Asynchronous Single Page Applications without a Line of HTML or Javascript.

Or why D is just awesome

Robert ”burner” Schadek

May 5, 2016

DConf

Javascript and HTML are awesome

- No explicit types
- Variables may be undefined
- Repetition repetition repetition
- No templates
- No compile time function execution
- No compile time

Goals

- Retain type-information of data from DB to Client's Browser

Goals

- Retain type-information of data from DB to Client's Browser
- Keeping things DRY

Goals

- Retain type-information of data from DB to Client's Browser
- Keeping things DRY
- Performance

Goals

- Retain type-information of data from DB to Client's Browser
- Keeping things DRY
- Performance
- Get things done

Vibe.d and Diet

Vibe.d

- Powerful asynchronous I/O and web toolkit for D
- Uses Fibers and Threads

Vibe.d

- Powerful asynchronous I/O and web toolkit for D
- Uses Fibers and Threads
  - $n \rightarrow m$ mapping
  - `yield()`
  - async IO

Vibe.d

- Powerful asynchronous I/O and web toolkit for D
- Uses Fibers and Threads
  - $n \rightarrow m$ mapping
  - `yield()`
  - async IO
- async DB connectivity for MongoDB, Redis, MySQL
- You don't need to care about async
- Web interface generator
- REST interface generator

REST Interface Generator

```
interface MyAPI {
    // GET /weather -> responds {"text": "...", "
        ↪ temperature": ...}
    Weather getWeather();
}
```

REST Interface Generator

```
interface MyAPI {
    // GET /weather -> responds {"text": "...", "
        ↪ temperature": ...}
    Weather getWeather();
}

class MyAPIImplementation : MyAPI {
    auto weather = [ "sunny", "rainy", "cats and dogs"
        ↪ , "snow" ];

    Weather getWeather() {
        return Weather(
            weather[uniform(0, weather.length)],
            uniform(-10,30)
        );
    }
}
```

REST Interface Generator

```
auto router = new URLRouter;
router.get("/", staticTemplate!"index.dt");
router.get("/main.html", staticTemplate!"main.dt");

router.registerRestInterface!MyAPI(new
    ↪ MyAPIImplementation, restsettings);
```

REST Interface Generator

```
auto router = new URLRouter;
router.get("/", staticTemplate!"index.dt");
router.get("/main.html", staticTemplate!"main.dt");

router.registerRestInterface!MyAPI(new
    ↪ MyAPIImplementation, restsettings);



struct Weather {
    string text;
    double temperature;
}
```

AngularJS Typescript

```
interface Weather {
    text : string,
    temperature : number
}
```

AngularJS Typescript

```
interface Weather {
    text : string,
    temperature : number
}

interface MainScope extends ng.IScope {
    weather: Weather
}
```

AngularJS Typescript

```
interface Weather {
    text : string,
    temperature : number
}

interface MainScope extends ng.IScope {
    weather: Weather
}

class MainCtrl {
    public static $inject = [ '$scope', '$http' ];

    constructor(private $scope: MainScope,
                private $http : ng.IHttpService)
    {
        this.weather();
    }
```

AngularJS Typescript

```
weather() : void {
    var s = '/weather';
    this.$http.get(s).success((data : Weather) => {
        this.$scope.weather = data;
    });
}
```

Diet

```
doctype html
html(ng-app="myapp")
    head
        title DConf 2016 Weather
        - javascript("./angular.js");
        - javascript("./angular-route.js");
        - javascript("./myapp.js");

    body
        div(ng-view)
```

Diet

```
doctype html
html(ng-app="myapp")
    head
        title DConf 2016 Weather
        - javascript("./angular.js");
        - javascript("./angular-route.js");
        - javascript("./myapp.js");

    body
        div(ng-view)

- void javascript(string name)
    script(src="#{name}")
```

Diet

```
.container
    p {{weather.text}}
    p {{weather.temperature }}

    button(type="submit",ng-click="ctrl.weather();")
        ↪ Get Weather
```

Live Demo

Dataflow from the Server to the Frontend and back again

Dataflow from the Server to the Frontend and back again

```d
struct Weather {
    string text;
    double temperature;
}
```

Dlang

Dataflow from the Server to the Frontend and back again

```d
struct Weather {
    string text;
    double temperature;
}
```

Dlang

```typescript
interface Weather {
    text : string,
    tempereture : number
}
```

Typescript

dstructtotypescript

```
dstructtotypescript -i weather.d -p weather.ts -s
    ↪ Weather
```

dstructtotypescript

```
dstructtotypescript -i weather.d -p weather.ts -s
    ↪ Weather
```

```
struct Weather {
    string text;
    double temperature;
}
```

dstructtotypescript

```
dstructtotypescript -i weather.d -p weather.ts -s
    ↪ Weather
```

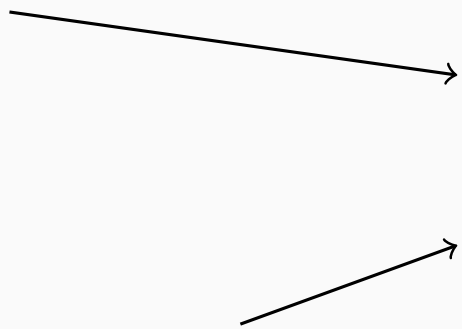
```d
struct Weather {
    string text;
    double temperature;
}
```

```d
import std.format;

foreach(it; __traits(allMembers, Weather))
```

dstructtotypescript

```
dstructtotypescript -i weather.d -p weather.ts -s
    ↪ Weather
```

