Microsoft

Smilr
Microservices Showcase & Demo Application

Ben Coleman
Cloud Architect & Evangelist

@BenCodeGeek

April 2018

Project Goals

Showcase for Azure compute & app dev scenarios

Tie together disparate architecture choices

Develop a real working demo system

Explore use of new design patterns and technologies

Technology Goals & Aspirations

Microservices

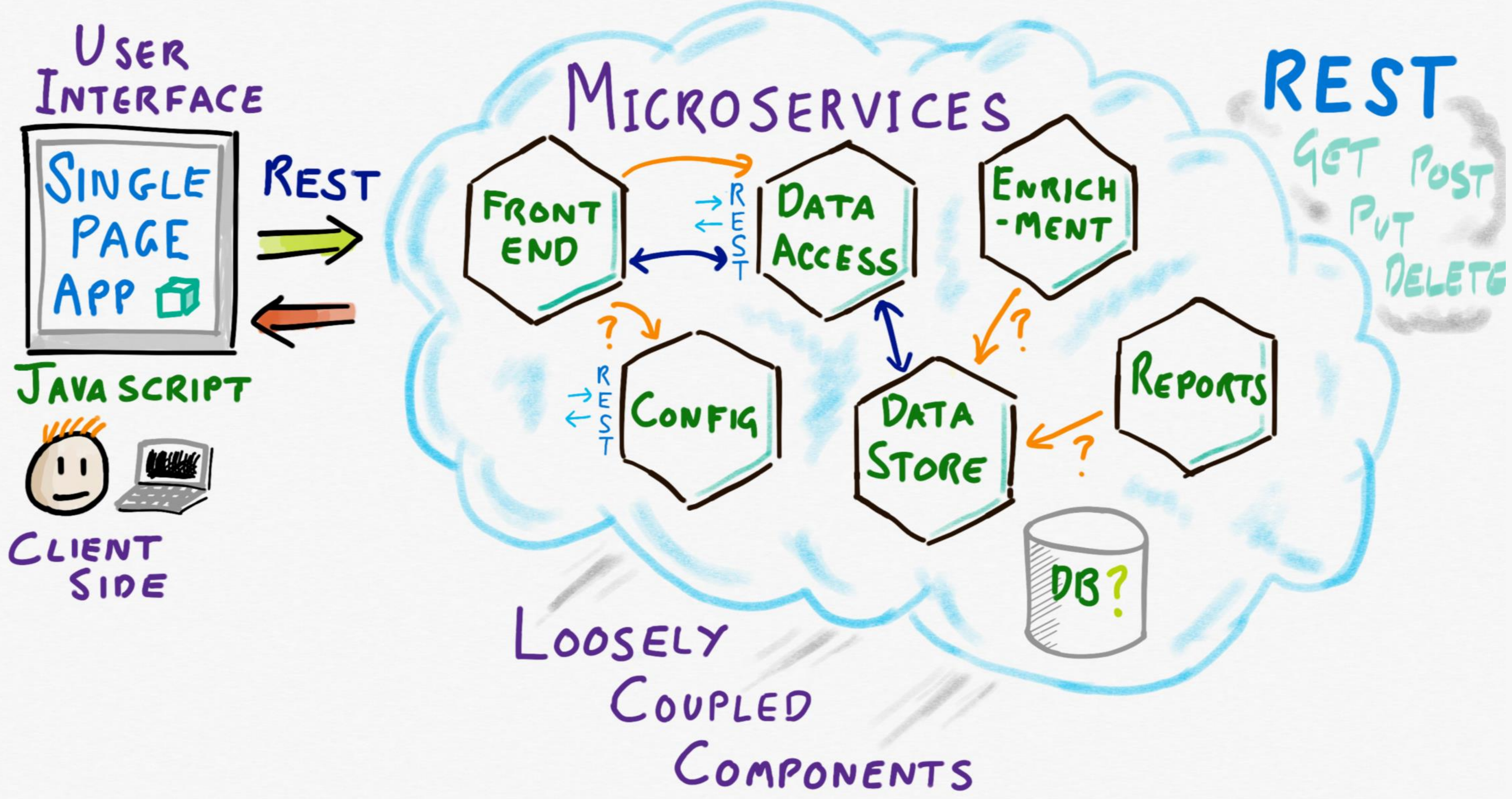
Modern

REST

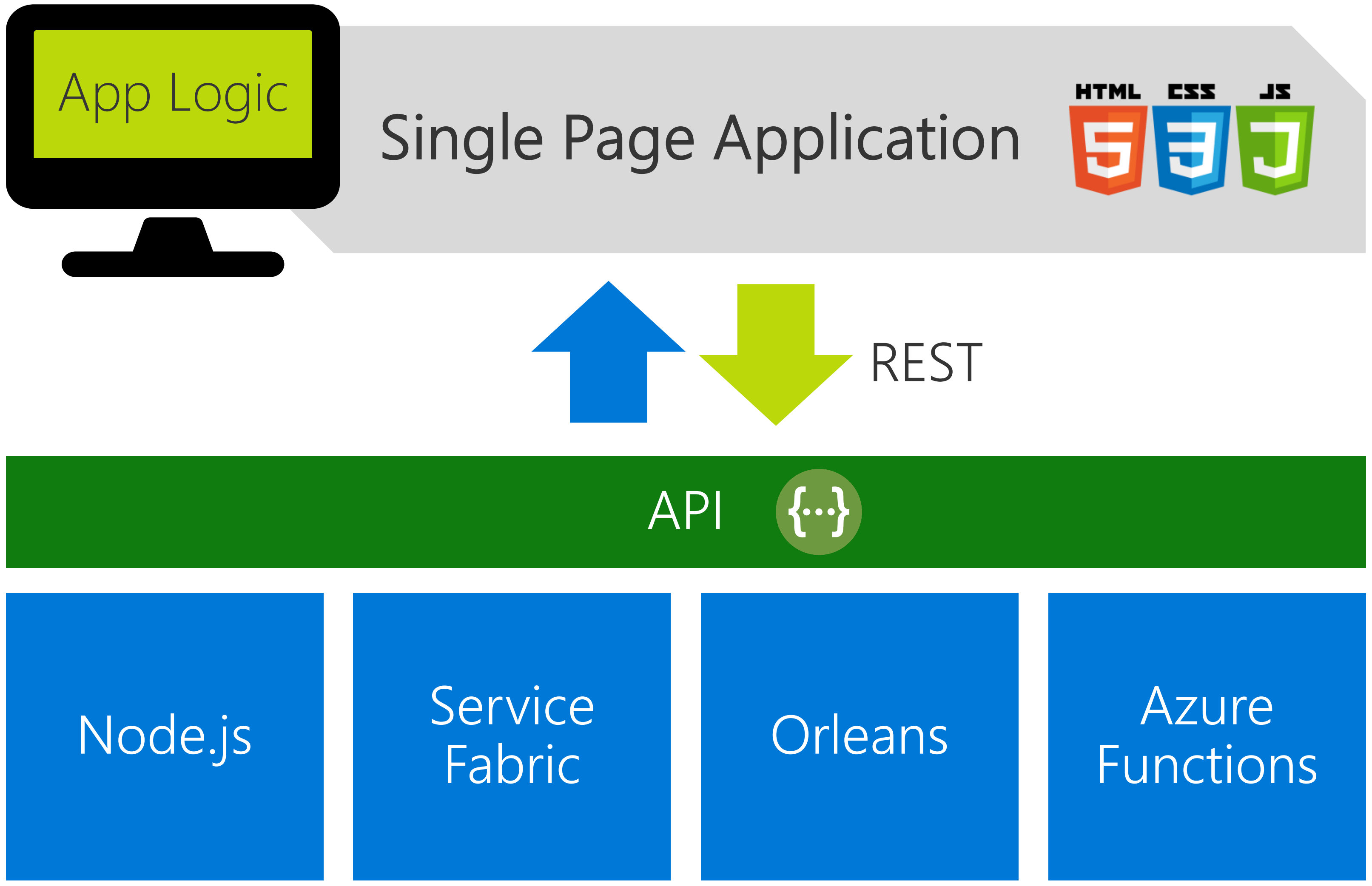
Data

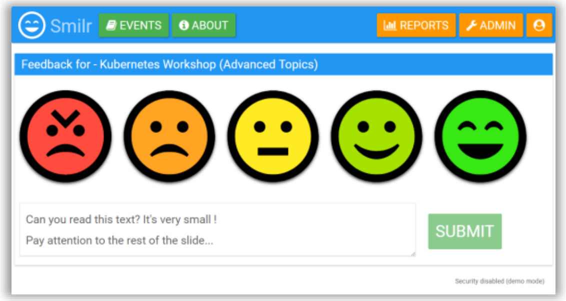
UI

Functional/Concept Architecture



Modern Web Architecture





Feedback for - Kubernetes Workshop (Advanced Topics)

Can you read this text? It's very small !

Pay attention to the rest of the slide...

Security disabled (demo mode)

Implementations

...more to come?
.NET Core, Go

Swagger / OpenAPI

- Defines the exact “shape” of the API
- Describes
  - API operations
  - Input & outputs
  - Data formats
- Like WSDL but easy to use ;)

