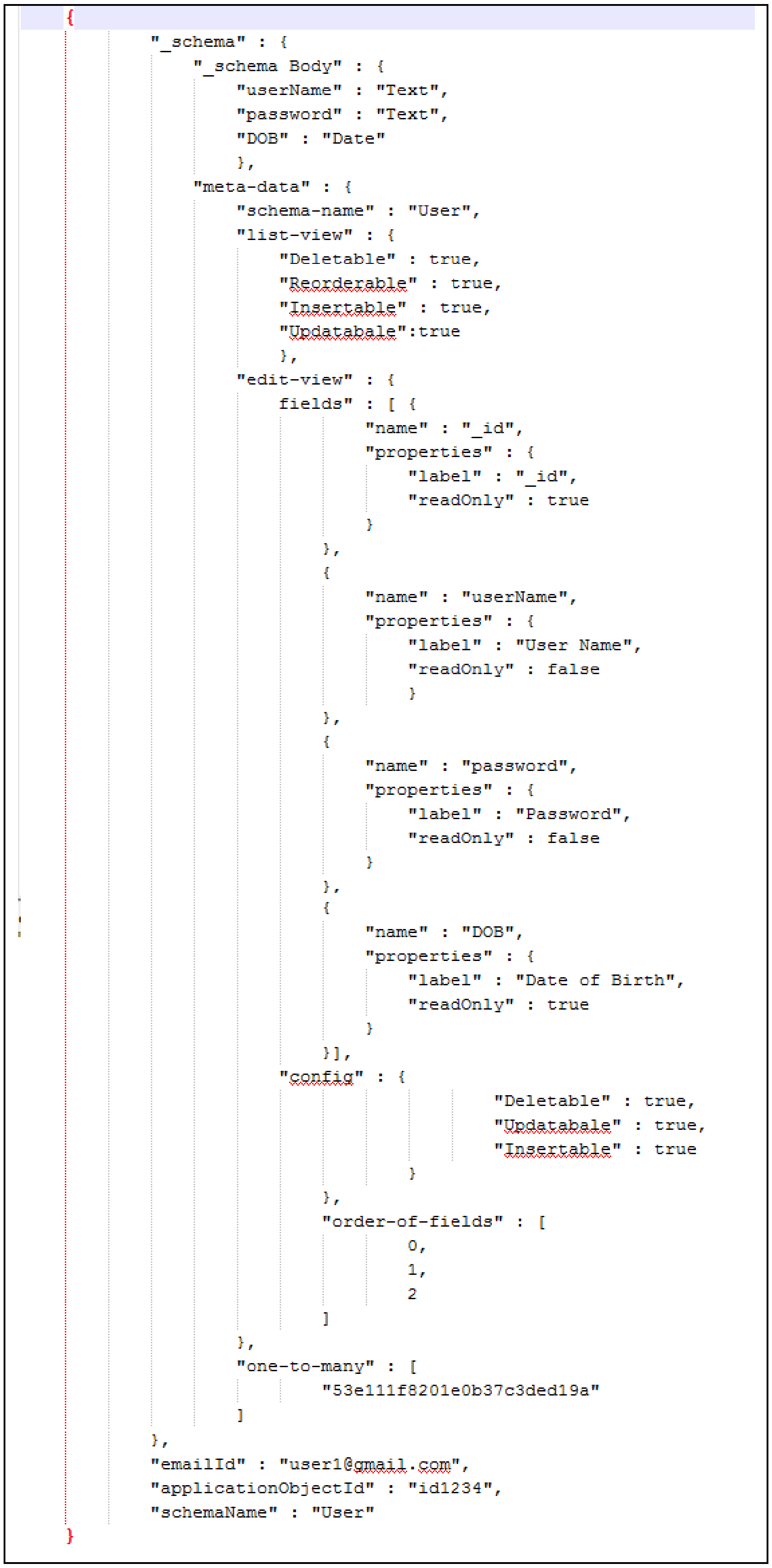
```
"order-of-fields" : [0,2,1]
```

iv. **one-to-many**: Stores the object id of the entity that is the child for this entity. Example, if this is User entity it would store the reference of Trip entity since a user goes on multiple trips

```
"one-to-many" : [53e11187201e0b37c3ded199"]
```

