

# Microservices + Authentication

SWE 432, Fall 2016

Design and Implementation of Software for the Web

# Today

- How do we split up the different modules of our apps?
  - Services vs Modules
  - Practical: Authentication services

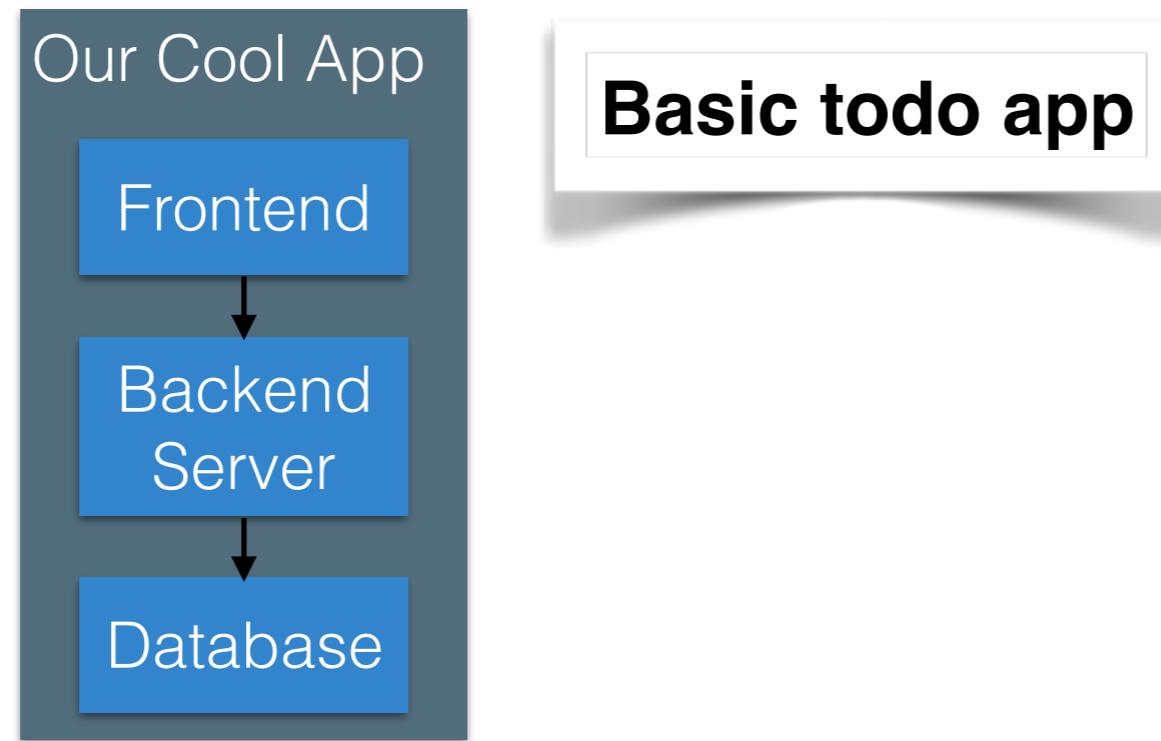
For further reading:

<http://www.martinfowler.com/articles/microservices.html>

<https://blog.karmawifi.com/how-we-build-microservices-at-karma-71497a89bfb4#.54o4pr8nd>

<http://techblog.netflix.com/search?q=microservices>

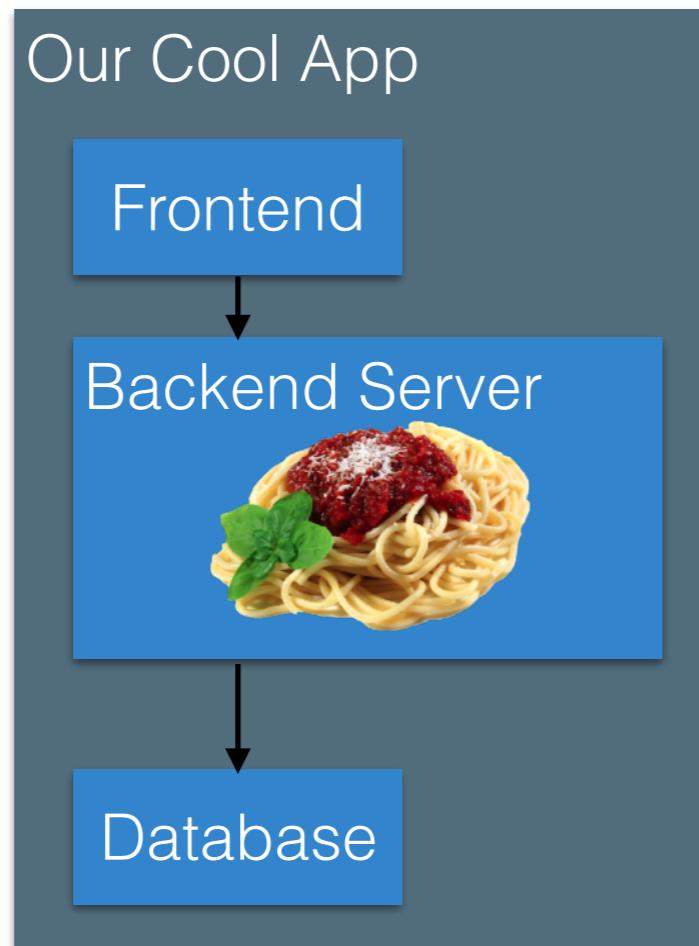
# How do we build big apps?



**Basic todo app**

What happens when we want to add more functionality to our backend?