The contract that all implementations need to adhere to

POSTMAN

Implementation

aka.ms/smilr

Azure PaaS Demo

Open Source Implementation – Tech Stack

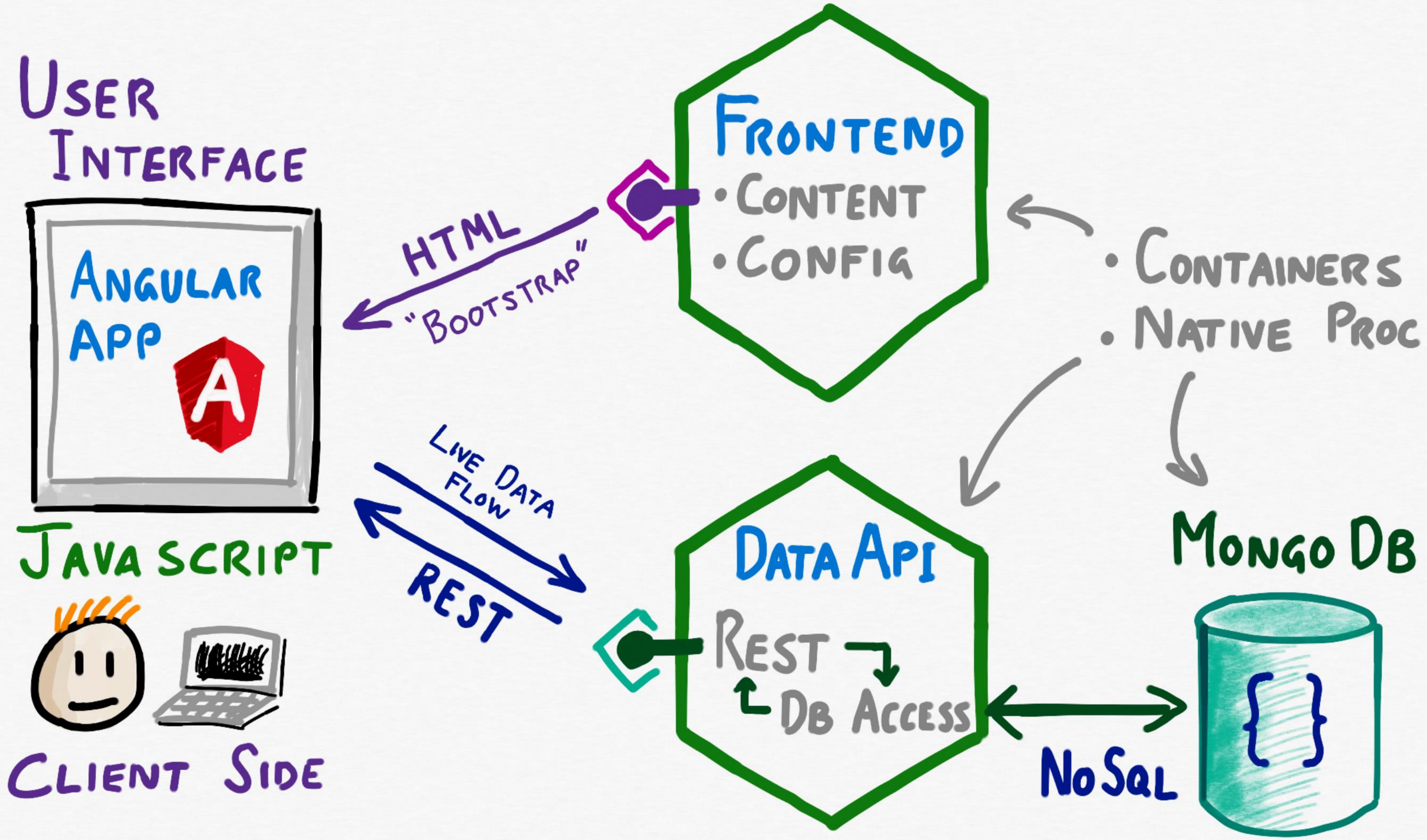
Angular 5 – UI & Client

Node.js & Express – Services & API

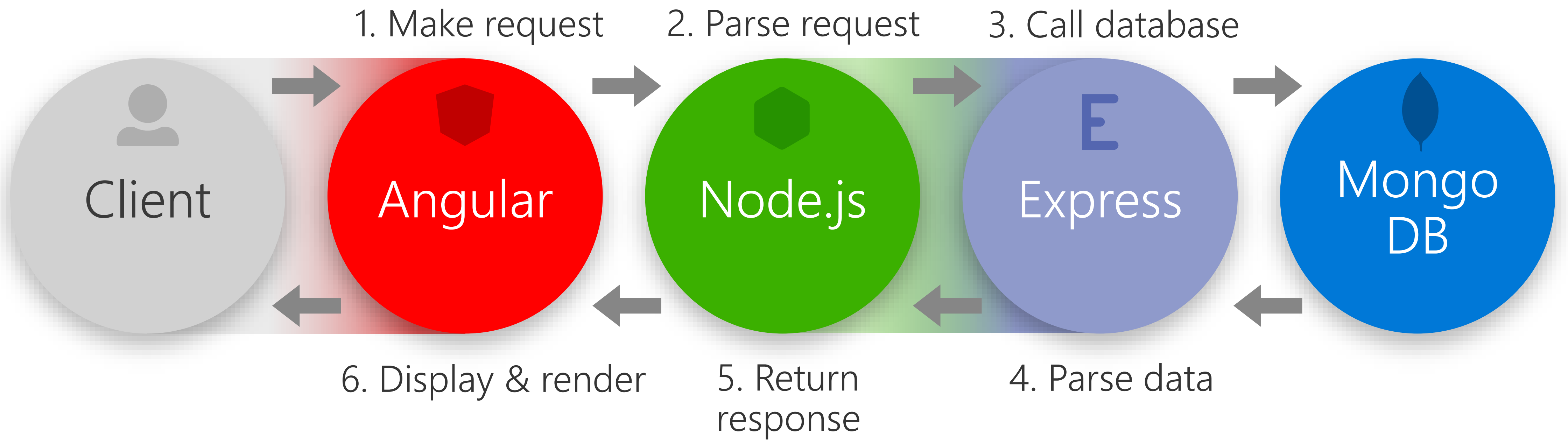
MongoDB – NoSQL Database

Mongo Express Angular Node Stack

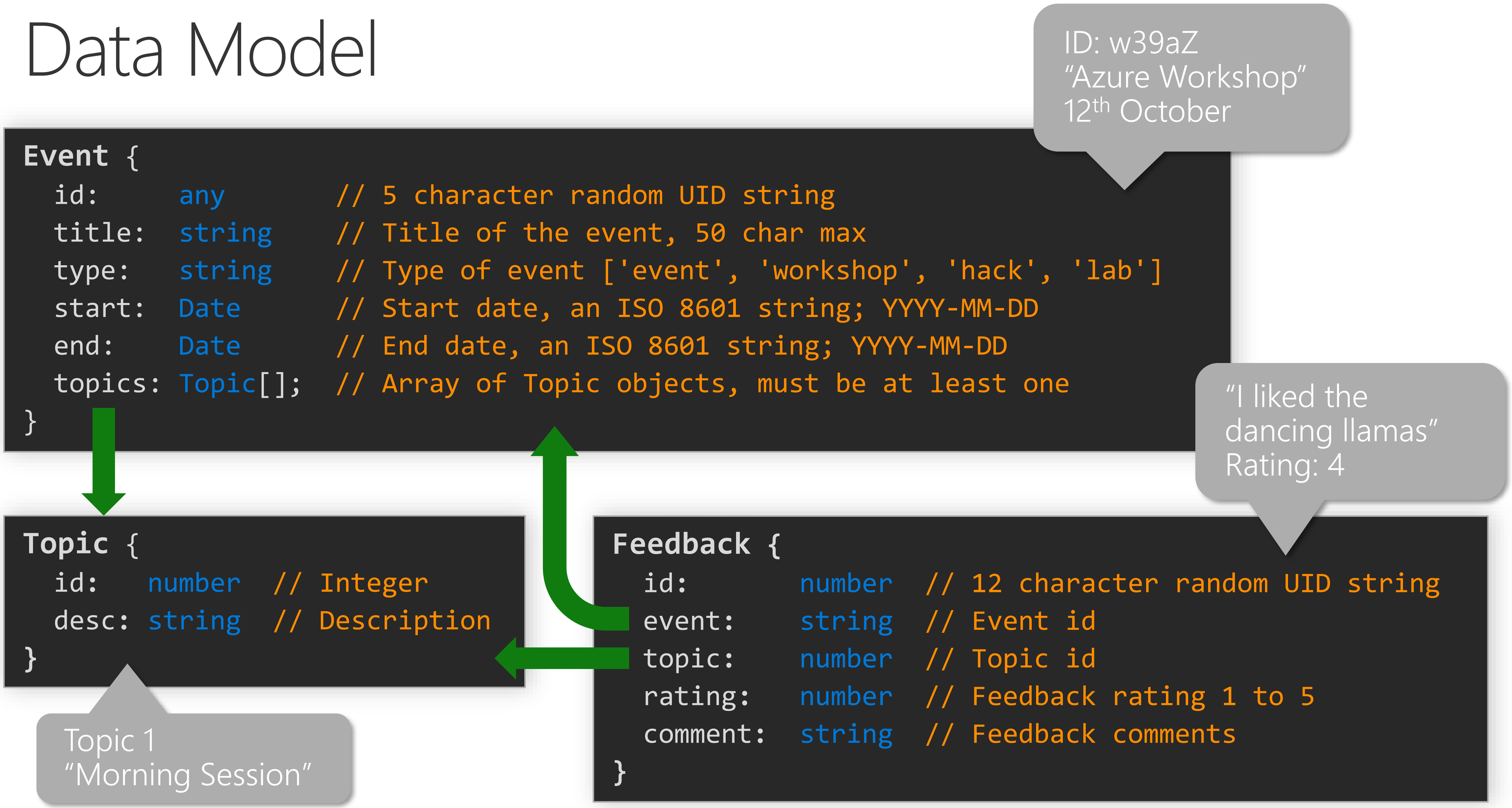
Node.js Architecture



Basic Data Flow & Interaction



Data Model

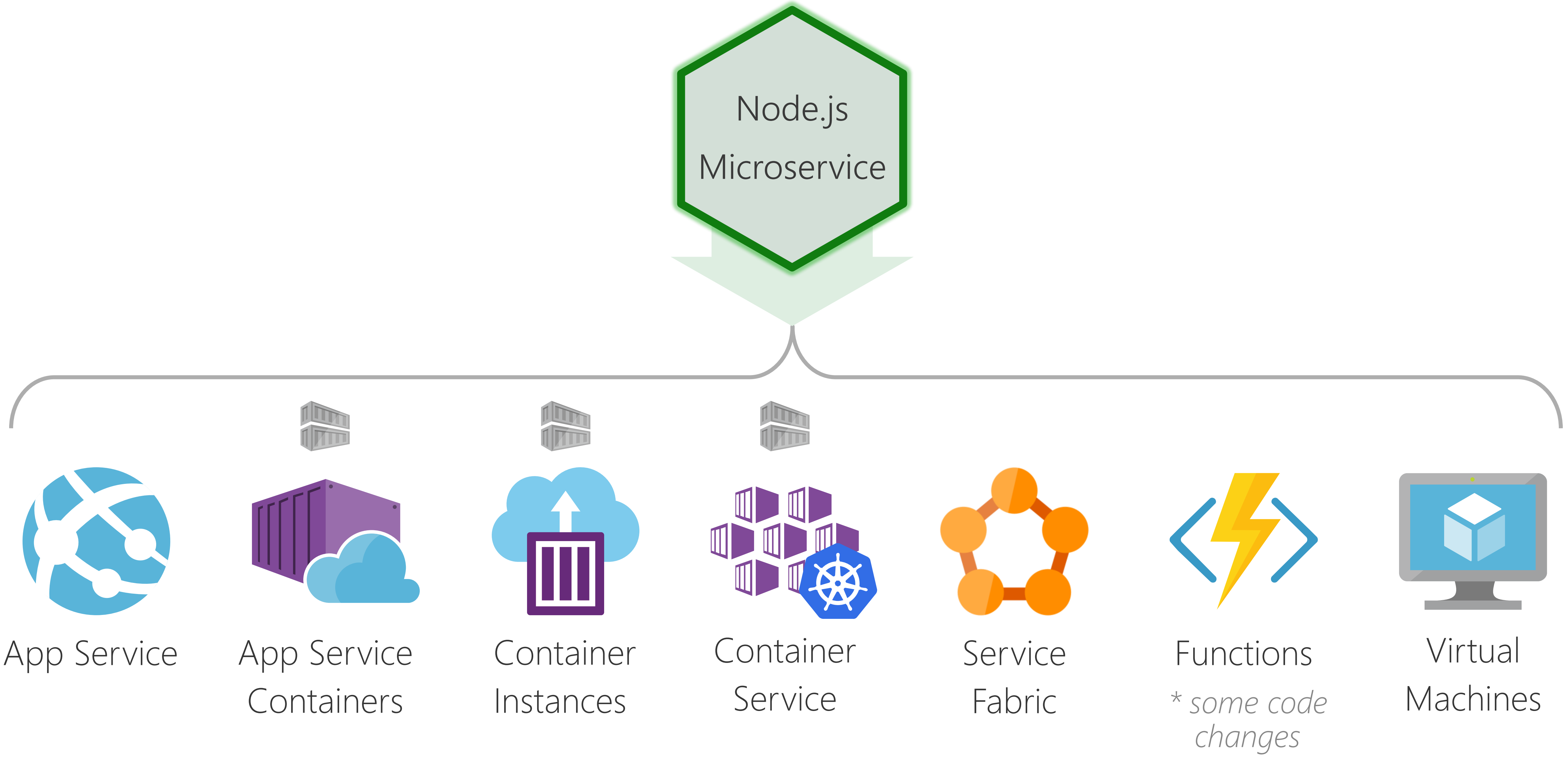


```
Event {
  id:     any       // 5 character random UID string
  title:  string    // Title of the event, 50 char max
  type:   string    // Type of event ['event', 'workshop', 'hack', 'lab']
