STRATEGIC
SYSTEMS & TECHNOLOGY CORPORATION

Backbone Overview

How does the Backbone platform help the SST development team create more with less?

Team Size vs Results

2 Devs

1 SDET

1 QA

8 Active Application Use Cases

(and some minor variations not listed):

- Asset Tracking - a system for inventorying and tracking physical assets
- Item Tracking - An RFID based system for tracking items
- Rapid Equipment Exchange (REX) - system for managing equipment RMAs and repairs
- FLAIR Asset Tracking - asset tracking specific to the State of Florida

(continued next slide)

Team Size vs Results

2 Devs

1 SDET

1 QA

8 Active Application Use Cases

(and some minor variations not listed):

- Wise ID - a system for checking people in and out
- Metadata Creator - an internal tool I'll talk about more
- Nucleus - another internal tool used to manage the tenants of use cases and similar information.
- One Touch Deployment - an internal tool for deploying the application to Kubernetes and MongoDB

Team Size vs Results

2 Devs

1 SDET

1 QA

CI/CD - All code:

Runs through automated builds and is thoroughly tested with:

- 1,876 API tests,
- 2,300 UI automation tests (including device UIs and difficult automation scenarios like physical RFID tag scanning) and
- Some large number of unit tests (1,929 on just the front end UI).

Team Size vs Results

2 Devs

1 SDET

1 QA

CI/CD -

- All application services are containerized and deployed to GKE (Google Cloud Kubernetes) with the touch of one button
- The team strives for biweekly deployments, with feature flags to control exposure of new functionality
- Blue/Green deployment environments are available and can be used as necessary (high test coverage and feature flags usually make this unnecessary)

Team Size vs Results

2 Devs

1 SDET

1 QA

Backbone Provides Data Replication and Offline Usage

- Offline support for all front ends/clients.
- Real-time transaction replication across devices while online.
- Conflict reconciliation and synchronization for offline usage.

Team Size vs Results

2 Devs

1 SDET

1 QA

Backbone has:

- Front ends/clients running on responsive web, Android native (for RFID reading mobile devices) and several RFID fixed reader devices.
- 2 Brands, each with consistent styling and design for all front-end UIs.
- Documented REST APIs for each use case.
- SOC II compliance
- Negligible downtime since inception of the product.
- Currently averages ~100 requests per second (mostly RFID tag reads).

How?

First some context…

Changes in Focus and Fluctuations in Team Size

SST focuses on software and hardware to track things.

The focus has shifted back and forth between custom development and off the shelf software packages. The team size has fluctuated, increasing when custom development was the focus.

The problem with custom development…

Is that you have to maintain the code. Even if you don't agree to maintain the software from the outset, if enough money is offered, you will end up fulfilling a maintenance contract.

As you continue to accumulate clients, you will continue to accumulate maintenance contracts.

The problem with custom development…

This all leads to a nightmare where your most reliable employees will end up maintaining a very broad range of old, sometimes poorly selected technologies – effectively punishing them for being reliable – or you will have difficulty hiring new employees because they will not want to work with the older (sometimes horrible) technologies and codebases.

Solutions that did not work for us

- **Reusable cross-application components -** this is a catch-all term for everything that resembles DLLs (com, .net gac, etc..). In our experience, each new custom implementation team will usually either avoid the use of DLLs provided by previous implementations or will break their backward compatibility.

- **Code gen -** The idea here is to create templates for code generation so that it will be easy to upgrade old clients by generating new code from the latest templates. This had the same problems as DLLs, plus the added headache of reapplying (Or losing) modifications made post generation (or creating hooks to modify the results of generated code).

Solutions that <u>did not</u> work for us

- **Low/No Code -** Implement each use case with a vendor tool which creates 95% of your app. We found that we would spend the vast majority of our time hacking the edge cases which the tool could not handle. We would often repeat this when the next release of the vendor software broke our hacks.

- **Web/Microservices -** Same problems as DLLs. In our experience, each new custom implementation team will usually either avoid the use of web services and microservices provided by previous implementations or will break their backward compatibility.

Remember: these are not problems we encountered with the technologies themselves so much as the reuse of the technology artifacts over serial implementations of new custom products.