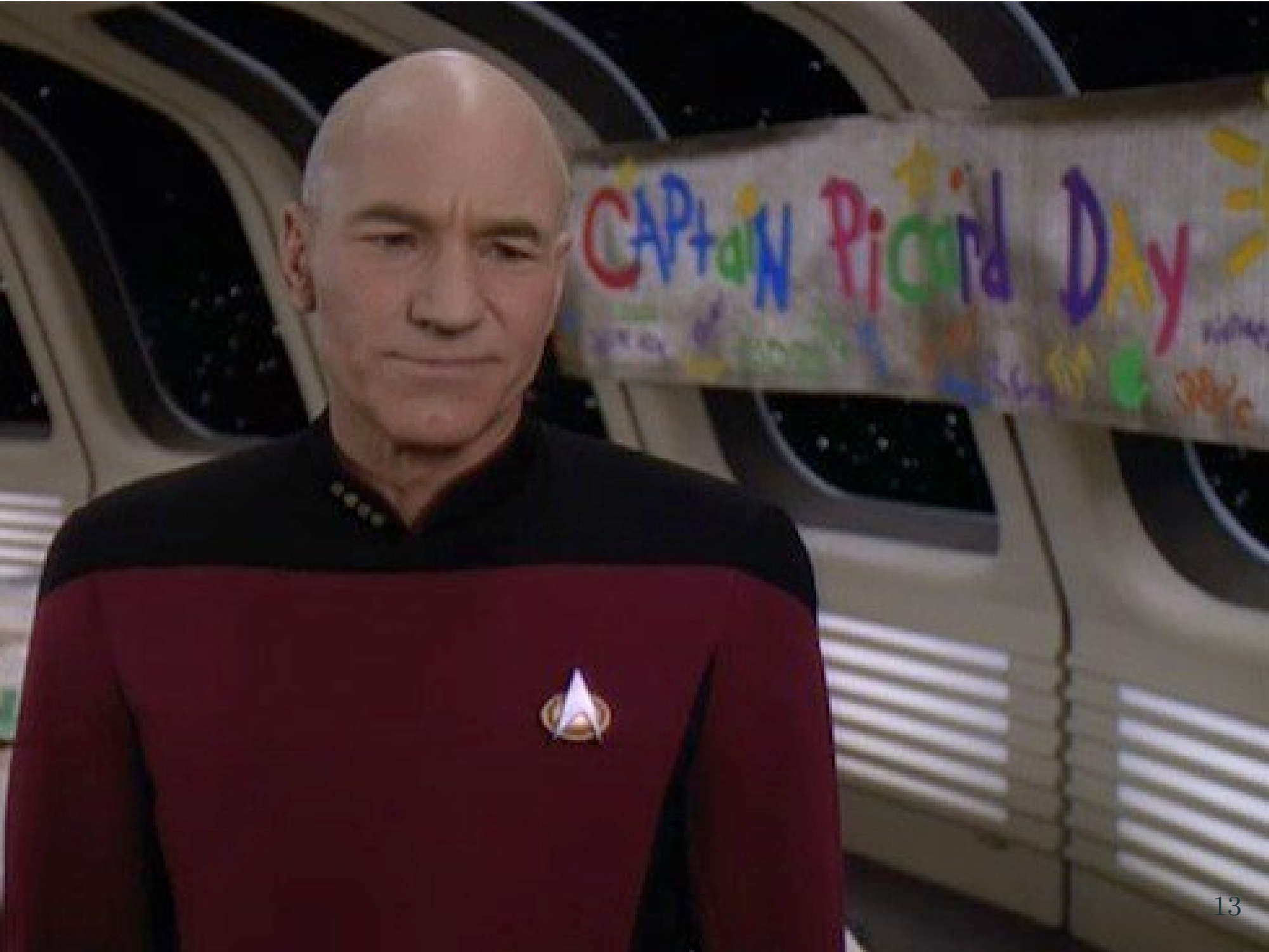
```
struct Weather {
    string text;
    double temperature;
}
```



```
interface Weather {
    text : string,
    temperature : number
}
```

```
import std.format;

foreach(it; __traits(allMembers, Weather))
```

Everything is wrong


CAPtaiN Picard DAy

Everything is wrong

- All we solved was a tiny specific problem
- What about server to database
- What if we would use Dart instead of Typescript
- How do we communicate the overall architecture
- How do we keep the architecture in sync with the code
- How do we communicate with non-developer
- ...

Everything is wrong

- All we solved was a tiny specific problem
- What about server to database
- What if we would use Dart instead of Typescript
- How do we communicate the overall architecture
- How do we keep the architecture in sync with the code
- How do we communicate with non-developer
- ...
- How do we deal with change?

Everything is still wrong

- Waterfall Model

Everything is still wrong

- Waterfall Model ← no change, never

Everything is still wrong

- Waterfall Model ← no change, never

- Hacking

Everything is still wrong

- Waterfall Model ← no change, never

- Hacking ← no plan to speak of, just change

Everything is still wrong

- Waterfall Model ← no change, never

- Agile Methods
- Hacking ← no plan to speak of, just change

Everything is still wrong

- Waterfall Model ← no change, never

- Agile Methods ← just hacking, with fancy names
- Hacking ← no plan to speak of, just change

Everything is still wrong

- Waterfall Model ← no change, never
- UML

- Agile Methods ← just hacking, with fancy names
- Hacking ← no plan to speak of, just change

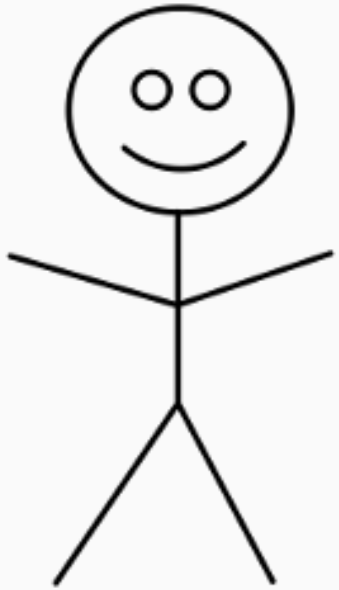
Everything is still wrong

- Waterfall Model ← no change, never
- UML ← just kill me already

- Agile Methods ← just hacking, with fancy names
- Hacking ← no plan to speak of, just change