2. **emailId**: Email id of the developer user of this application (unique)
3. **applicationObjectId**: MongoDB generated Object id of the application to which this entity belongs (foreign key to the applications collection)
4. **schemaName:** Name of the entity

Single entity in the masterApplicationsSchema collection looks like below. Example master applications schema is committed in a file separately, named ‘masterApplicationsSchema.txt’



```
{
        "_schema" : {
            "_schema Body" : {
                "userName" : "Text",
                "password" : "Text",
                "DOB" : "Date"
                },
            "meta-data" : {
                "schema-name" : "User",
                "list-view" : {
                    "Deletable" : true,
                    "Reorderable" : true,
                    "Insertable" : true,
                    "Updatabale":true
                    },
                "edit-view" : {
                    fields" : [ {
                            "name" : "_id",
                            "properties" : {
                                "label" : "_id",
                                "readOnly" : true
                            }
                        },
                        {
                            "name" : "userName",
                            "properties" : {
                                "label" : "User Name",
                                "readOnly" : false
                                }
                        },
                        {
                            "name" : "password",
                            "properties" : {
                                "label" : "Password",
                                "readOnly" : false
                            }
                        },
                        {
                            "name" : "DOB",
                            "properties" : {
                                "label" : "Date of Birth",
                                "readOnly" : true
                            }
                        }],
                    "config" : {
                                        "Deletable" : true,
                                        "Updatabale" : true,
                                        "Insertable" : true
                                }
                        },
                        "order-of-fields" : [
                                0,
                                1,
                                2
                        ]
                },
                "one-to-many" : [
                        "53e111f8201e0b37c3ded19a"
                ]
        },
        "emailId" : "user1@gmail.com",
        "applicationObjectId" : "id1234",
        "schemaName" : "User"
}
```

## appData collection entity

- This collection is an array of documents. Each document is data for each application
- The data in each document follows the schema structure in an embedded fashion, like for an application like Fish 360, appData has a document for Fish 360
- The document has an array of objects for User, each user object has embedded array of Trip, each trip object has embedded array of Fish
- Example application data is committed in a file separately, named 'appData.txt'
  Find below a document for Fish 360 example (appId: 53ad3cbfdea87c3bd8666120)

```
{           "appId": "53ad3cbfdea87c3bd8666120",
            "User" : [
                        {
                                    "_id":"u1",
                                    "userName": "user1",
                                    "password": "p1",
                                    "DOB":"03/13/1989",
                                    "Trip" : [
                                                {
                                                            "_id": "T1",
                                                            "tripName" : "NY",
                                                            "tripLatitude" : "12345",
                                                            "tripLongitude" : "67890",
                                                            "Fish" : [{
                                                                          "_id": "F1",
                                                                         "fishName" : "F1U1T1",
                                                                         "fishLength" : "203 cms",
                                                                         "dateCaught" : "06/15/2014"
                                                                        },
                                                                        {
                                                                         "_id": "F2",
                                                                         "fishName" : "F2U1T1",
                                                                        "fishLength" : "204 cms",
                                                                        "dateCaught" : "06/16/2014"
                                                                        }]
                                                            },
                                                {
                                                            "_id": "T2",
                                                            "tripName" : "LA",
                                                            "tripLatitude" : "12345",
                                                            "tripLongitude" : "67890",
                                                            "Fish" : [{
                                                                          "_id": "F1",
                                                                          "fishName" : "F1U1T2",
                                                                          "fishLength" : "203 cms",
                                                                          "dateCaught" : "06/15/2014"
                                                                        },
                                                                        {
```

```
                                                    "_id": "F2",
                                                   "fishName" : "F2U1T2",
                                                   "fishLength" : "203 cms",
                                                   “dateCaught" : "06/15/2014"
                                                 }]
                                        }
                                ]
                        },
                        {
                                "_id":"u2",
                                "userName": "user2",
                                "password": "p2",
                                "DOB":"03/13/1992",
                                "Trip" : [
                                        {
                                                "_id": "T1",
                                                "tripName" : "NY",
                                                "tripLatitude" : "12345",
                                                "tripLongitude" : "67890",
                                                "Fish" : [{
                                                           "_id": "F1",
                                                           "fishName" : "F1U2T1",
                                                          "fishLength" : "203 cms",
                                                          "dateCaught" : "06/15/2014"
                                                        }] //fish ends
                                        }] //trip ends
                        }
                ] //user ends
} //document ends
```