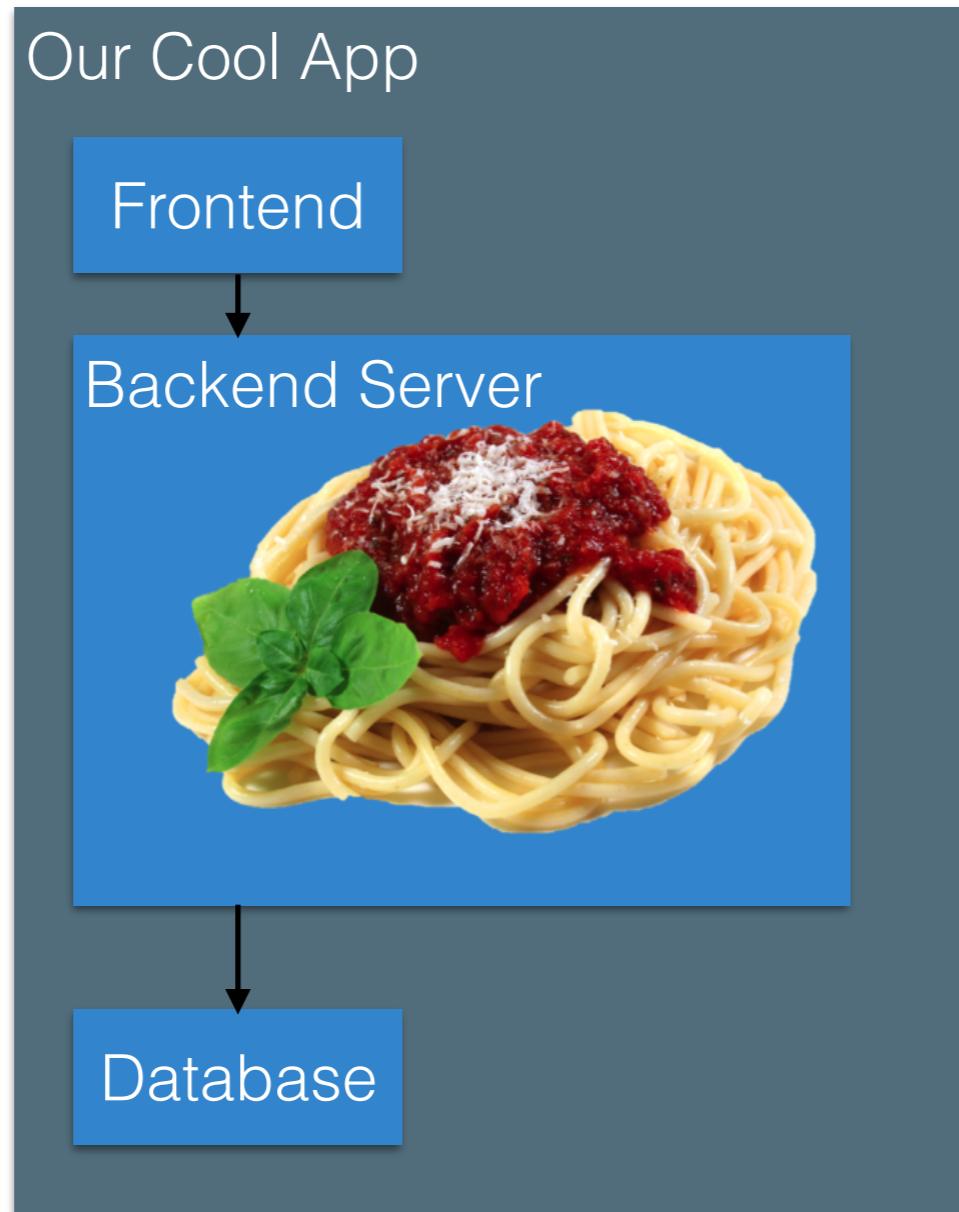
# How do we build big apps?



**Basic todo app with new feature to email todo reminders**

What happens when we add more functionality?

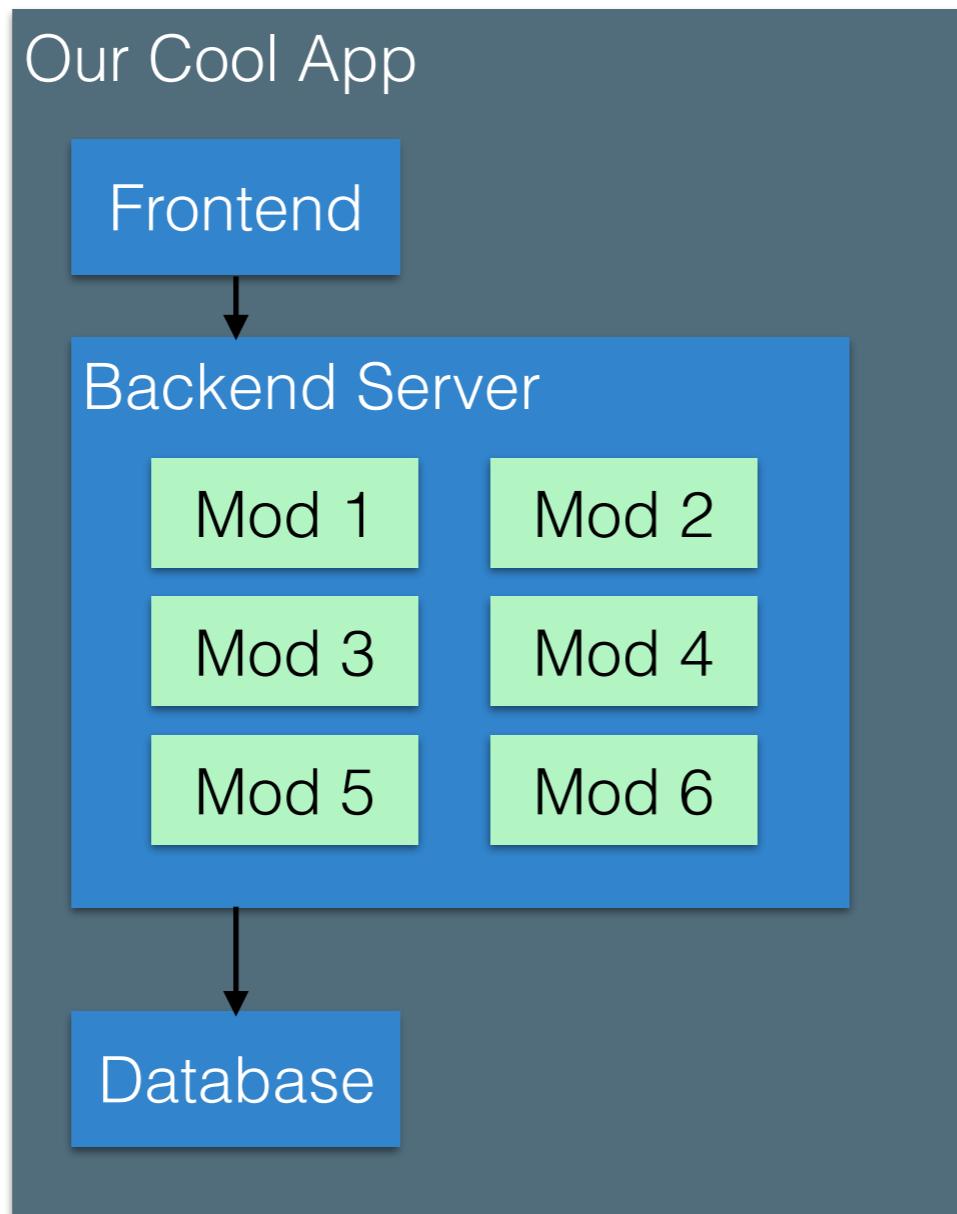
# How do we build big apps?



