Moving from Anarchy to Usability and Sustainability

moving user-facing applications from *startup* mode to *enterprise* mode

Peter Milne, Technology Architect  
@helipilot50  
helipilot50
Agenda

- Definitions
- The Problem
- The Solution
  - Front-end
  - Back-end
  - Production
- Uncle Pete’s advice - People
Anarchy

- **Anarchy** is the condition of a society, entity, group of people, or a single person that exists *without hierarchy*

- **Anarchism** is a political philosophy that advocates stateless societies based on voluntary associations.

- In practical terms, anarchy can refer to the curtailment or abolition of traditional forms of government and organizations.

**Startups** begin with a form of **benevolent Anarchy**
Sustainability

- **Sustainability** (from *sustain* and *ability*) is defined as a socio-ecological process characterized by the pursuit of a common ideal. An ideal is by definition unattainable in a given time and space. However, by persistently and dynamically approaching it, the process results in a sustainable system.

- In ecology, **sustainability** is the property of biological system to remain diverse and productive indefinitely.

---

In business, sustainability is when the **product or service remains viable indefinitely**.
Usability

• **Usability** is the ease of use and learnability of a human-made object such as a tool or device.

• In software engineering, **usability** is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.

Easy to use and people want to use it
The Problem
The Journey – Startup to Global Enterprise

“Once in a career opportunity”

“The journey is as important as the destination”

Basement Hackers

Worldwide influencers
Problem

- **Inconsistent** User Experience
- **130+ different** applications
- **Multiple** sign-on
- **Every** JavaScript framework *ever invented*
- **Multiple** JavaScript dialects
- **No common**
  - UI components
  - Features
- **Little shared**
  - Code
  - Services
- **400 (public) 650 (internal)** API endpoints with *unique* dialects
What We Have

- Bag of applications/stuff
- Feature on **Features** on Feature…
- Poor user experience

But it works

- Makes money
- Usable
User Facing Applications
All Are Functional - All Are Different
Navigation map of just one application
Products/Organization Reached a Crossroad…

Anarchy vs Usability & Sustainability
Goals of Reusability and Sustainability

Provide business with a Consistent user experience

• A single page web application
• Modular components promoting velocity and reuse
• Other UI technologies
  - mobile device
  - Alexa, Siri, etc
  - Augmented reality
  - ... things not invented yet

Provide Development with Common set of

• Tools
• Framework(s)
• Language(s)
• Data
• Patterns
• Infrastructure

Portable Software Engineers

• Easier move from between projects
• Shorter “new guy” time
The solution
Solution

New User Experience facilitated by:

• Front-end
  - Single web page application
  - Common components
  - Single Sign-on
  - Reusable problem solvers (Applets)
  - Declarative navigation (Workflows)

• Back-end:
  - Single API end-point (Apollo server)
  - Single dialect (GraphQL)
  - Modular schema describing Entities
Requirements and Personas

**Personas** written to describe the users and what they need from the applications to do their job

... a persona is similar to a **UML Actor**

**Requirements** to be based on personas

... requirements are similar to **UML Usecases**
<table>
<thead>
<tr>
<th>Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• React</td>
</tr>
<tr>
<td>• GraphQL</td>
</tr>
<tr>
<td>• Apollo</td>
</tr>
<tr>
<td>• Node.js</td>
</tr>
<tr>
<td>• .NET core</td>
</tr>
<tr>
<td>• RabbitMQ</td>
</tr>
<tr>
<td>• OAuth 2.0 an OpenID Connect</td>
</tr>
<tr>
<td>• Open API - Swagger 2.0</td>
</tr>
<tr>
<td>• Docker</td>
</tr>
<tr>
<td>• Kubernetes</td>
</tr>
</tbody>
</table>
Front-end Components, Applets and Workflows
Single Page Web => One Application
Reusable UX Components

Component
React

Composite Components
React

Applet
- React HOC
- Entity
- GraphQL

Workflow
- Applets
- Workflows
  - Finite State Machine
  - Workflow engine

Complexity
Applet Example: Microsoft Word

Components of a word document

• Paragraph
• Style
• Excel
• Picture

Think OLE/ActiveX
Workflow Example: Consumer Experiences

Insert photo

Pick folder

Choose image

Confirm

Done!

Insert photo

Pick folder

Choose image

Confirm

Done!
Applets Are...

Small **problem solvers**
- **Human** interaction
- Focused on **limited problem**
- **Ignorant** of who call it
- **Apathetic** of what is done with the result

All **new functionality** built using Applets
Applet

- React High Order Component
- Renders one type of entity
- Uses GraphQL to Query, Mutate and Subscribe to entities
- Invoked “similar” to a promise

```java
class AuditLogListPage extends Component {
  static appletName = 'OAC-APPLET-AUDITLOGLIST-EXAMPLE';
  ...
}

render() {
  return (
    <div>
      <FlexCol spacing="m">
        <Card>
          <Title>Open Audit Log Applet</Title>
          <Content>
            <FlexCol spacing="m" padding="m" style={{ background: '#e7e9ec' }}>
              <Button type="primary" onClick={this.showAuditLogDialog}>Show Logs</Button>
            </Content>
        </Card>
      </FlexCol>
    </div>
  );
}
```
Workflows