Enough context - so what DID work?

Instead of separate apps for each use case, several years ago, we built one evergreen app (which we call Backbone) that could be configured to implement all of our use cases.

What constraints made the most sense?

Hopefully, it is obvious that an infinitely configurable application would consume infinite development resources by requiring infinite time to implement – so certain constraints were necessary.

At least three types of constraints were identified:

1. Domain constraints
2. Architectural constraints
3. Technology constraints

Domain Constraints

Backbone is for use cases that allow users to perform data entry and track things.

Architectural Constraints

We designate a term “namespace” to describe a large category of data (conceptually similar to a database) and designate a term “relation” to describe a grouping of similar data that resides inside a namespace (similar to a table).
If we use certain technologies (which I’ll discuss next), then we can use those same namespaces and relations in every tier of Backbone.

Technology Constraints

When evaluating alternatives, our primary driver was reusability of code and data schemas across as many contexts as possible. Our developers needed to be comfortable in every part of our application. When designing Backbone, we needed to find multipurpose technologies which would flatten the overall learning curve.

At the time, JavaScript was the only programming language that could reasonably work in a browser, on the server side, on devices and in our build pipeline.

Backbone was developed near the beginning of the JavaScript renaissance. The technologies we chose: React, React Native, Node, MongoDB Atlas and Kubernetes were still very cutting edge and risky choices. Most developers didn't even know what Kubernetes was when we decided to bet on it.

React   node JS   React Native   MongoDB Atlas   kubernetes

Technology Constraints

To enable code sharing and easy maintainability, we wanted data to look the same in every part of the application. Working across tiers would be vastly easier if the data (i.e. namespaces and relations) were nearly identical.

A document oriented database which stored JSON seemed like the obvious choice and after much comparison, we chose Atlas, the MongoDB cloud offering.

An Example: Namespaces and Relations

Imagine you configure an application use case like this:

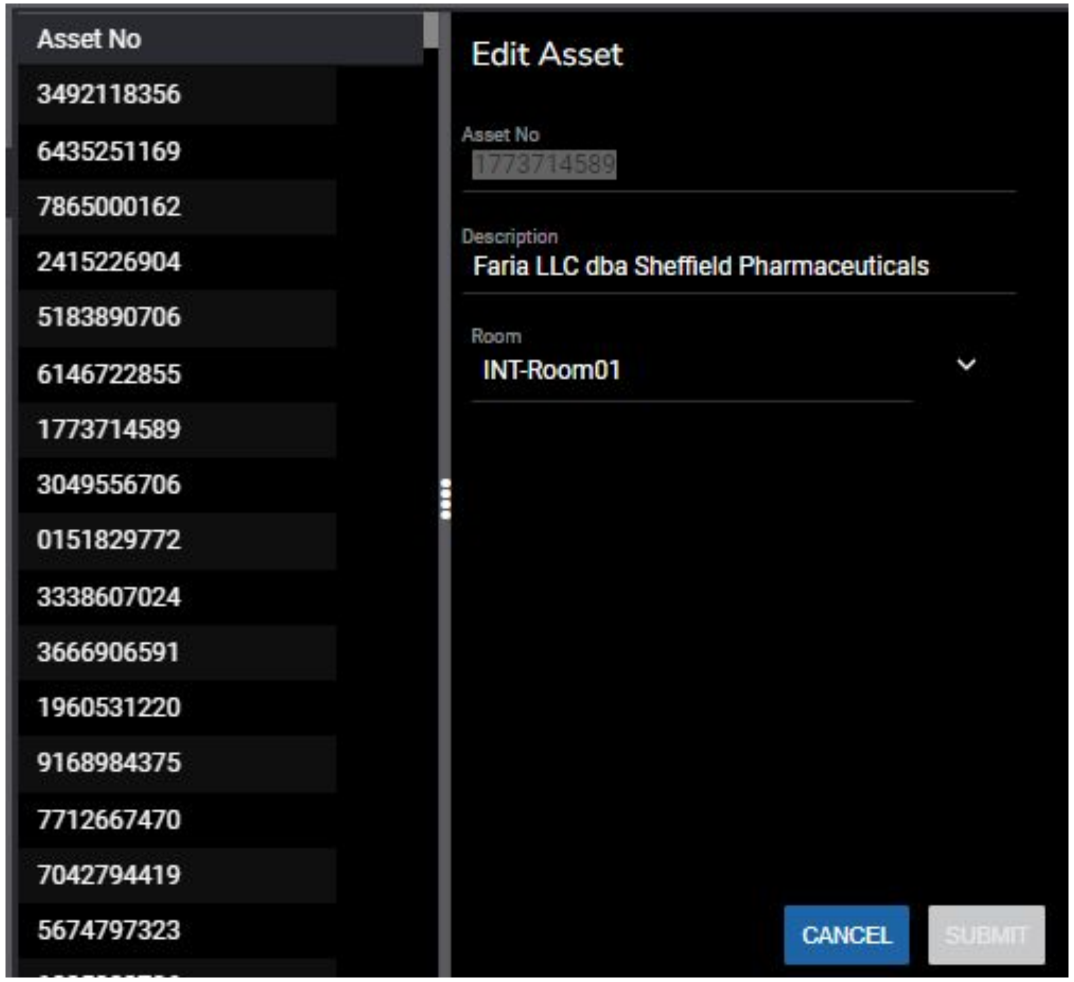
Define ‘**Item**’ as a namespace,
and ‘**Asset**’ as a relation in that namespace,