**Basic todo app with new feature to email todo reminders PLUS something to find events on Facebook and create Todos for them**

But we learned about modules, so our backend isn't total spaghetti but rather...

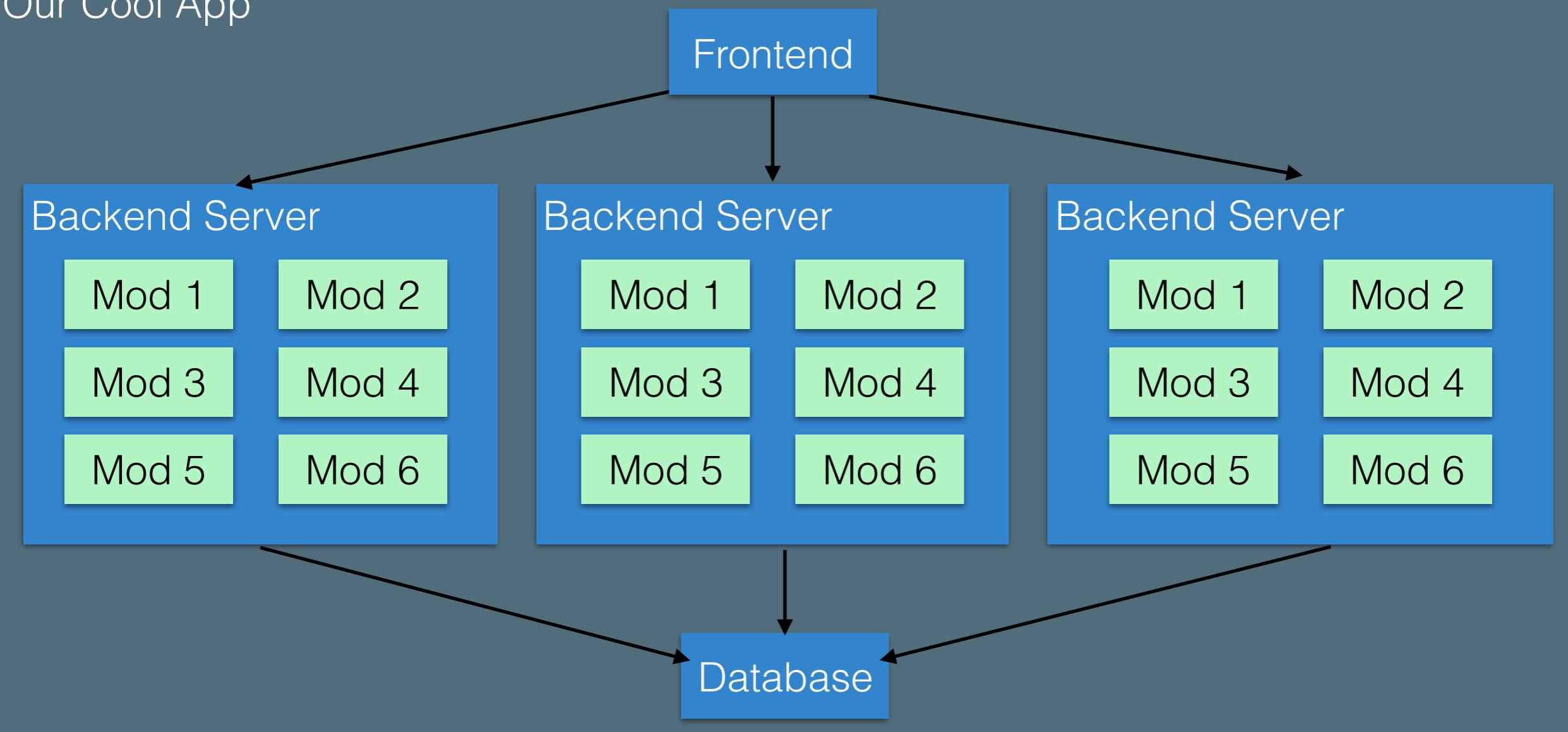
# How do we build big apps?



Our backend is not an unorganized mess, but instead organized into modules. Now how do we scale it? Run multiple backends?