Everything is still wrong

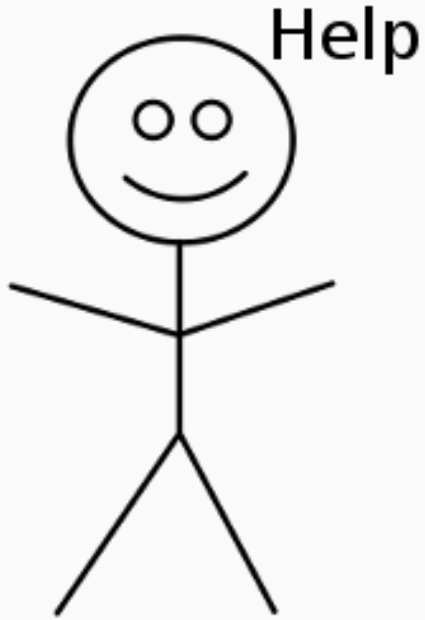
- Waterfall Model ← no change, never
- UML ← just kill me already



- Agile Methods ← just hacking, with fancy names
- Hacking ← no plan to speak of, just change

Everything is still wrong

- Waterfall Model ← no change, never
- UML ← just kill me already



- Agile Methods ← just hacking, with fancy names
- Hacking ← no plan to speak of, just change

What do we want

- Speak about the system at different levels of detail with different people
- Quickly introduce people to the system
- Keep data classes (Model) synchronized across
  - Frontend
  - Server
  - Database
- Write only one model for everything, to keep stuff in sync
- Have description line based, because git
- Generate everything possible from the model