Then define ‘**Location**’ as a namespace,
and ‘**Room**’ as a relation in that namespace.

This allows us to shape the **UI** in a way which is very similar to the shape of the **API** and the shape of the data.

An Example:

An Asset relation might have a UI that looks like this:



Asset No
3492118356
6435251169
7865000162
2415226904
5183890706
6146722855
1773714589
3049556706
0151829772
3338607024
3666906591
1960531220
9168984375
7712667470
7042794419
5674797323

Edit Asset

Asset No
1773714589

Description
Faria LLC dba Sheffield Pharmaceuticals

Room
INT-Room01

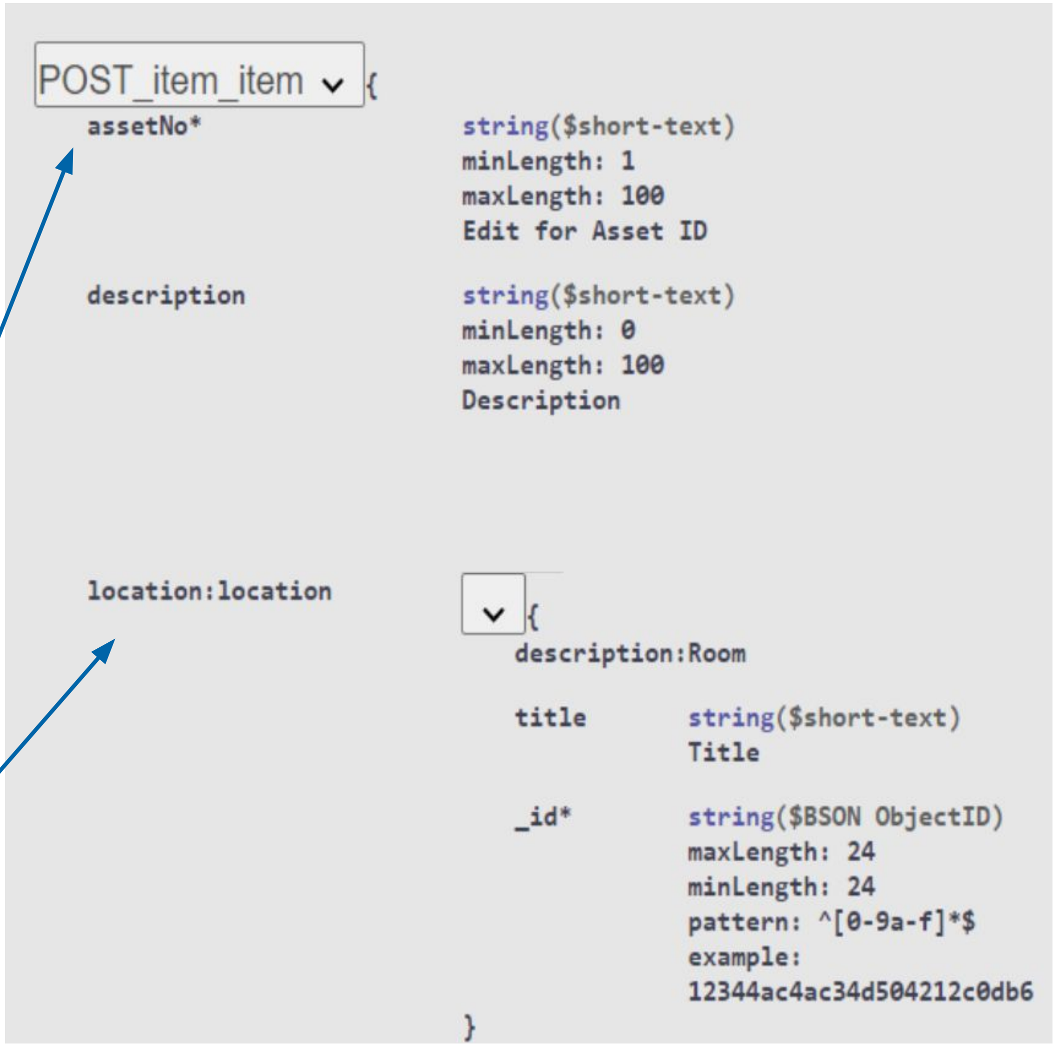
An Example:

An Asset relation might address a swagger UI that looks similar to this:

Clarification – namespace/relations will usually have "pretty names" (i.e. aliases) which vary between use case.

So an "item:item" namespace:relation might have pretty names of "Item:Asset".

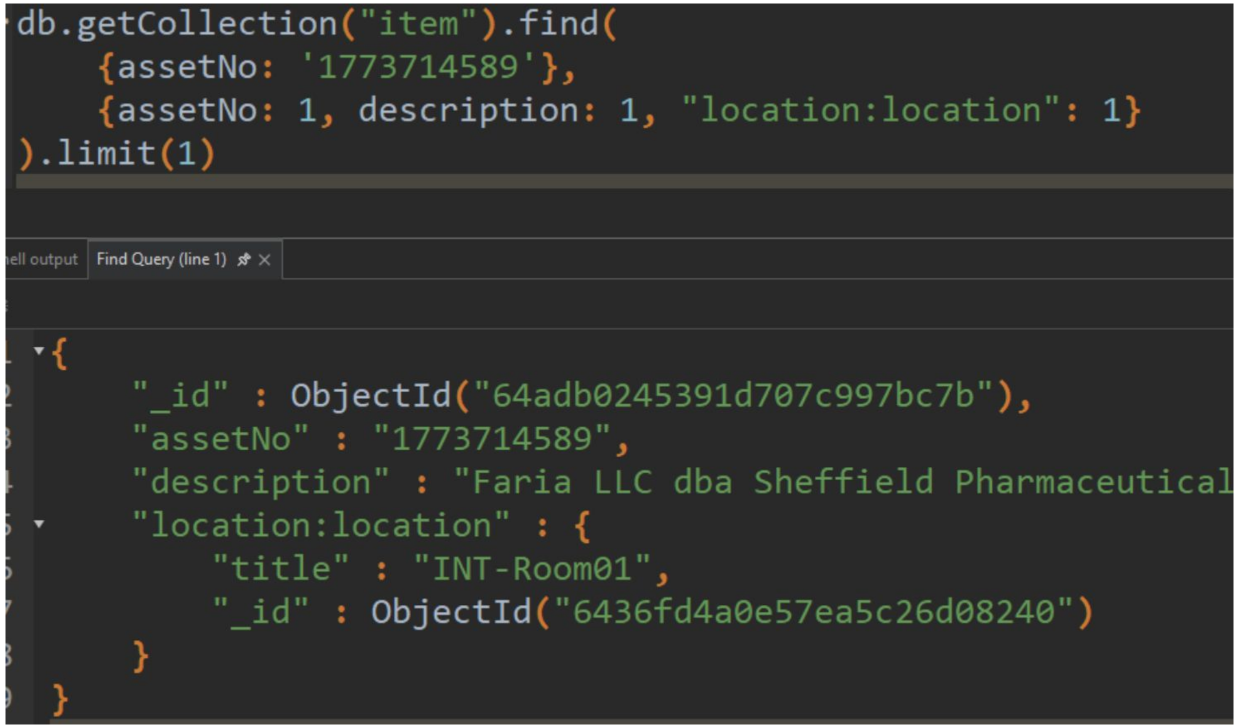
A "location:location" namespace:relation might have pretty names of "Location:Room"



```
POST_item_item {
    assetNo*              string($short-text)
                          minLength: 1
                          maxLength: 100
                          Edit for Asset ID

    description           string($short-text)
                          minLength: 0
                          maxLength: 100
                          Description


    location:location     {
                            description:Room

                            title       string($short-text)
                                        Title

                            _id*        string($BSON ObjectID)
                                        maxLength: 24
                                        minLength: 24
                                        pattern: ^[0-9a-f]*$
                                        example:
                                        12344ac4ac34d504212c0db6
                          }
```