# Now how do we scale it?

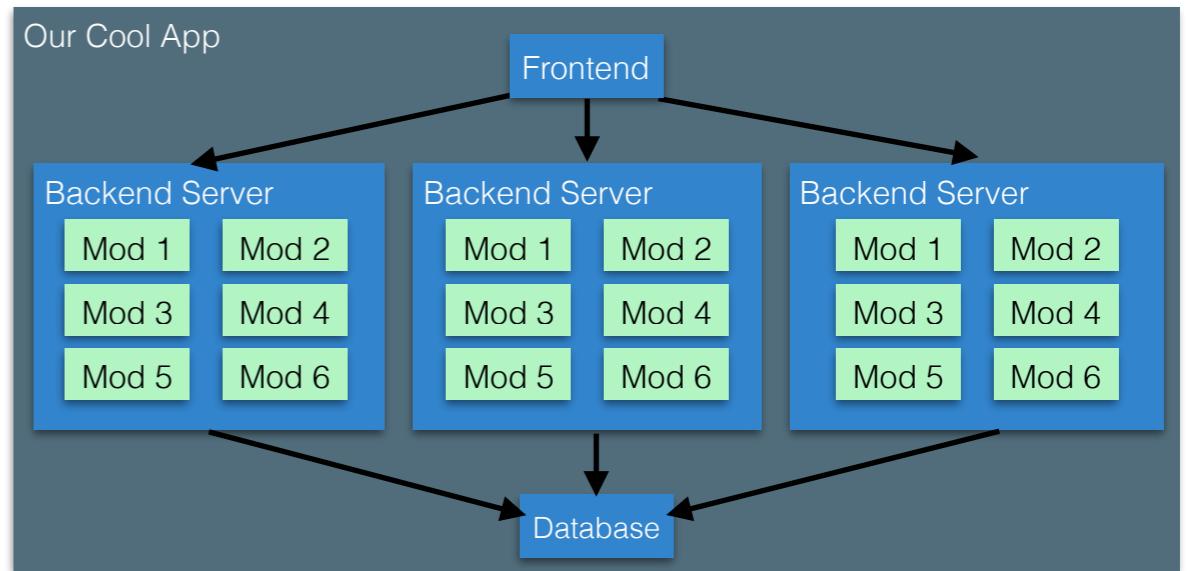
Our Cool App



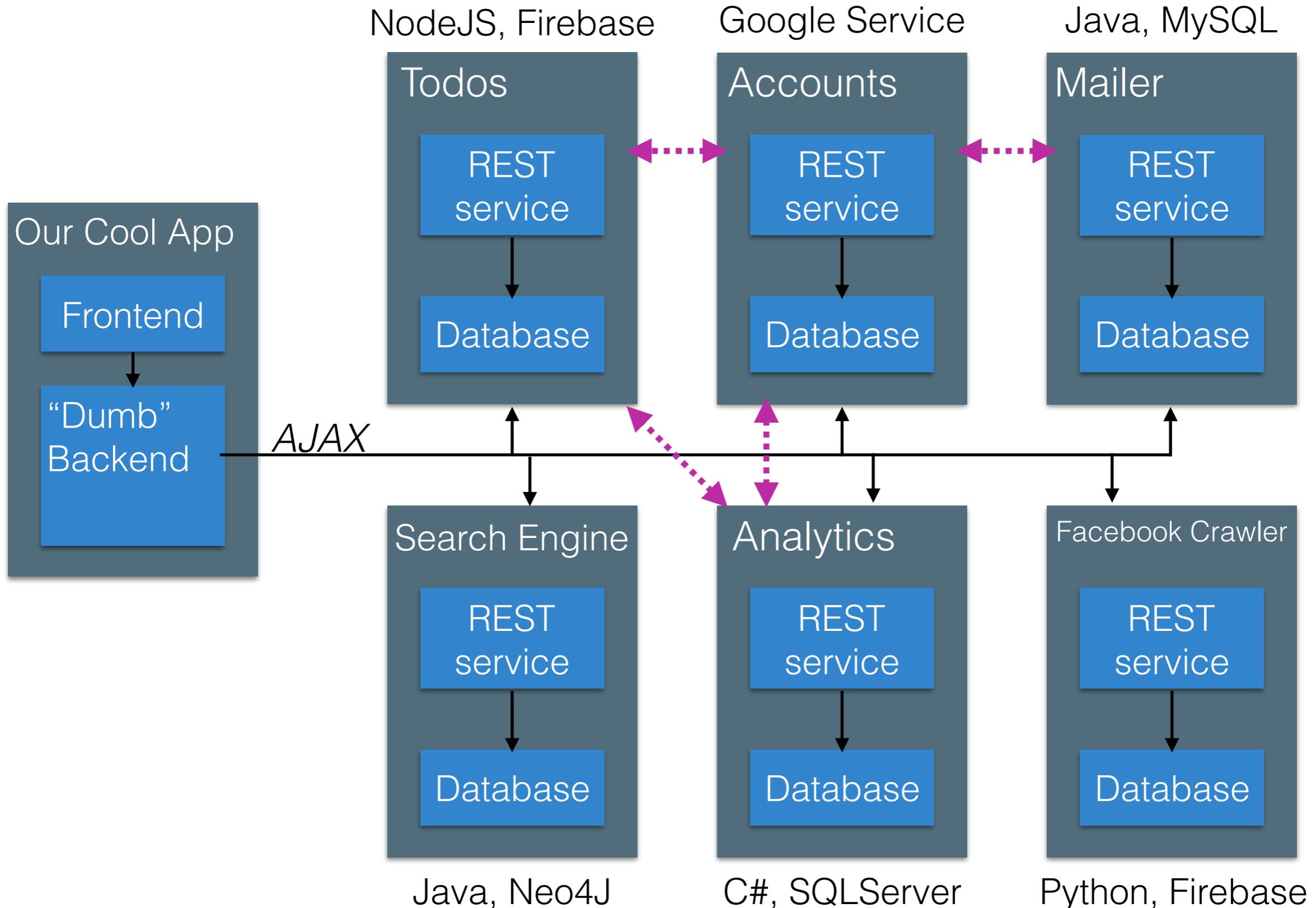
We run multiple copies of the backend, each with each of the modules

# What's wrong with this picture?

- This is called the “monolithic” app
- If we need 100 servers...
- Each server will have to run EACH module
- What if we need more of some modules than others?
- How do we update individual modules?
- Do all modules need to use the same DB and language, runtime etc?