Introducing the C4 Architecture

System Context

The system plus users
and system dependencies

Containers

The overall shape of the architecture
and technology choices

Components

Logical components and their
interactions within a container

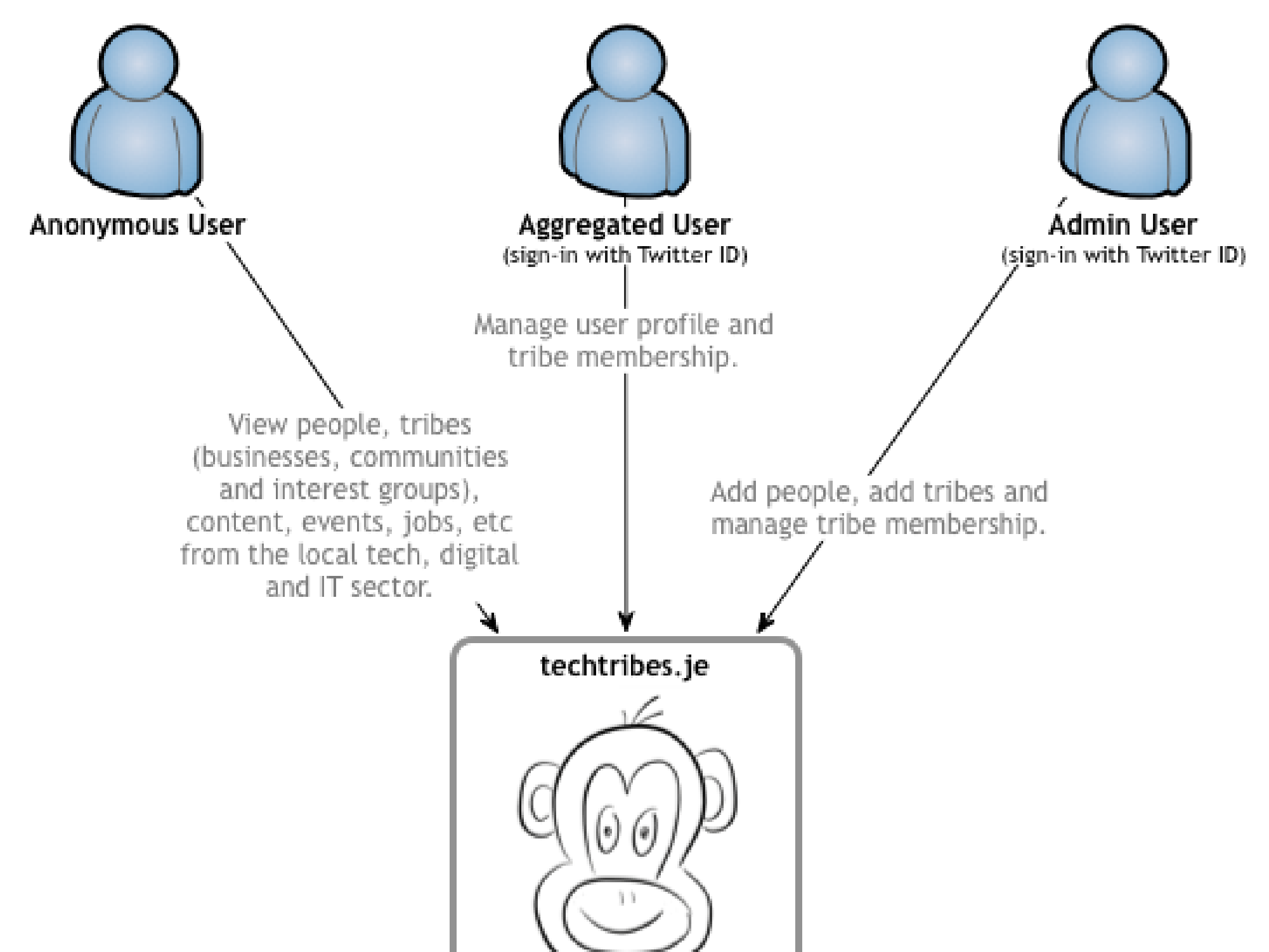
Classes

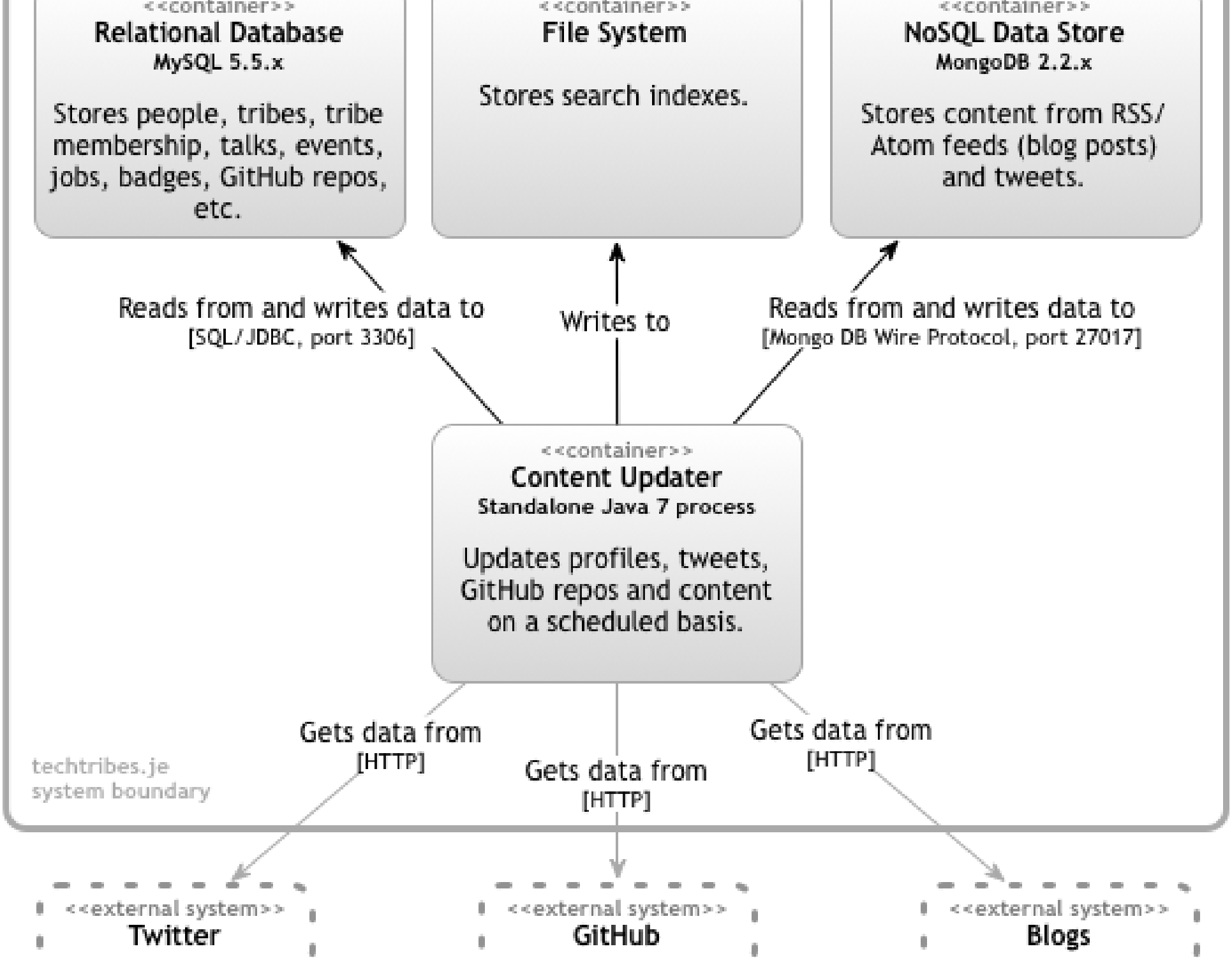
Component or pattern
implementation details