  start:  Date      // Start date, an ISO 8601 string; YYYY-MM-DD
  end:    Date      // End date, an ISO 8601 string; YYYY-MM-DD
  topics: Topic[];  // Array of Topic objects, must be at least one
}
```

```
Topic {
  id:   number  // Integer
  desc: string  // Description
}
```

```
Feedback {
  id:       number  // 12 character random UID string
  event:    string  // Event id
  topic:    number  // Topic id
  rating:   number  // Feedback rating 1 to 5
  comment:  string  // Feedback comments
}
```

Azure Compute Hosting Options



Development Story - UI / Frontend

- Picked Angular - Over React and Vue.js
    - Considering switching to Vue.js
- Used Angular CLI project structure
- UI Framework - Bootstrap v3, tried and tested
    - Google Material considered - too early/beta state

- In memory DB & API for rapid development
- Moved from Angular v4 to v5
- Angular CLI
    - No webpack hell
    - Hides complexity

- Runtime configuration difficult
    - Config service
- Angular.js v1 results when searching for problems
- Angular CLI build is slow
- Docs hit & miss

Development Story - Microservices

- Picked Node.js – Very well known and stable
- Express web framework - Over Koa or Loopback or 100's others!
- Investigated YARN over NPM
  - Fast but tripped over some issues

?