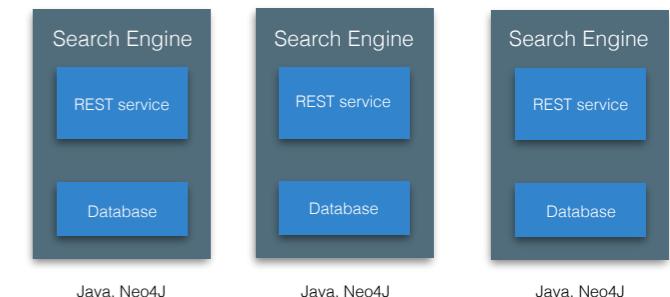
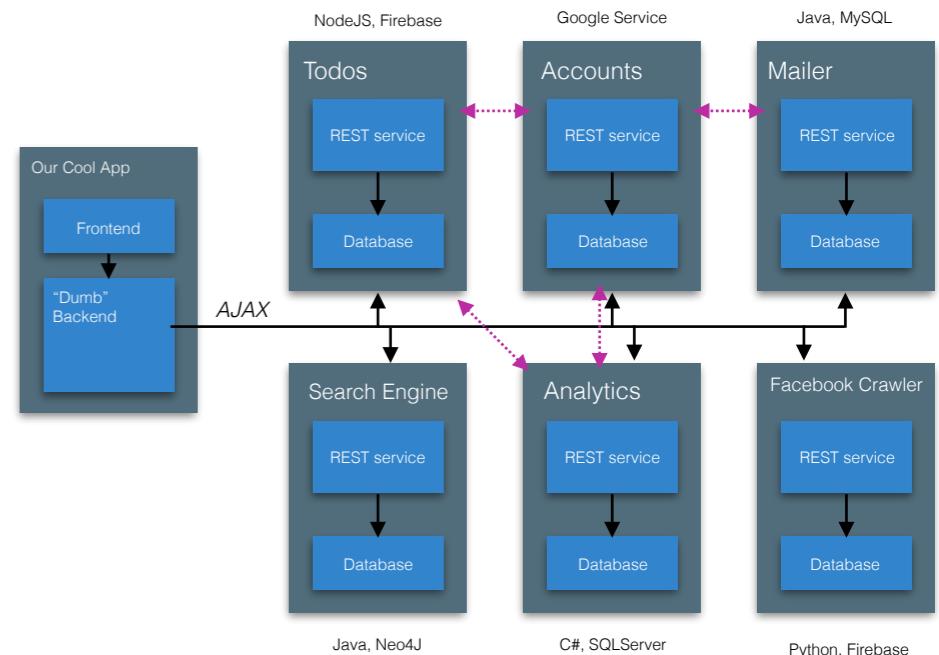


# Microservices



# What's good about this picture?

- Spaghetti is contained
- Components can be developed totally independently
  - Different languages, runtimes, OS, hardware, DB
- Components can be replaced easily
  - Could even change technology entirely (or use legacy service)
- Can scale individual components at different rates
  - Components may require different levels of resources



# Requirements for successful microservices

- 1 component = 1 service
- 1 business use case = 1 component
- Smart endpoints, dumb pipes
- Decentralized governance
- Decentralized data management
- Infrastructure automation
- Design for failure
- Evolutionary design

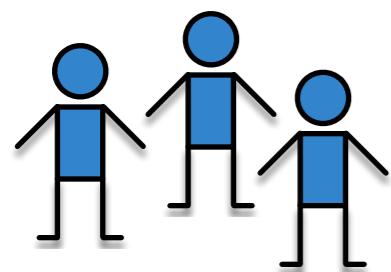
# How big is a component?

- Metaphor: Building a stereo system
- Components are independently replaceable
- Components are independently updatable
- This means that they can be also independently developed, tested, etc
- Components can be built as:
  - Library (e.g. module)
  - Service (e.g. web service)

# Components as Libraries or Services?

- Microservices says 1 service per component
- This means that we can:
  - Develop them independently
  - Upgrade them independently
  - Have ZERO coupling between components, aside from their shared interface

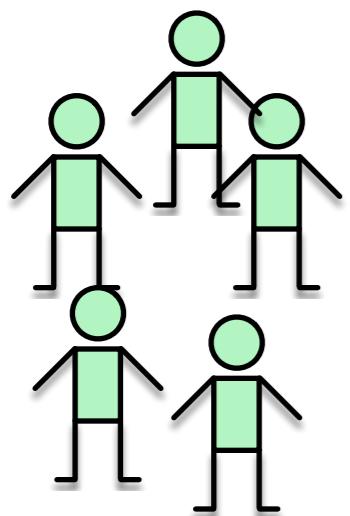
# Organization around business capabilities



Frontend

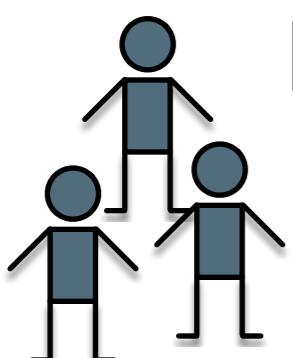
Orders, shipping, catalog

**Classic teams:**  
**1 team per “tier”**



Backend

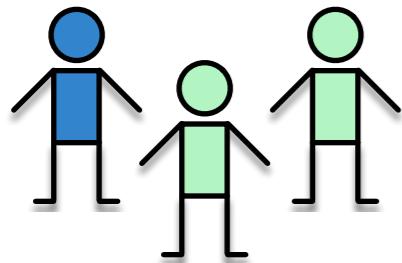
Orders, shipping, catalog



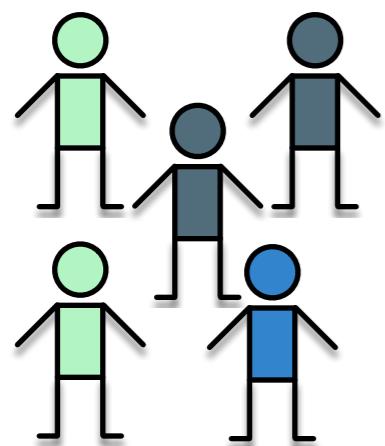
Database

Orders, shipping, catalog

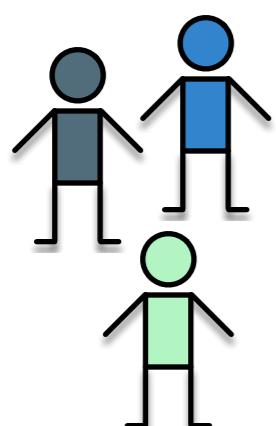
# Organization around business capabilities