### Pros and Cons

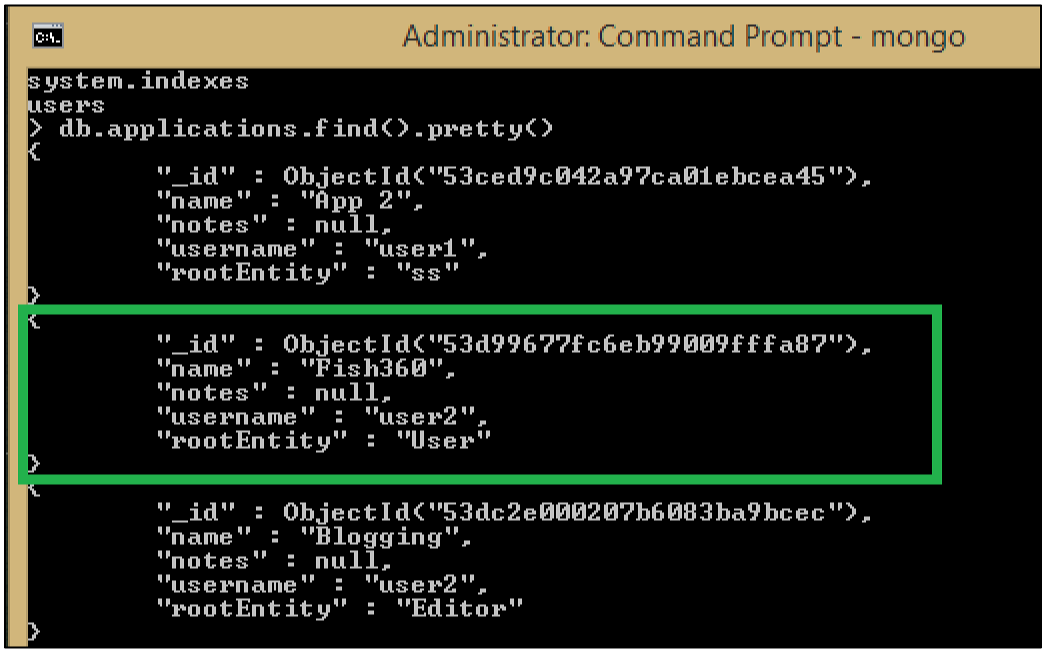
Design wise this is better. Ideally, MongoDB is designed for embedded objects. The problem is that MongoDB doesn’t support operations with 2 more levels of nesting in an array. Refer https://jira.mongodb.org/browse/SERVER-831. This is a major issue w.r.t this design, so we have moved to the design explained below.

## *Schema and app data both normalized*

In this approach, the schema and data collections are both normalized. This is like relational DB, where we have foreign key references in other collections. This designed is well explained as below:

### ‘applications’ collection

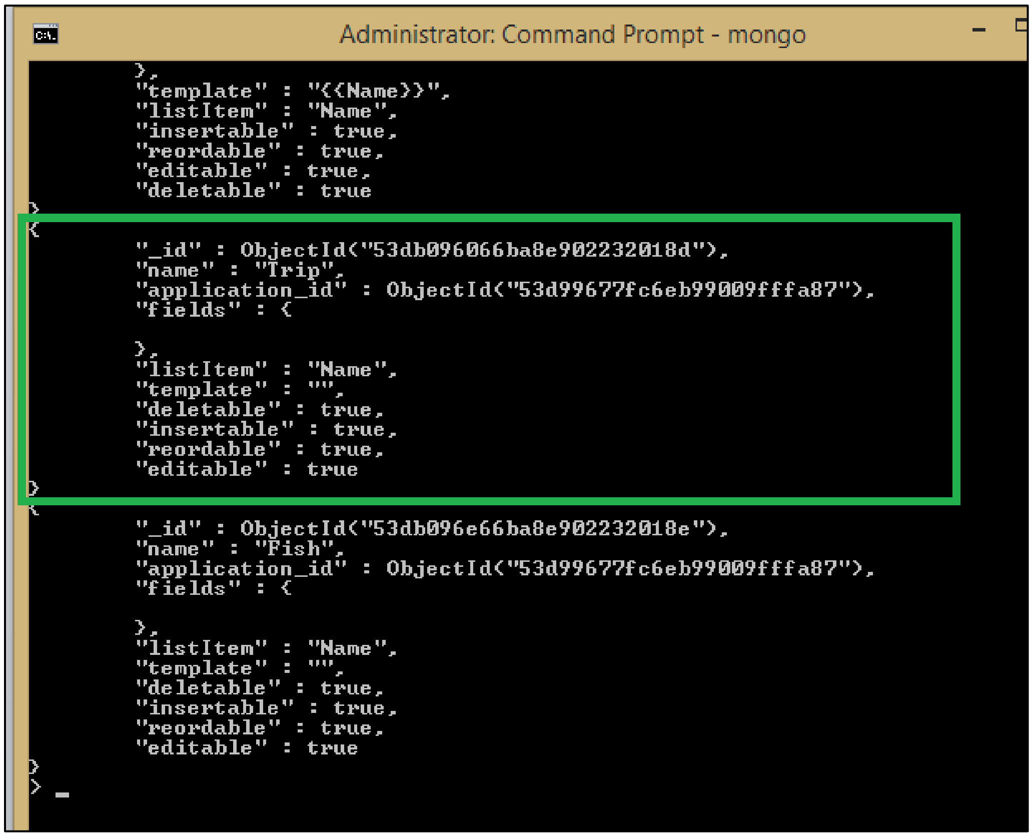
This collection has the application details with the developer’s name and root entity for the application



```
Administrator: Command Prompt - mongo
system.indexes
users
> db.applications.find().pretty()
{
        "_id" : ObjectId("53ced9c042a97ca01ebcea45"),
        "name" : "App 2",
        "notes" : null,
        "username" : "user1",
        "rootEntity" : "ss"
}
{
        "_id" : ObjectId("53d99677fc6eb99009fffa87"),
        "name" : "Fish360",
        "notes" : null,
        "username" : "user2",
        "rootEntity" : "User"
}
{
        "_id" : ObjectId("53dc2e000207b6083ba9bcec"),
        "name" : "Blogging",
        "notes" : null,
        "username" : "user2",
        "rootEntity" : "Editor"
}
```

## Entities collection

This collection has all the entities for all the applications developed by all developers. Each document has a reference to the applicationId to which it belongs and the list-view configuration



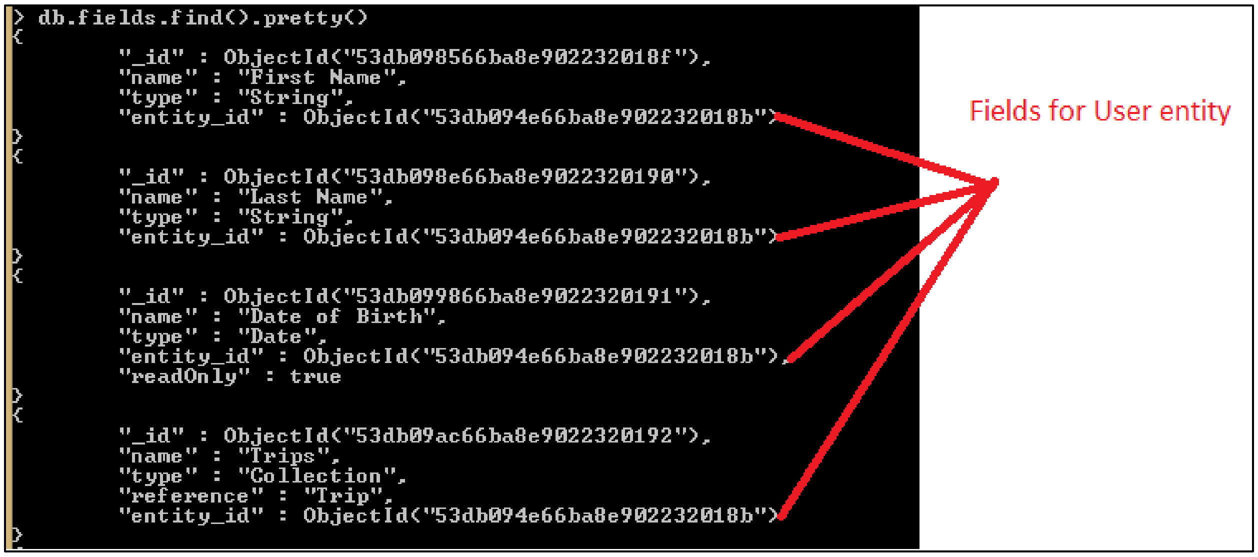
```
Administrator: Command Prompt - mongo
        },
        "template" : "{{Name}}",
        "listItem" : "Name",
        "insertable" : true,
        "reordable" : true,
        "editable" : true,
        "deletable" : true
}
{
        "_id" : ObjectId("53db096066ba8e902232018d"),
        "name" : "Trip",
        "application_id" : ObjectId("53d99677fc6eb99009fffa87"),
        "fields" : {

        },
        "listItem" : "Name",
        "template" : "",
        "deletable" : true,
        "insertable" : true,
        "reordable" : true,
        "editable" : true
}
{
        "_id" : ObjectId("53db096e66ba8e902232018e"),
        "name" : "Fish",
        "application_id" : ObjectId("53d99677fc6eb99009fffa87"),
        "fields" : {

        },
        "listItem" : "Name",
        "template" : "",
        "deletable" : true,
        "insertable" : true,
        "reordable" : true,
        "editable" : true
}
> _
```