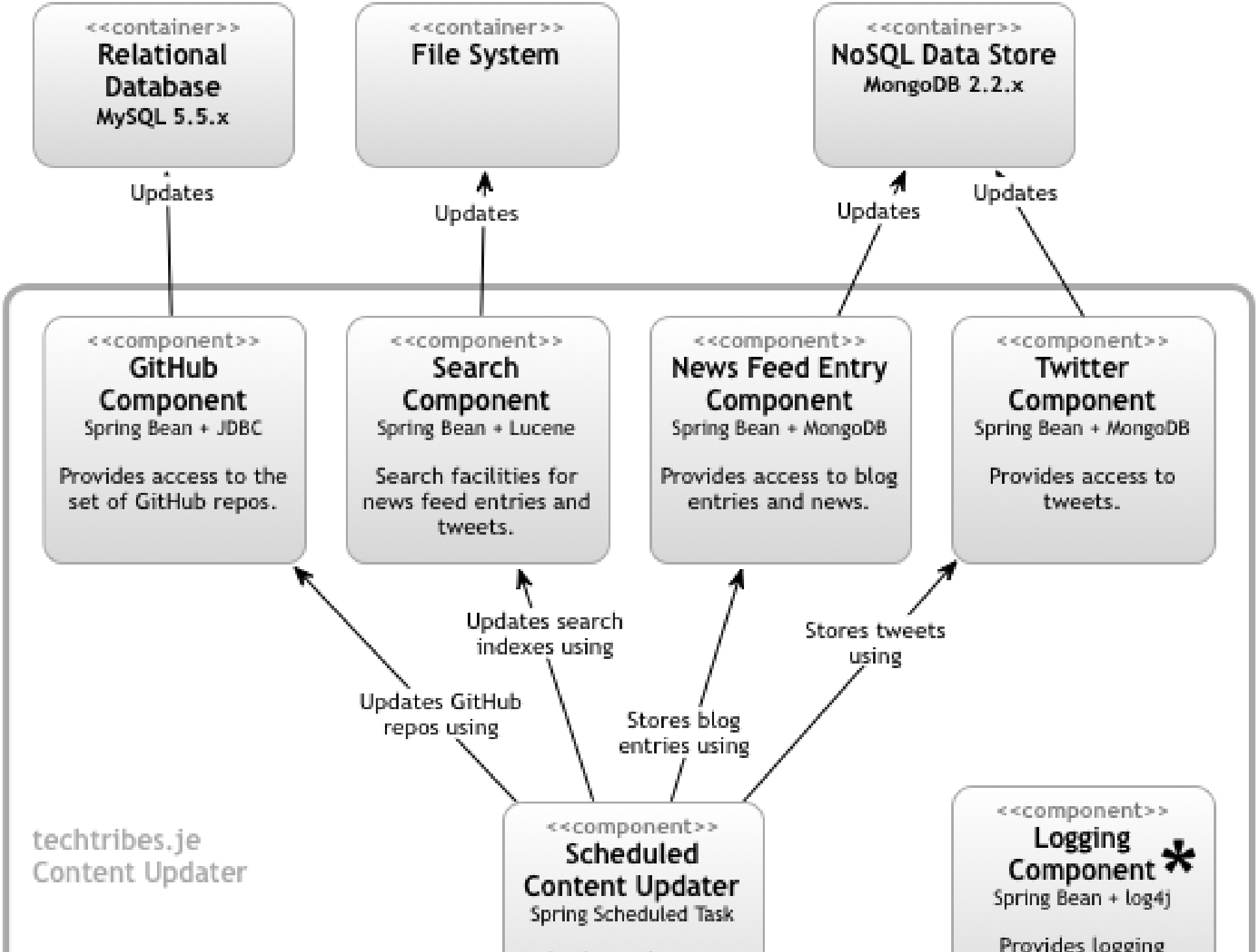
Overview first

Zoom and filter

Details on demand


Anonymous User
Aggregated User
(sign-in with Twitter ID)
Admin User
(sign-in with Twitter ID)
Manage user profile and tribe membership.
View people, tribes (businesses, communities and interest groups), content, events, jobs, etc from the local tech, digital and IT sector.
Add people, add tribes and manage tribe membership.
techtribes.je


<<container>>
Relational Database
MySQL 5.5.x
Stores people, tribes, tribe membership, talks, events, jobs, badges, GitHub repos, etc.
<<container>>
File System
Stores search indexes.
<<container>>
NoSQL Data Store
MongoDB 2.2.x
Stores content from RSS/ Atom feeds (blog posts) and tweets.
Reads from and writes data to
[SQL/JDBC, port 3306]
Writes to
Reads from and writes data to
[Mongo DB Wire Protocol, port 27017]
<<container>>
Content Updater
Standalone Java 7 process
Updates profiles, tweets, GitHub repos and content on a scheduled basis.
Gets data from
[HTTP]
Gets data from
[HTTP]
Gets data from
[HTTP]
techtribes.je
system boundary
<<external system>>
Twitter
<<external system>>
GitHub
<<external system>>
Blogs


<<container>>
Relational Database
MySQL 5.5.x
<<container>>
File System
<<container>>
NoSQL Data Store
MongoDB 2.2.x
Updates
Updates
Updates
Updates
<<component>>
GitHub Component
Spring Bean + JDBC
Provides access to the set of GitHub repos.
<<component>>
Search Component
Spring Bean + Lucene
Search facilities for news feed entries and tweets.
<<component>>
News Feed Entry Component
Spring Bean + MongoDB
Provides access to blog entries and news.
<<component>>
Twitter Component
Spring Bean + MongoDB
Provides access to tweets.
Updates GitHub repos using
Updates search indexes using
Stores blog entries using
Stores tweets using
techtribes.je
Content Updater
<<component>>
Scheduled Content Updater
Spring Scheduled Task
<<component>>
Logging Component
Spring Bean + log4j
Provides logging

Structurizr

- Implements C4 Architecture Model
- Java library to build the model

Structurizr

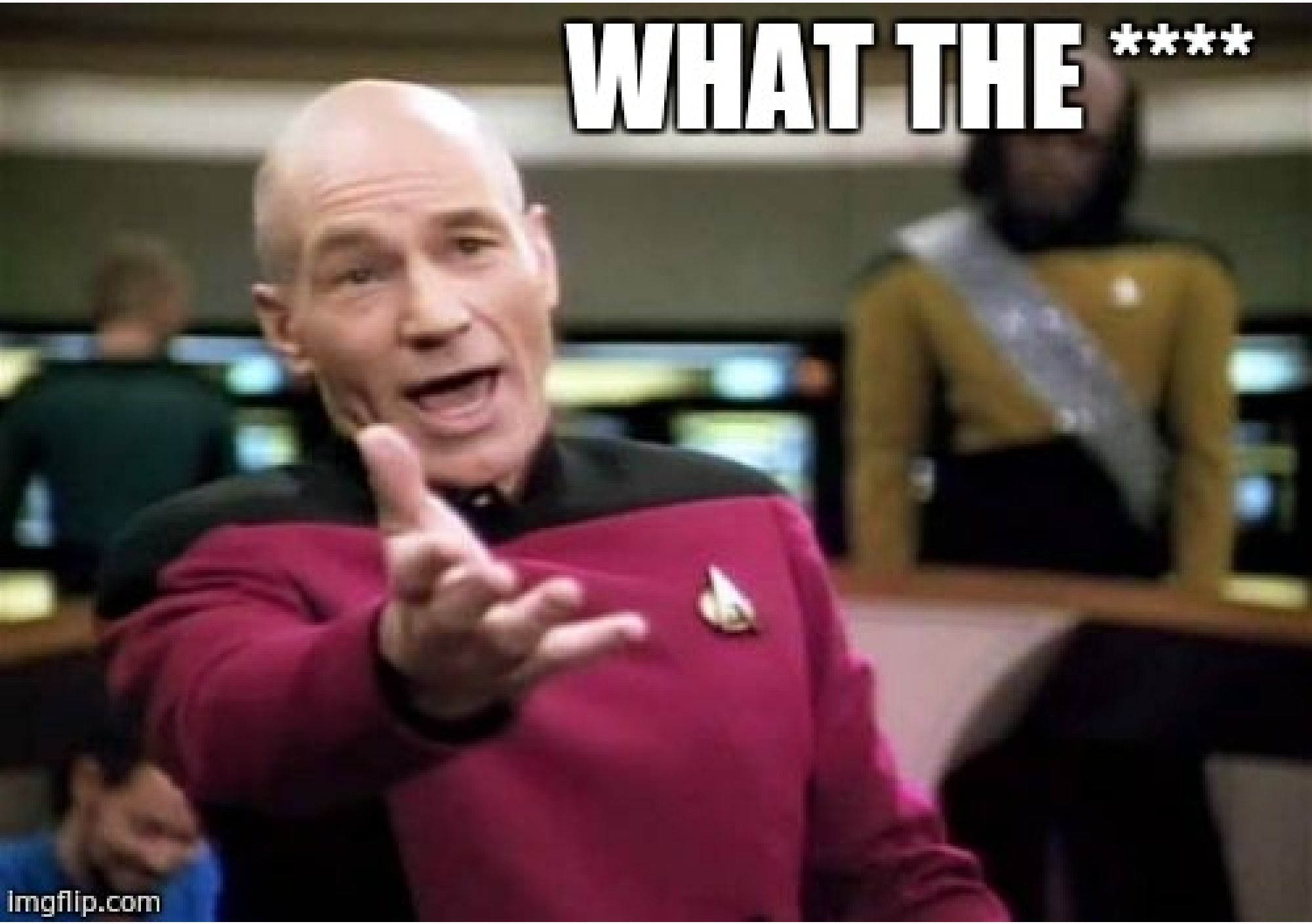
- Implements C4 Architecture Model
- Java library to build the model
  - Its code, its fits into git