Orders



Shipping



Catalog

## Example: Amazon

Teams can focus on one business task  
And be responsible directly to users

**“Full Stack”**

**“2 pizza teams”**

# Decentralized Data Management

- Decentralizes implementation decisions
- Services exchange data ONLY through their exposed APIs - NO shared databases
- Allows each service to manage data in the way that makes the most sense for that service

# Infrastructure Automation (Teaser for next week)

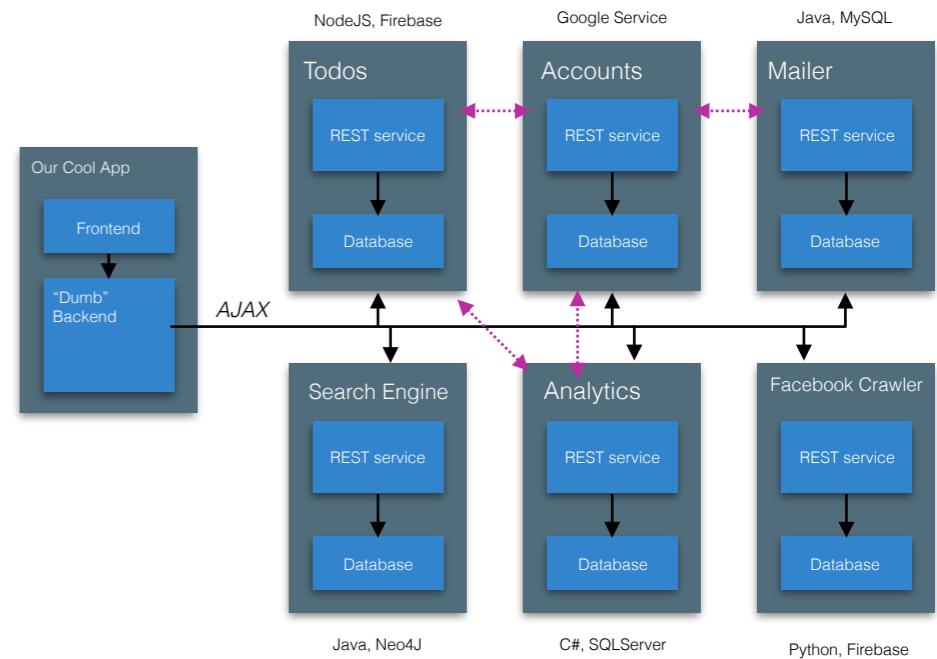
- To effectively use microservices, you'll probably be:
  - Developing services independently
  - Deploying services independently (and often)
- MUST HAVE:
  - Rapid provisioning of servers so you can take advantage of scaling
  - Monitoring to see when things aren't talking nicely
  - Rapid deployment of new/updated services
  - Strong culture of integration between those doing the monitoring and those doing the development ("devops")
- Key to successfully using microservices is automating all of this
- We'll talk about this stuff more next lecture

# Design for Failure

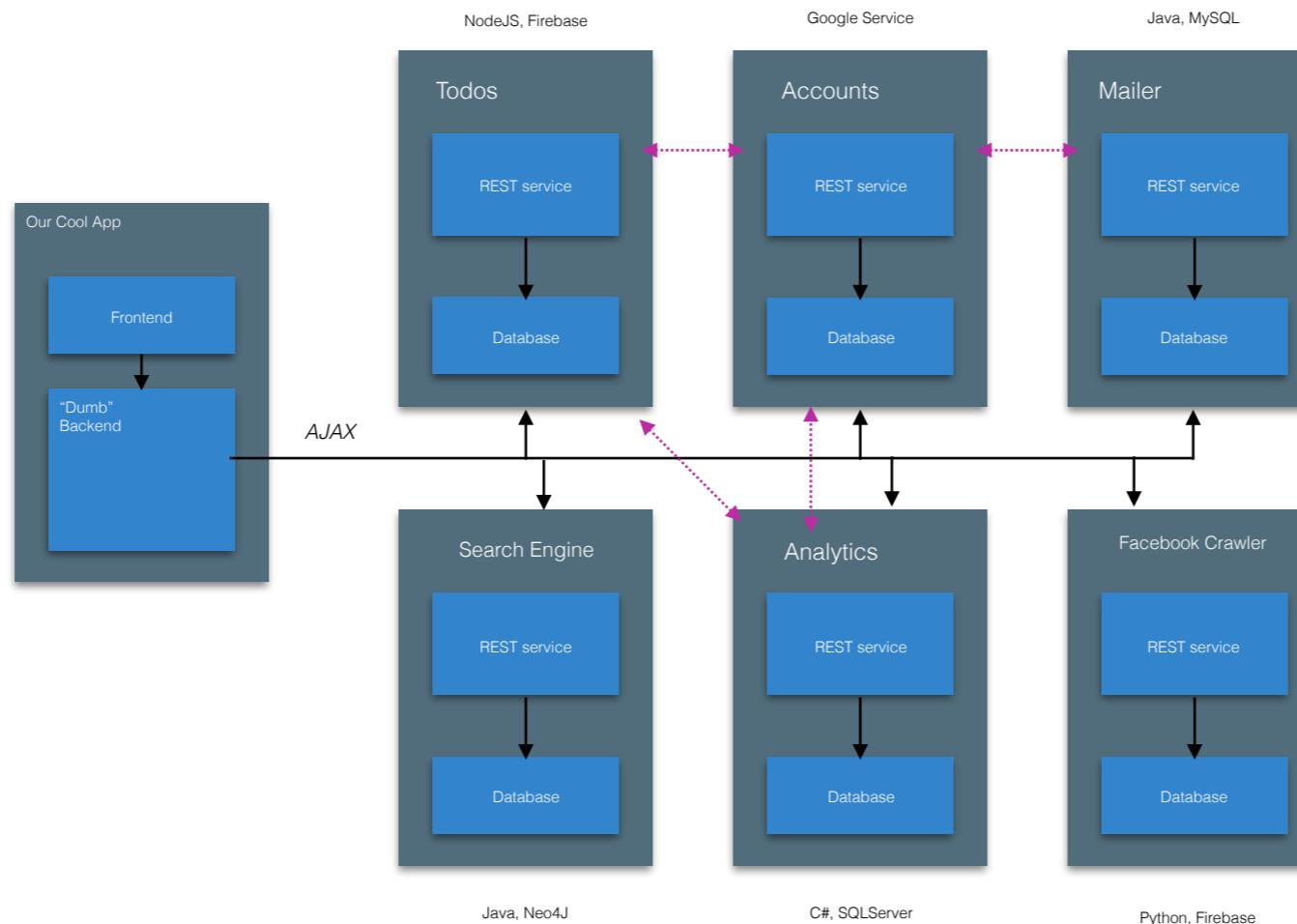
- Our system picture has gotten a lot more complicated
- Lots more moving pieces that might **fail**:
  - Services might have bugs
  - Services might be slow to respond
  - Entire servers might go down
    - If I have 60,000 HD's, 3 fail a day
- Key: design every service assuming that at some point, everything it depends on might disappear - must fail “gracefully”
- Netflix simulates this constantly with “ChaosMonkey”