- No issues moving from Node v6 to v8
  - v8 async & await
- Runs in containers very well, with simple Dockerfiles
- Works on Windows and Linux without change

- Deploying to Azure App Service requires some knowledge & prep
-

Development Story - State & Database

- Initial prototype used Azure Tables
  - JSON serialization, no means to query, no Functions trigger
- First moved to Cosmos DB with 'Document DB' API
- Finally - switched to MongoDB
  - Works under Cosmos DB and container based deployment

mongoDB

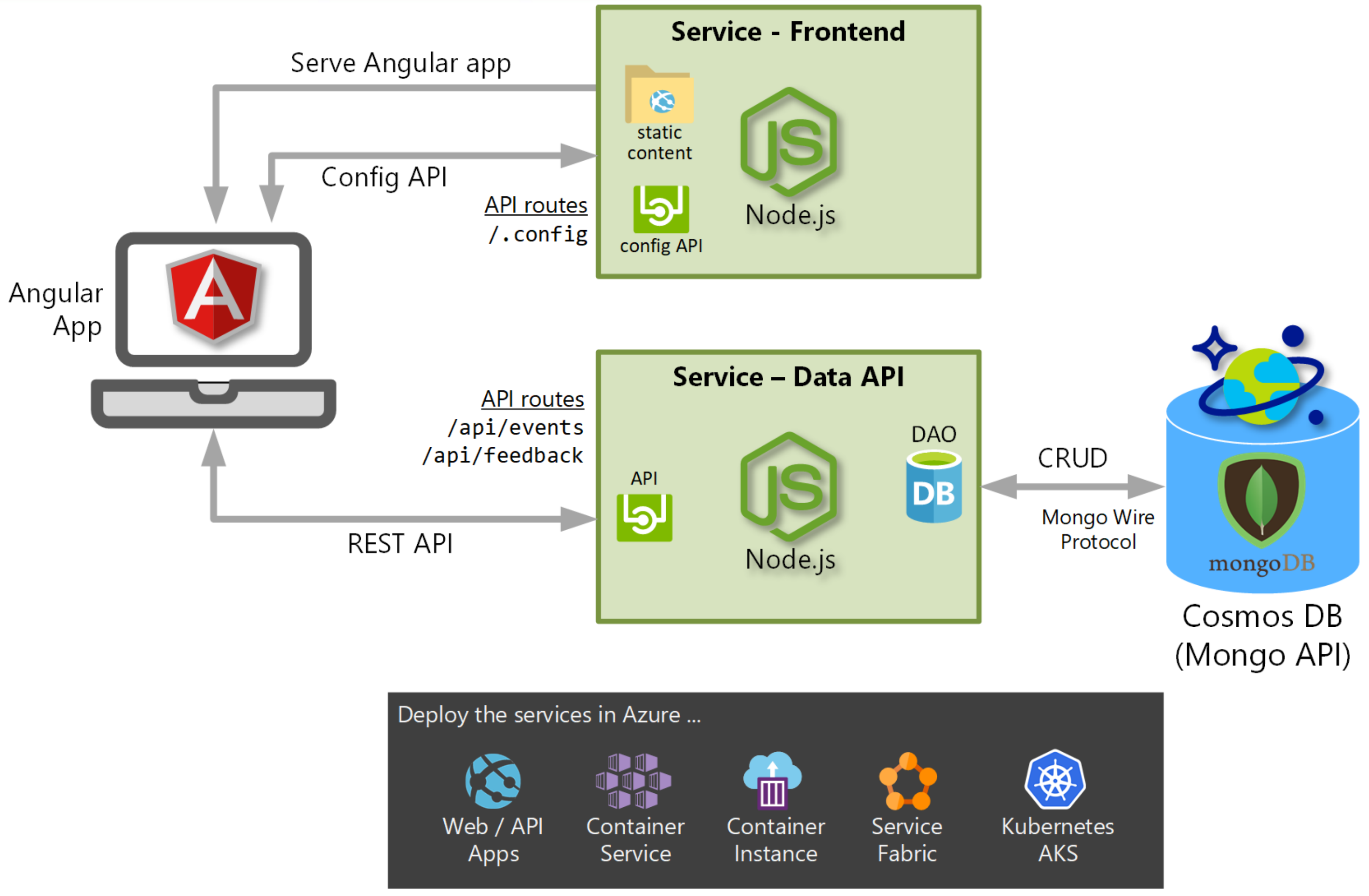
- Document DB & MongoDB are JSON native
  - Simplifies Data API middle tier
- MongoDB allows for more deployment scenarios
- Run local as container or in WSL

- Using multiple collections in Cosmos has cost implications
- Implications switching to a connection oriented DB