Structurizr

- Implements C4 Architecture Model
- Java library to build the model
  - Its code, its fits into git
- Structurizr generates code

Structurizr

- Implements C4 Architecture Model
- Java library to build the model
  - Its code, its fits into git
- Structurizr generates code
- Only Java and .net


WHAT THE ****
imgflip.com

How hard can it be

How do we approach the development

- We’re not gonne create a new language

How do we approach the development

- We’re not gonne create a new language, at first

How do we approach the development

- We're not gonne create a new language, at first, most likely

How do we approach the development

- We're not gonne create a new language, at first, most likely
- The model (ast) is pretty simple

How do we approach the development

- We're not gonne create a new language, at first, most likely
- The model (ast) is pretty simple
  - The world
  - The world has
  - Actors, software/hardware systems
  - A software systems has
  - Containers (everything with a unique pid)
  - A container has
  - Components (think `module`) and classes
  - A component has
  - Components and classes
  - Classes have members
  - Connections between the above
    - UML Association, Aggregation, Composition, Dependency, ...
  - Additional informations, names, descriptions

Using Degenerator 1/2

```
auto world = new TheWorld("TheWorld");
Actor users = world.getOrNewActor("The Users");
users.description = "This is a way to long
    ↪ description for something "
     ~ "that should be obvious.";

auto system = world.getOrNewSoftwareSystem("
    ↪ AwesomeSoftware");
```

Using Degenerator 1/2

```
auto world = new TheWorld("TheWorld");
Actor users = world.getOrNewActor("The Users");
users.description = "This is a way to long
    ↪ description for something "
     ~ "that should be obvious.";

auto system = world.getOrNewSoftwareSystem("
    ↪ AwesomeSoftware");

Container frontend = system.getOrNewContainer("
    ↪ Frontend");
frontend.technology = "Angular";
auto frontendUserCtrl = frontend.getOrNewComponent(
    ↪ "frontUserCtrl");
```

Using Degenerator 2/4

```
auto database = system.getOrNewContainer("Database"
    ↪ );
database.technology = "MySQL";
world.getOrNew!Dependency("serverDatabase",
     server, database
).description = "CRUD";
```

Using Degenerator 2/4

```
auto database = system.getOrNewContainer("Database"
    ↪ );
database.technology = "MySQL";
world.getOrNew!Dependency("serverDatabase",
    server, database
).description = "CRUD";

Class user = getOrNewClass("User",
    frontendUserCtrl, serverUserCtrl, database
);
```

Using Degenerator 3/4

```
Class user = getOrNewClass("User",
    frontendUserCtrl, serverUserCtrl, database
);
```

Using Degenerator 3/4

```
Class user = getOrNewClass("User",
    frontendUserCtrl, serverUserCtrl, database
);

MemberVariable userId = user.getOrNew!
    ↪ MemberVariable("id");
userId.type = integer;
userId.addLandSpecificAttribute("MySQL", "PRIMARY
    ↪ KEY");
userId.addLandSpecificAttribute("MySQL", "AUTO
    ↪ INCREMENT");
```

Using Degenerator 4/4

```
Class address = getOrNewClass("Address",
    frontendUserCtrl, serverUserCtrl, database
);
```

Using Degenerator 4/4

```
Class address = getOrNewClass("Address",
    frontendUserCtrl, serverUserCtrl, database
);

Aggregation userAddress = world.getOrNew!
    ↪ Aggregation("addressUser",
    address, user
);
```

Types in Degenerator

- Types

Types in Degenerator

- Types e.g. strings are not always strings

Types in Degenerator

- Types e.g. strings are not always strings
  - D string
  - MySQL text
  - C++ std::string

Types in Degenerator

- Types e.g. strings are not always strings
  - D string
  - MySQL text
  - C++ std::string

```
struct Type {
    string name;
    string[string] typeMapping;
}

auto pwdHash = Type("PasswordString");
```