# Maintaining Consistency

- One of our rules was “no shared database”
- But surely some data will be shared
- Updates are sent via AJAX...
- No guarantee that those updates occur immediately
- Instead, guarantee that they occur ***eventually***
- Can force some ordering, but that’s expensive



# Maintaining Consistency



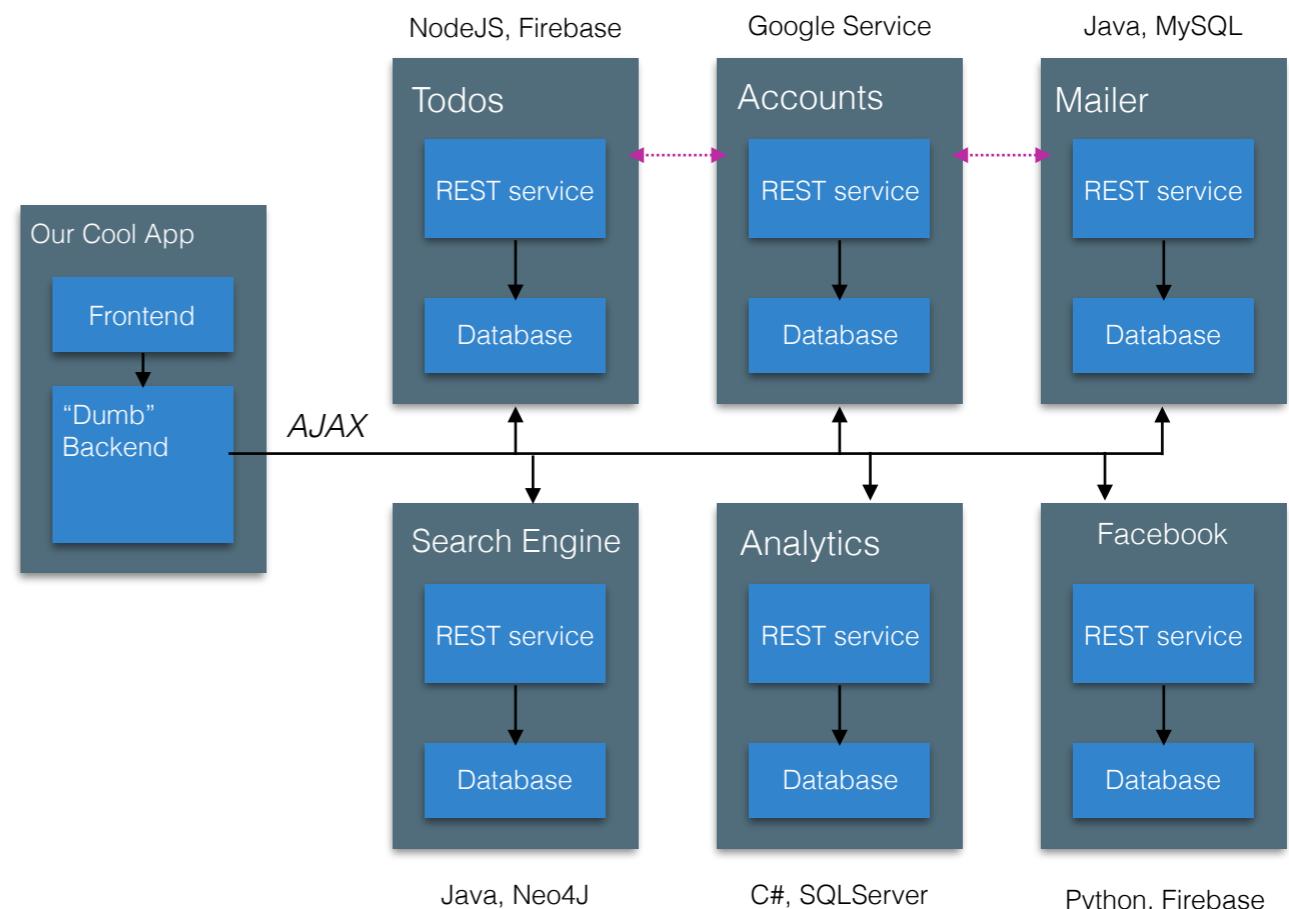
- Core problem: different services may respond to requests at **different times**.
  - What if a request results in change to resource in one service, but other service has not yet processed corresponding request?
  - May end up with different states in different resources.
  - Logic needs to be written to correctly handle such situations.