Try it out

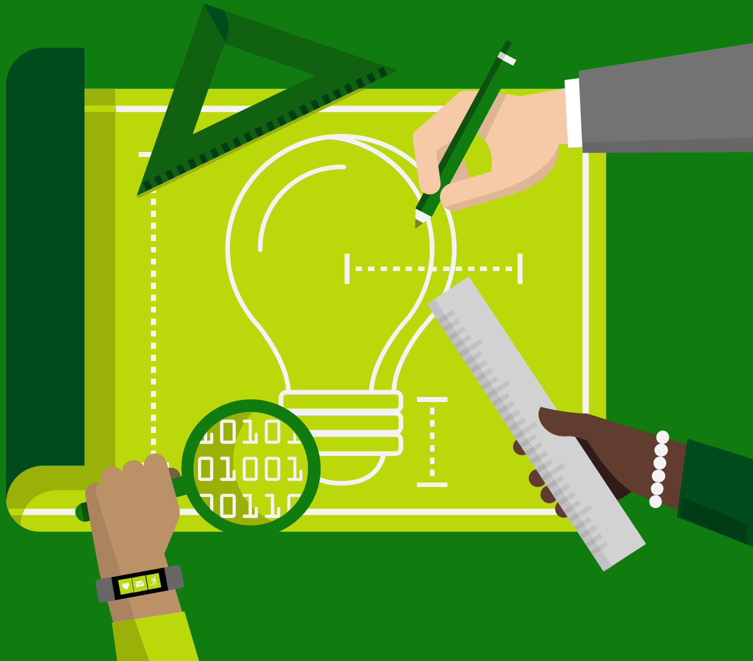
aka.ms/smilr-project

- Guides
- Source code
  - Angular
  - Node services
  - Functions
- ARM Templates
- Kubernetes deployment



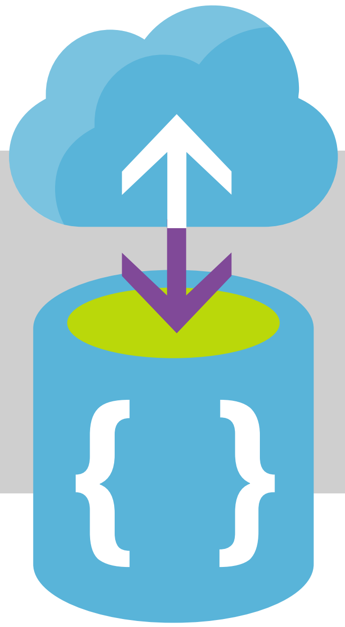
Microsoft ❤ Open Source

Deeper Dive



Angular - In Memory API

- Dummy API and database for Angular apps
- Allowed for rapid development of frontend design before the real Node REST service(s) were developed
- Not used when deployed in “production” mode



- Represents simple static JSON objects as a fake DB
- Intercepts AJAX (XHR / Fetch) calls and responds like a REST API
- Supports GET, POST, PUT, DELETE

https://github.com/angular/in-memory-web-api

Frontend / UI Service

Just 7 lines of code

- Uses Express
- Simply serves files as static content
- Redirect requests to index.html

```
var express = require('express');
var app = express();

// Serve all static content (index.html, js, css, images, etc.)
app.use('/', express.static(__dirname));

// Redirect all other requests to Angular app index.html,
app.use('*', function(req, res) {
  res.sendFile(`${__dirname}/index.html`);
});

// Start the server
var server = app.listen(process.env.PORT || 3000);
```

https://angular.io/guide/deployment

Frontend / UI Service - Config API

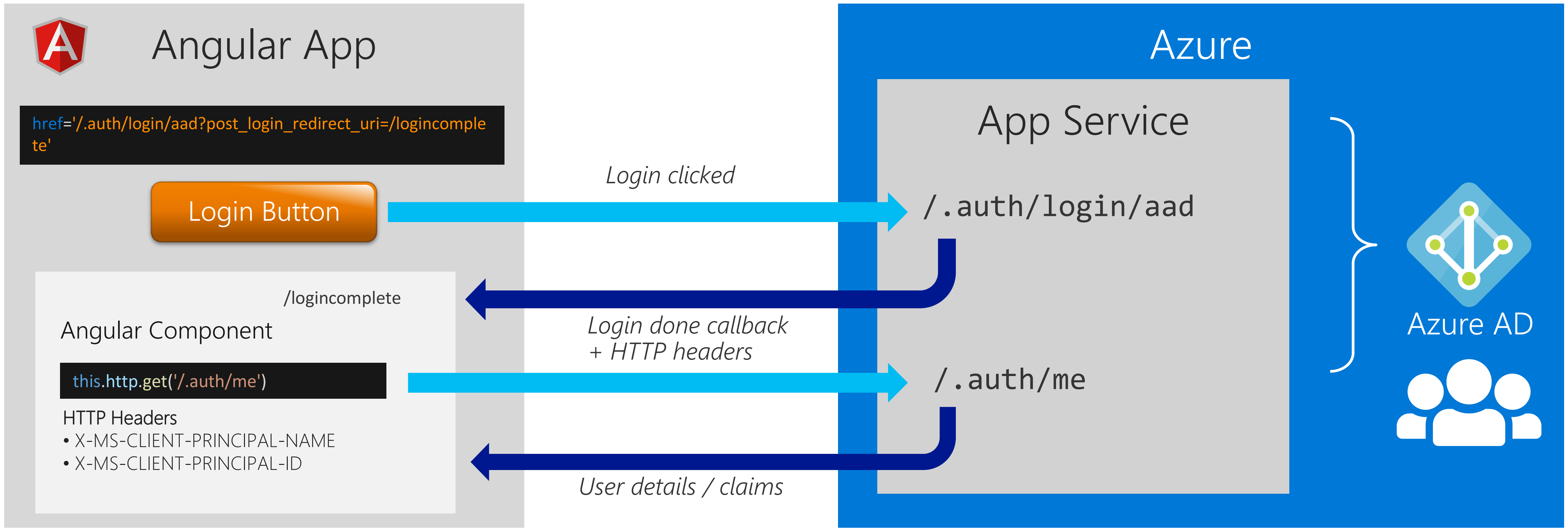
- **Problem** - Angular executes 100% on the client browser.
  How can we dynamically configure settings (e.g. API endpoint address) without code changes and rebuilding?
  (Note. This is a problem for all SPA frameworks, e.g. React, Vue.js etc)