An Example:

And it might have data that looks like this:



```
db.getCollection("item").find(
    {assetNo: '1773714589'},
    {assetNo: 1, description: 1, "location:location": 1}
).limit(1)
```

ell output | Find Query (line 1)

```
{
    "_id" : ObjectId("64adb0245391d707c997bc7b"),
    "assetNo" : "1773714589",
    "description" : "Faria LLC dba Sheffield Pharmaceutical
    "location:location" : {
        "title" : "INT-Room01",
        "_id" : ObjectId("6436fd4a0e57ea5c26d08240")
    }
}
```

An Example

To summarize, in our software if you:

Define ‘**Item**’ as a namespace,

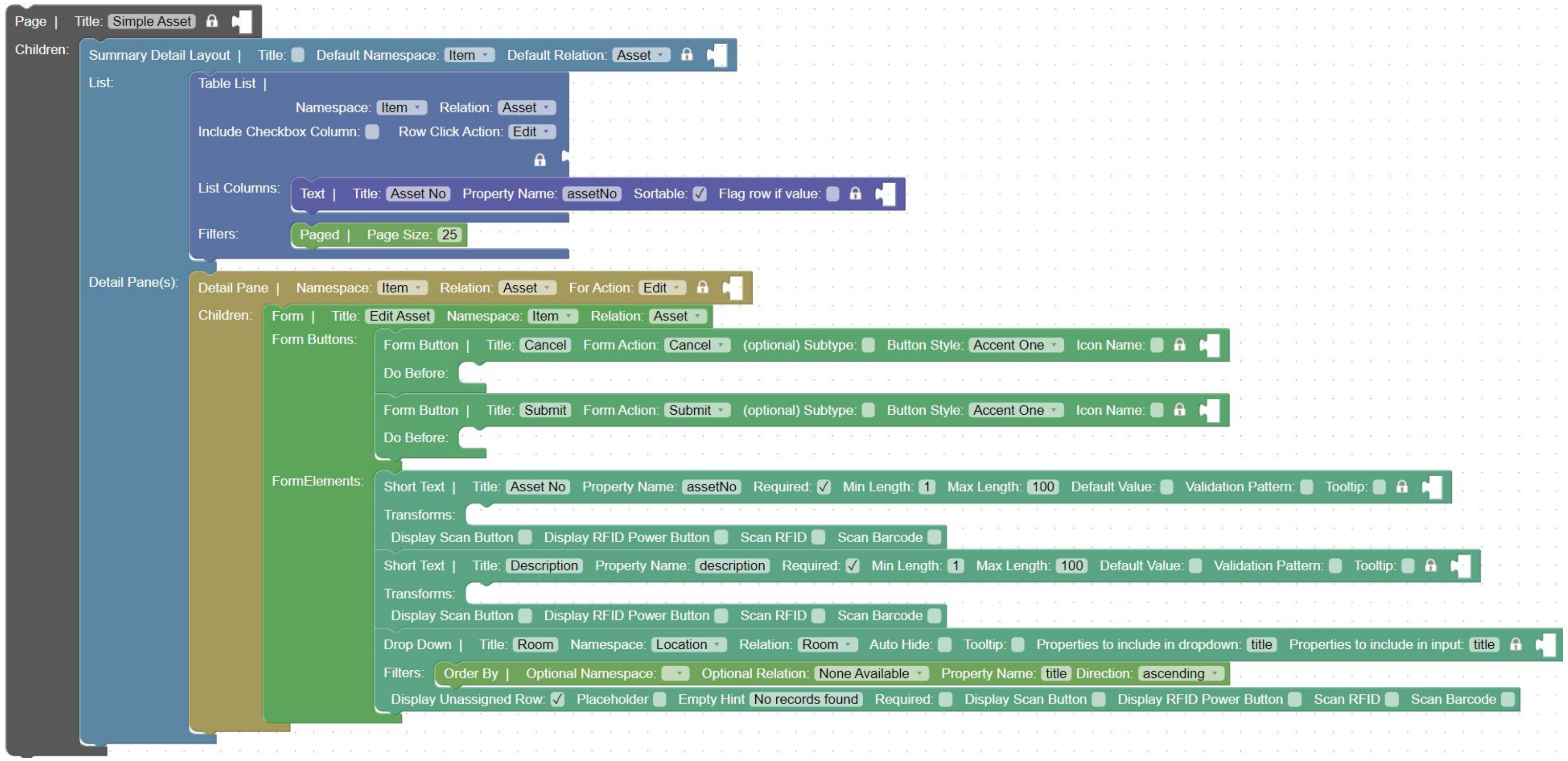
and ‘**Asset**’ as a relation in that namespace,