A Workflow is a reusable business scenario (use case)

The **Glue that connects applets** into usable scenarios
Workflow - FSM

Workflows are as finite state machine where:

Applets (what the user sees) are the states,

State transitions are user navigation.
Declarative Workflow

- Finite state machine defined in JSON
- Starts with an initial state
- Applets emit events to stimulate transitions
Back-end
Access to Entities
Business Entity model => GraphQL Schema
GraphQL Schema Definition

- **Declarative Schema**
- **Define**
  - **Types** (entities)
  - **Queries** (reads)
  - **Mutations** (writes)
  - **Subscriptions** (events)
import { gql } from 'react-apollo';

export const AuditLogQuery = gql`
query audit {
  changes(limit: 5, orderDirection: descending) {
    id
    entityId
    userId
    companyId
    companyType
    entityType
    changeType
    changeTime
    product
    changes {
      type
      propertyName
      propertyType
      before
      after
    }
  }
}
``

export default AuditLogQuery;

Applet using a GraphQL Query

The applet makes a query using the Apollo GraphQL client

Single endpoint – Single Protocol
**Schema Server - GraphQL**

- **Single business entity model** implemented with GraphQL
- **Transform the inconsistencies** of our current API suite to a consistent view
- **Easy versioning/deprecation**
  - Migration rather than revolution
- Implemented in **Node.js as a microservice**
- Constructed with **Schema Modules**
- Production code and HA deployment in development
  - L7 load balancer, Kubernetes and Containers
- Secured with **OAuth 2.0 scopes**
Four (4) types

1. **REST** Module – encapsulates existing Adform API
   - Schema Module tool generates 85-100%

2. **Bespoke** Module – hand coded
   - Schema module tool generates an template

3. **Relation** Module – Defines relationships (graphs) to REST or Bespoke modules.
   - Schema module tool generates an template and you extend types and add resolvers to create relationship

4. **Remote Schema** Module – Remote GraphQL API
   - Super easy to include in the schema server
Middleware Subsystems

- **Middleware server**
  - A fancy proxy
  - OAuth 2.0
  - Other common things…

- **Schema Server**
  - Provides access to data

- **Configuration Registry**
  - System and User specific configuration

- **State Server**
  - Node.js, Apollo and Express
Production
DevOps
Dev Pipeline

- **GitHub Enterprise**
  - Repository for *EVERYTHING*

- **Drone**
  - CI/CD for docker

- **Helm**
  - Kubernetes package manager

- **Artifactory** – Private “binary” repository
  - NPM
  - Maven
  - NuGet
  - …everything
Containers

Containers provide

- Isolation (cgroups)
- Resource limiting
- Control
- Prioritization

Technologies

- .NET core
- Node
- JVM (Scala/Java)
Orchestration

- **Kubernetes**
  - Common cluster
  - Bare metal machines
  - 3 data centres

- **Helm** Charts
  - Package manager for Kubernetes

```
image: docker.artifactory.adform.com/aap/schema-ser
# tag: stable # SET BY DRONE CI
pullPolicy: IfNotPresent

service:
type: ClusterIP
internalPort: 9010 # for Pods
externalPort: 9010 # for Service

ingress:
enabled: true
path: /
hosts:
- aap-schema-server.d1.adform.zone
- aap-schema-server.devkube.d1.adform.zone
tls:
- hosts: [aap-schema-server.devkube.d1.adform.zone]
```
Common services

Provided by **DevOps**

- Authorisation
- Logging
- Monitoring
- Metrics
- … others
Uncle Pete’s Advice: People
Software Engineers

- Create something from nothing
- Have an opinion – and think it is right
- Have favourite technologies
- Very passionate
- Love to argue
- Are more creative than
  - Musicians
  - Artists
  - Writers
DevOps Engineers

Like Scotty on the Enterprise

Legends
Every 2 - 5 years there is a New Fashion

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>Cobol</td>
<td>Cobol</td>
<td>Cobol</td>
<td>C++</td>
<td>Java</td>
</tr>
<tr>
<td>Cobol</td>
<td>Pascal</td>
<td>Basic</td>
<td>Smalltalk</td>
<td>Javascript</td>
<td>JavaScript</td>
</tr>
<tr>
<td>Fortran</td>
<td>PL1</td>
<td>C</td>
<td>Perl</td>
<td>Python</td>
<td>Python</td>
</tr>
<tr>
<td>Lisp</td>
<td>Forth</td>
<td>RPG</td>
<td>Visual Basic</td>
<td>PHP</td>
<td>PHP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delphi</td>
<td>C#</td>
<td>C#</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOOL</td>
<td>Objective C</td>
<td>Objective C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C#</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Groovy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Rust</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Swift</td>
</tr>
</tbody>
</table>
Technology Is Additive – Example: Memory

- NV DIMM
- Nand Memory
- TTL/CMOS Static & Dynamic
- Discrete flip flops
- Magnetic Core
- Mercury ripple tank
- Relays
Things Change

• Don’t expect things to be constant
  - Technology
  - Tools
  - Use cases

• Accept
• Adapt
• Re-train
Questions?

@helipilot50