- **Solution -** Very basic "micro API" on server allowing Angular code to fetch values at startup
  - Lets Angular client side get settings from the server
  - Environmental variables
  - Returns values in JSON

```
//
// MICRO API allowing dynamic configuration of the client/browser side Angular
// Allow Angular to fetch a comma separated set of environmental vars from the server
//
app.get('/.config/:keypairs', function (req, res) {
  let data = {};
  req.params.keypairs.split(",").forEach(varname =>{
    data[varname] = process.env[varname];
  })
  res.send(data);
});
```

Authorization

- Enabled with feature toggle in Angular app
- Requires front-end service to be hosted in Azure App Service
- Uses turn key 'App Service Authorization' feature
- Enabled, but anonymous access allowed



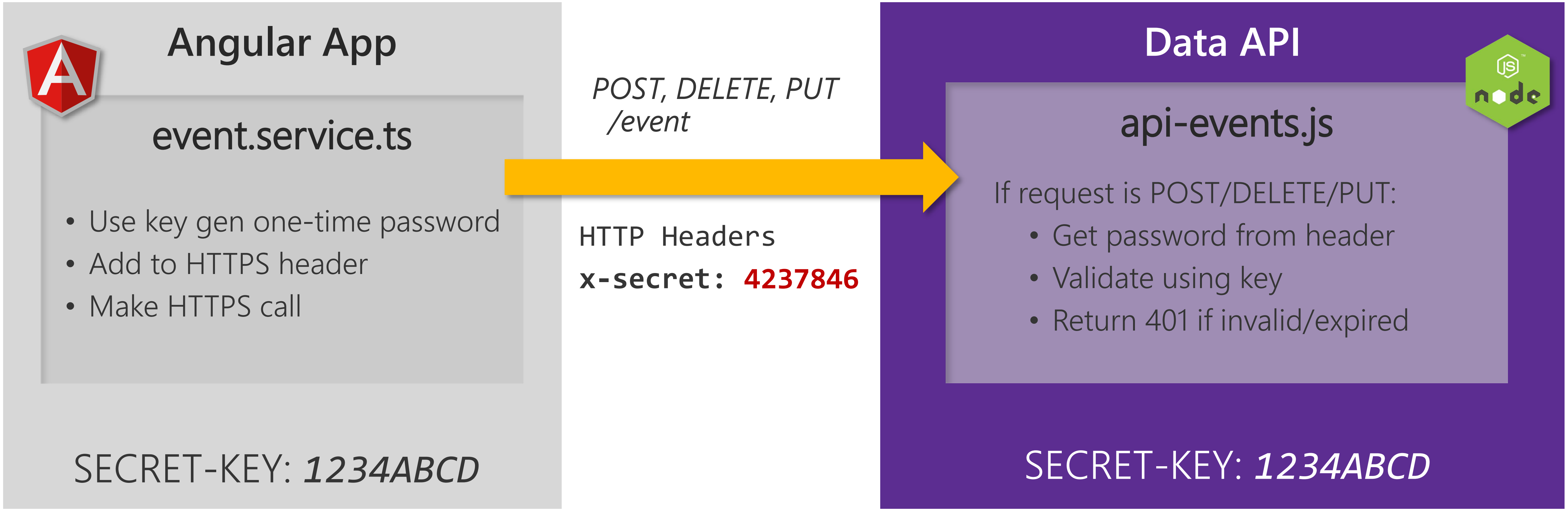
```
href='/.auth/login/aad?post_login_redirect_uri=/logincomplete'
```

```
this.http.get('/.auth/me')
```

HTTP Headers
- X-MS-CLIENT-PRINCIPAL-NAME
- X-MS-CLIENT-PRINCIPAL-ID

Securing the Data API

- Event admin operations (POST, DELETE, PUT) needed securing
- Using 'Time-based One Time Passwords' (TOTP)
- A pre-shared key, generates passwords valid for 30 seconds



- Use key gen one-time password
- Add to HTTPS header
- Make HTTPS call

SECRET-KEY: *1234ABCD*

```
HTTP Headers
x-secret: 4237846
```

If request is POST/DELETE/PUT:

- Get password from header
- Validate using key
- Return 401 if invalid/expired

SECRET-KEY: *1234ABCD*

Microsoft