Then define ‘**Location**’ as a namespace,
and ‘**Room**’ as a relation in that namespace.

Then you can extrapolate a **UI**, an **API** and the **shape of your data** with **metadata** that looks like this (see next slide).

An Example

Metadata as described in the Metadata Creator use case of Backbone.



```
Page | Title: Simple Asset
Children: Summary Detail Layout | Title: ☐ Default Namespace: Item Default Relation: Asset
  List: Table List |
          Namespace: Item Relation: Asset
          Include Checkbox Column: ☐ Row Click Action: Edit
        List Columns: Text | Title: Asset No Property Name: assetNo Sortable: ✓ Flag row if value: ☐
        Filters: Paged | Page Size: 25
  Detail Pane(s): Detail Pane | Namespace: Item Relation: Asset For Action: Edit
        Children: Form | Title: Edit Asset Namespace: Item Relation: Asset
          Form Buttons: Form Button | Title: Cancel Form Action: Cancel (optional) Subtype: ☐ Button Style: Accent One Icon Name: ☐
                          Do Before:
                        Form Button | Title: Submit Form Action: Submit (optional) Subtype: ☐ Button Style: Accent One Icon Name: ☐
                          Do Before:
          FormElements: Short Text | Title: Asset No Property Name: assetNo Required: ✓ Min Length: 1 Max Length: 100 Default Value: ☐ Validation Pattern: ☐ Tooltip: ☐
                          Transforms:
                          Display Scan Button ☐ Display RFID Power Button ☐ Scan RFID ☐ Scan Barcode ☐
                        Short Text | Title: Description Property Name: description Required: ✓ Min Length: 1 Max Length: 100 Default Value: ☐ Validation Pattern: ☐ Tooltip: ☐
                          Transforms:
                          Display Scan Button ☐ Display RFID Power Button ☐ Scan RFID ☐ Scan Barcode ☐
                        Drop Down | Title: Room Namespace: Location Relation: Room Auto Hide: ☐ Tooltip: ☐ Properties to include in dropdown: title Properties to include in input: title
                          Filters: Order By | Optional Namespace: Optional Relation: None Available Property Name: title Direction: ascending
                          Display Unassigned Row: ✓ Placeholder ☐ Empty Hint No records found Required: ☐ Display Scan Button ☐ Display RFID Power Button ☐ Scan RFID ☐ Scan Barcode ☐
```

This is exactly what Backbone does

We define the shape of all our Use Cases with configuration that consumes a set of namespaces and relations to describe characteristics of a UI and characteristics of the data in that UI.

Backbone provides a set of services that can consume that metadata and together, create an entire use case.