Types in Degenerator

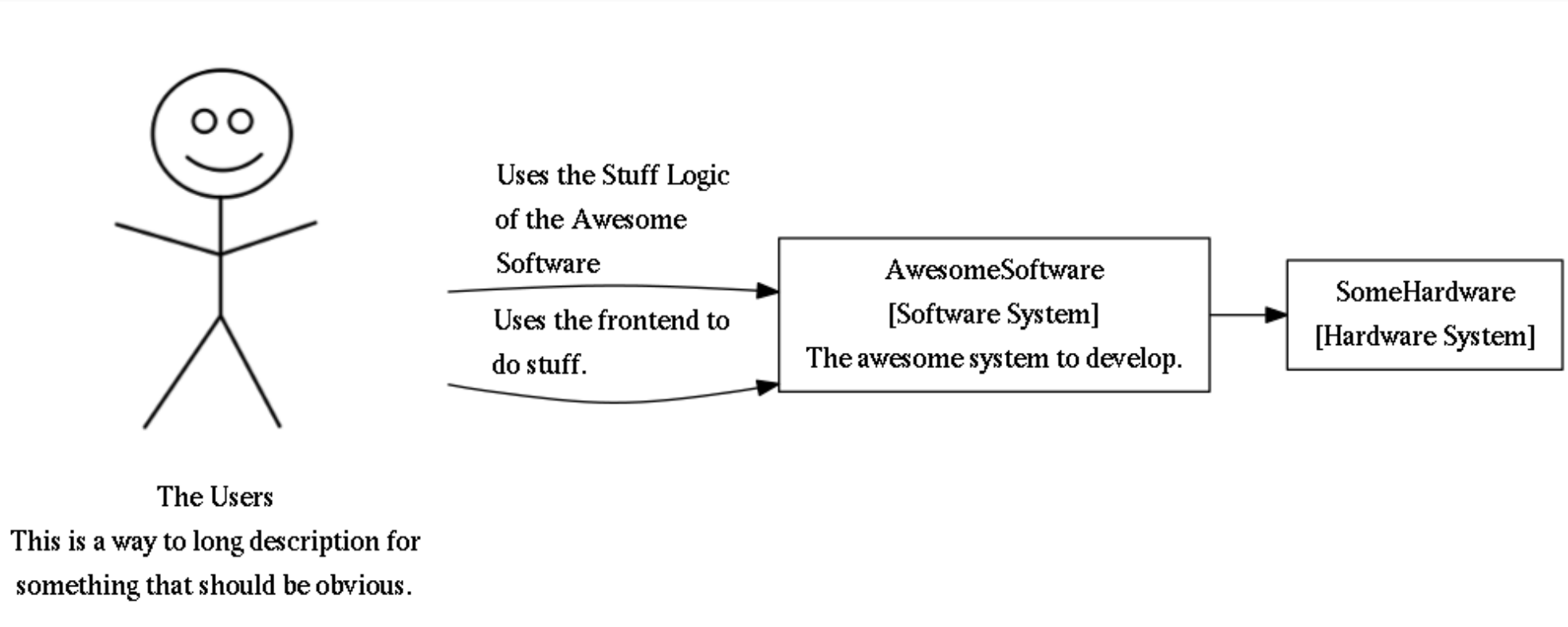
- Types e.g. strings are not always strings
  - D string
  - MySQL text
  - C++ std::string

```
struct Type {
    string name;
    string[string] typeMapping;
}

auto pwdHash = Type("PasswordString");
pwdHash.typeMappings["D"] = "string";
pwdHash.typeMappings["MySQL"] = "VARCHAR(128)";
```

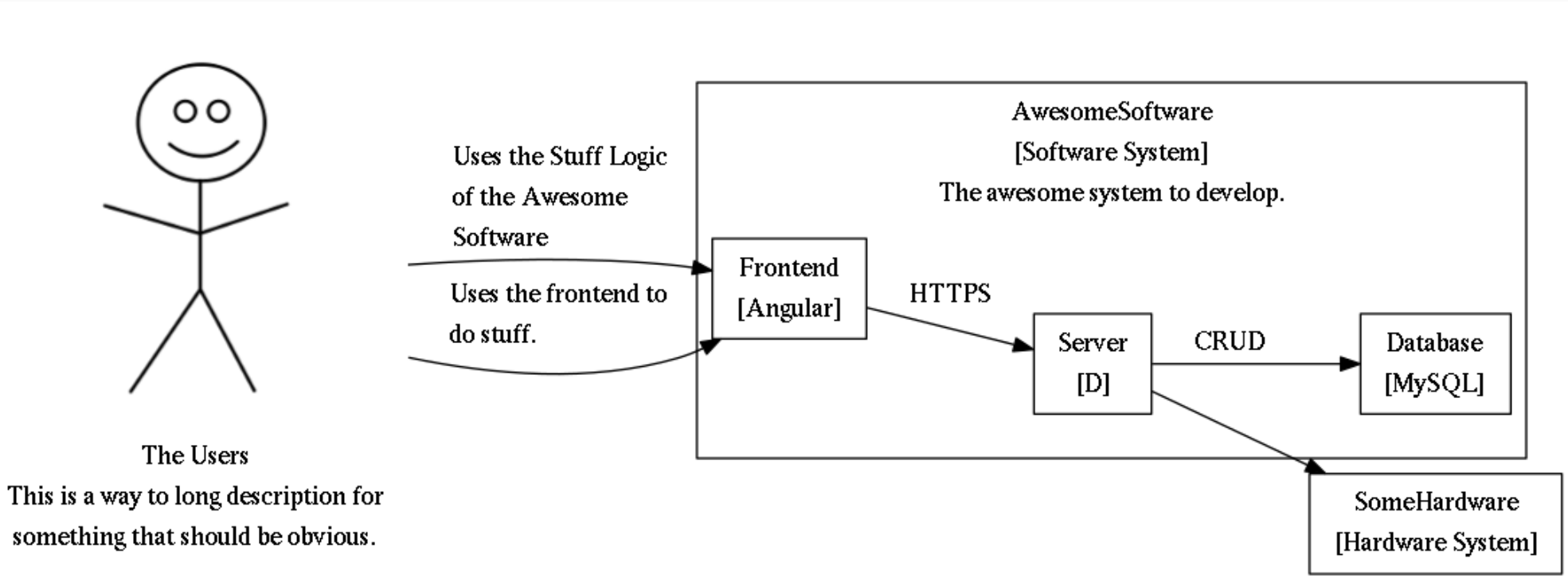
Generating

```
Graphvic gv = new Graphvic(world, "GraphvizOutput");
gv.generate();

MySQL mysql = new MySQL(world, "MySQL");
mysql.generate(database);
```