## Fields collection

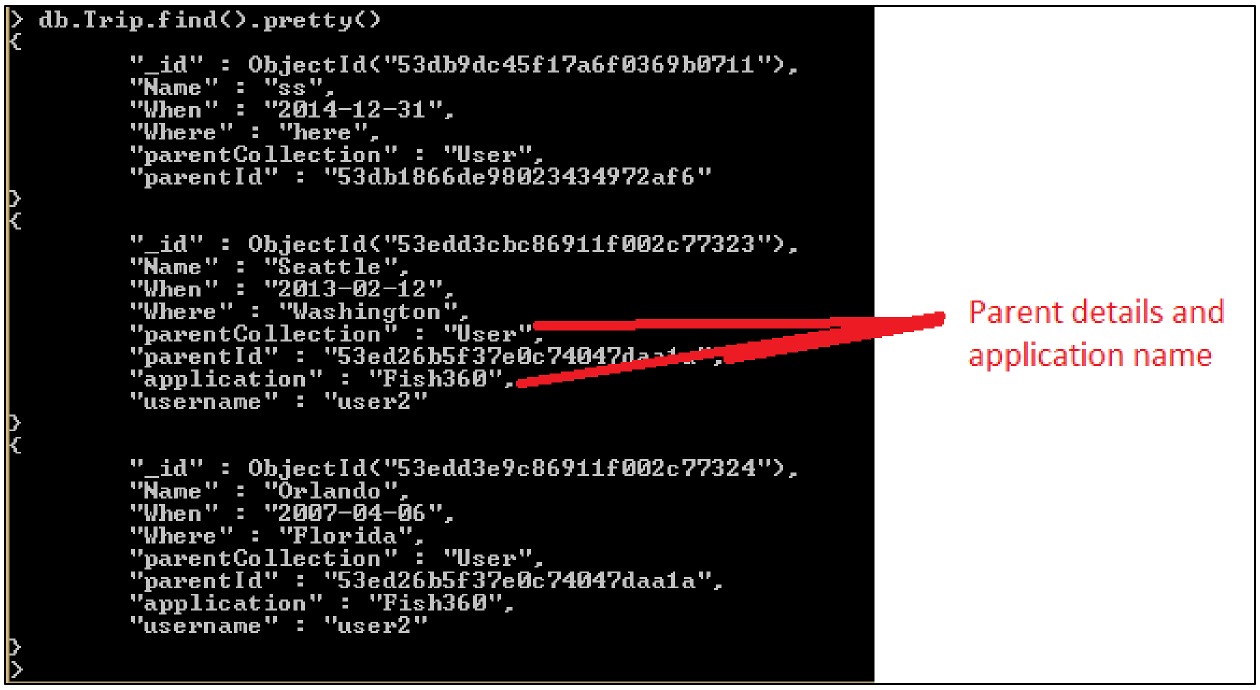
This stores the fields for each entity for each application. This has a reference to entity id. It stores the type for each field accordingly the input control is rendered on the page



```
> db.fields.find().pretty()
{
        "_id" : ObjectId("53db098566ba8e902232018f"),
        "name" : "First Name",
        "type" : "String",
        "entity_id" : ObjectId("53db094e66ba8e902232018b")
}
{
        "_id" : ObjectId("53db098e66ba8e9022320190"),
        "name" : "Last Name",
        "type" : "String",
        "entity_id" : ObjectId("53db094e66ba8e902232018b")
}
{
        "_id" : ObjectId("53db099866ba8e9022320191"),
        "name" : "Date of Birth",
        "type" : "Date",
        "entity_id" : ObjectId("53db094e66ba8e902232018b"),
        "readOnly" : true
}
{
        "_id" : ObjectId("53db09ac66ba8e9022320192"),
        "name" : "Trips",
        "type" : "Collection",
        "reference" : "Trip",
        "entity_id" : ObjectId("53db094e66ba8e902232018b")
}
```