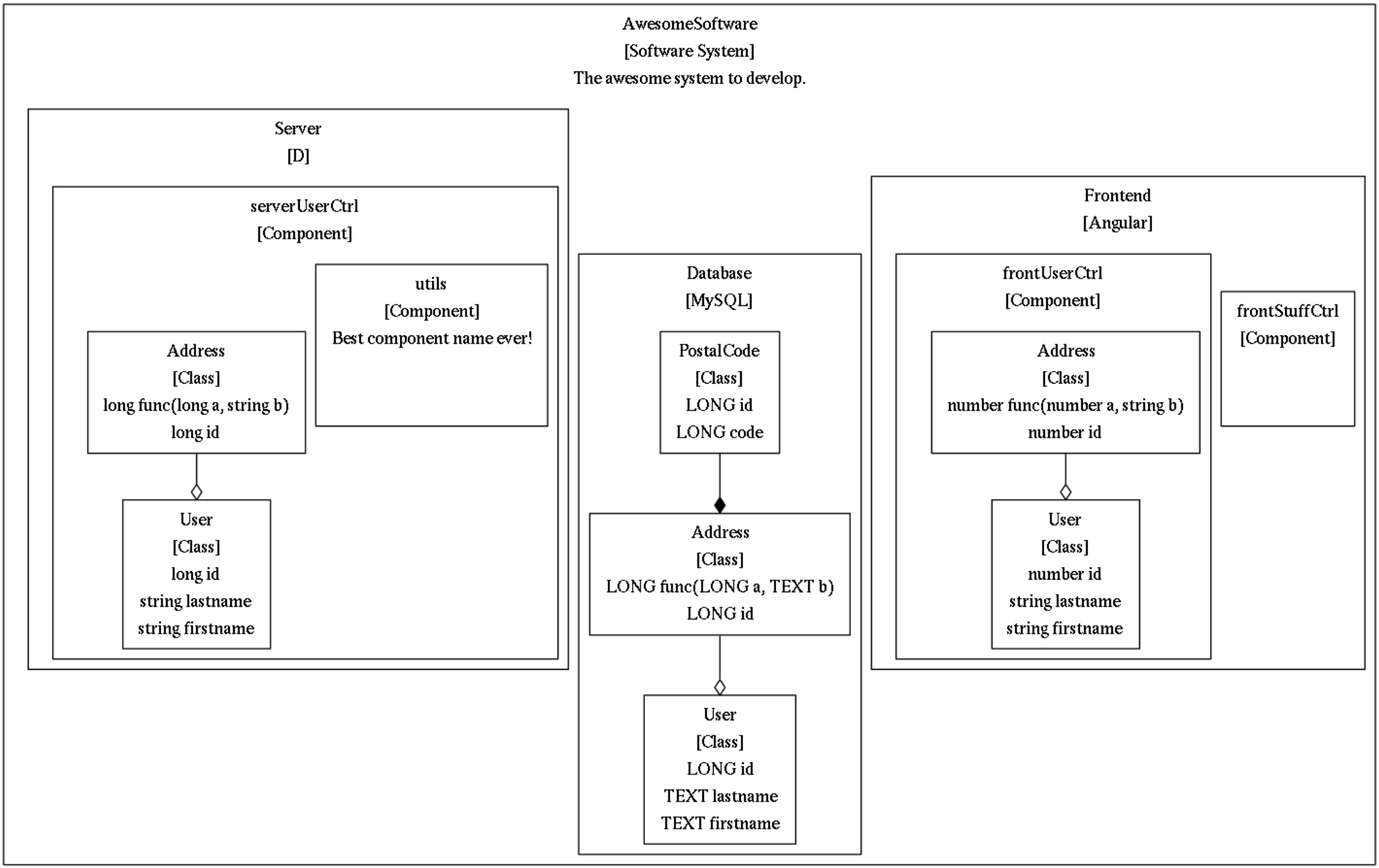
The World


Uses the Stuff Logic
of the Awesome
Software
Uses the frontend to
do stuff.
AwesomeSoftware
[Software System]
The awesome system to develop.
SomeHardware
[Hardware System]
The Users
This is a way to long description for
something that should be obvious.

The World and Containers



Awesome Software System



Generating the Database CREATE TABLE Statements

```
CREATE TABLE Address {
    id LONG PRIMARY KEY
};
```

```
CREATE TABLE User {
    id LONG PRIMARY KEY AUTO
        ↪ INCREMENT,
    lastname TEXT,
    firstname TEXT
};
```

```
CREATE TABLE Address_User {
    User_id LONG
    FOREIGN KEY(User_id)
        ↪ REFERENCES User(id) ON
        ↪ UPDATE CASCADE ON
        ↪ DELETE CASCADE,
    Address_id LONG
    FOREIGN KEY(Address_id)
        ↪ REFERENCES Address(id)
        ↪ ON UPDATE CASCADE ON
        ↪ DELETE CASCADE
}
```

```
CREATE TABLE PostelCode {
    id LONG PRIMARY KEY AUTO
        ↪ INCREMENT,
    code LONG,
    Address_id LONG
    FOREIGN KEY(Address_id)
        ↪ REFERENCES Address(id)
        ↪  ON UPDATE CASCADE ON
        ↪ DELETE CASCADE
};
```

What can we generate

- Diagrams describing the project at different levels of detail
- Database schema
- phpmyadmin clones
- Database access code
- Data objects (D struct/class, Typescript interface/class, ...
- Server skeletons
- Frontend skeletons

What can we generate

- Diagrams describing the project at different levels of detail
- Database schema
- phpmyadmin clones
- Database access code
- Data objects (D struct/class, Typescript interface/class, ...
- Server skeletons
- Frontend skeletons
- Graphviz mostly done, MySQL is getting there, Vibe.d and Angular2 next

The End

- vibe.d `https://vibed.org`
- typescript `https://www.typescriptlang.org/`
- dstructtotypescript
  `https://github.com/burner/dstructtotypescript`
- C4 Architecture (Simon Brown)
  `http://www.codingthearchitecture.com`
- Structurizr `https://structurizr.com/`
- Degenerator `https://github.com/burner/Degenerator`