## Data collections

We have separate collections with the name of the entities that stores user's data. Note that the data is also storing references to the child entities as defined in the entities collection for an application.



```
> db.Trip.find().pretty()
{
        "_id" : ObjectId("53db9dc45f17a6f0369b0711"),
        "Name" : "ss",
        "When" : "2014-12-31",
        "Where" : "here",
        "parentCollection" : "User",
        "parentId" : "53db1866de98023434972af6"
}
{
        "_id" : ObjectId("53edd3cbc86911f002c77323"),
        "Name" : "Seattle",
        "When" : "2013-02-12",
        "Where" : "Washington",
        "parentCollection" : "User",
        "parentId" : "53ed26b5f37e0c74047daa1a",
        "application" : "Fish360",
        "username" : "user2"
}
{
        "_id" : ObjectId("53edd3e9c86911f002c77324"),
        "Name" : "Orlando",
        "When" : "2007-04-06",
        "Where" : "Florida",
        "parentCollection" : "User",
        "parentId" : "53ed26b5f37e0c74047daa1a",
        "application" : "Fish360",
        "username" : "user2"
}
>
```

### Pros and Cons

This approach is easier, this was chosen mainly because, not all nested array operations are supported by MongoDB.

## Learnings

- Absolutely everything!
- The way this pattern works, the 'MEAN stack'
- Design something from scratch
- There were a lot of findings about each of the technology
- Since the application is pretty big, we have kind of covered a lot of uses case, I am sure there is more to cover
    - Iterating over array of objects
    - Iterating over array of objects with nested array in it
    - Accessing dynamic variables while querying MongoDB
    - Logic for maintaining the state
    - Use of regex for similar url pattern
    - Displaying something on conditions
    - Deal with async calls, wait for calls to return back, nested mongodb queries
    - Workarounds for no support for Positional Operator Matching Nested Arrays (though it isn't good performance wise)
    - Tackle challenges mentioned below

## Challenges

- I have never worked with MongoDB, Node.js and Angular.js. The very first task was to know these properly to be able to develop an application in it
- Design for any application is always a default challenge, deciding on that was difficult. A design would work in theory, but when it came to implementation some features were not supported my MongoDB itself. Like, pushing to nested arrays 2 and more level deep. Details can be read here: https://jira.mongodb.org/browse/SERVER-831
- With embedded structure, the major challenge was to access the nested arrays. Perform CRUD operations on them.
- After figuring out how to query in MongoDB, the next was to fire the query from the code (different syntax), dynamic values, this part wasn't difficult but took up time
- Since this is something all of us were trying for the first time, we had to do some re-work
- One more challenge that I personally faced was, knowing the exact search term to Google

## Code flow and structure

1. All the routes are defined in app.js file
2. The controllers for each feature are defined in their respective .js files in the js folder
3. The templates for these are defined in the .html files in the templates folder
4. index.html is the start page that has the ng-view that is replaced by the html templates depending on the URL route and imports all the css and angular js and jquery

5. sever.js has the connection to the database and all the post/get/delete/put functions that insert/save/update/delete into MongoDB

### Testing

- Performed unit testing by enabling/disabling each feature
- Performed integration testing after getting all features in place
- Performed regression testing after adding every new feature
- Created different type of applications and tested it

## Note of Thanks

Professor Jose gave us quite a lot of his time to get started and reviewed whatever we did and accordingly guided us. Thank you so much for all the help and giving an opportunity to learn new technologies and work on some real time project

## Future

Since I really liked this project, out of personal interest I would continue working on this!