# When to use?

- Monolith:
  - Simplicity
    - Microservices require distributed computing, a lot of async business...
  - Consistency
  - Easy to refactor what's in 1 module vs another - better early on
- Microservice:
  - Partial deployment
    - Netflix benefited big from this - hourly/daily updates to components
  - Availability - even if one service goes down, the rest stays up
  - Modularity is enforced
  - Easy to use multiple platforms
- Often times, might have something of a hybrid

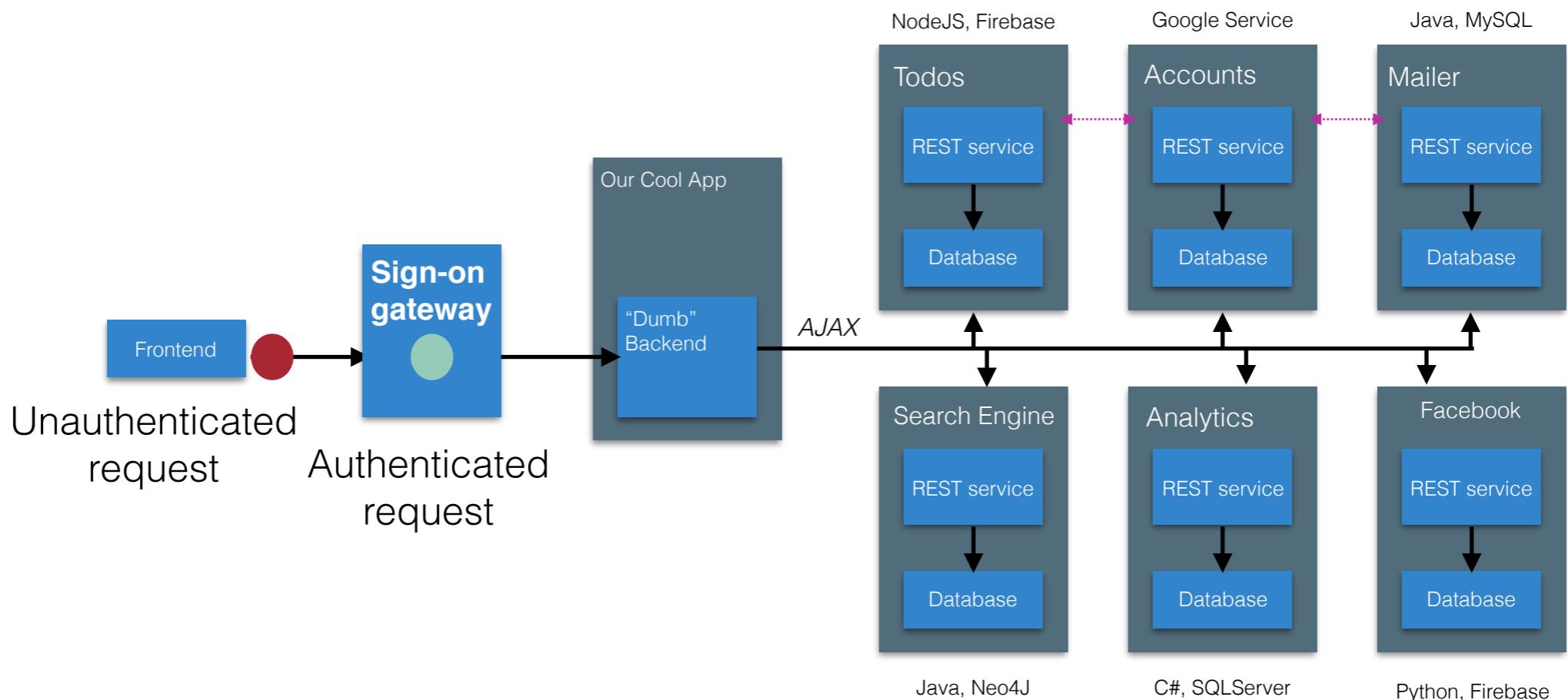
# Microservices & Authentication

- If using microservices, how do we decide who is logged in?
- Typical solution: Sign-on gateway



# Microservices & Authentication

- If using microservices, how do we decide who is logged in?
- Typical solution: Sign-on gateway

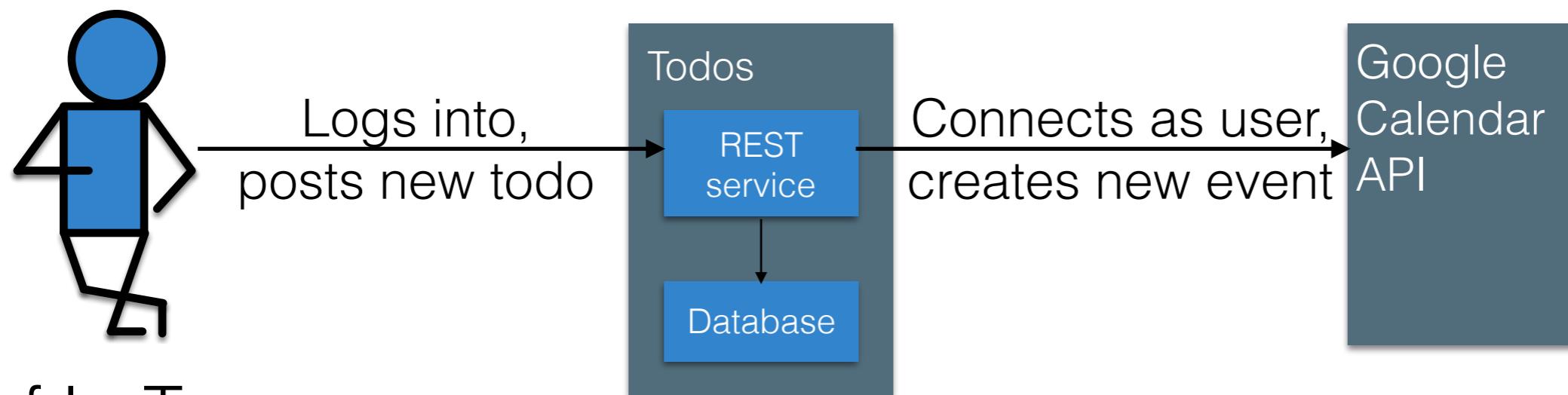


# Authentication & Authorization

- Putting this sign on gateway will ensure that people are signed in
- But how do we ensure that someone is **authorized** to view some given data or make some request?
- Role of individual services to check back (either with authorization service, or some other service)

# Bigger picture - authentication with multiple service providers

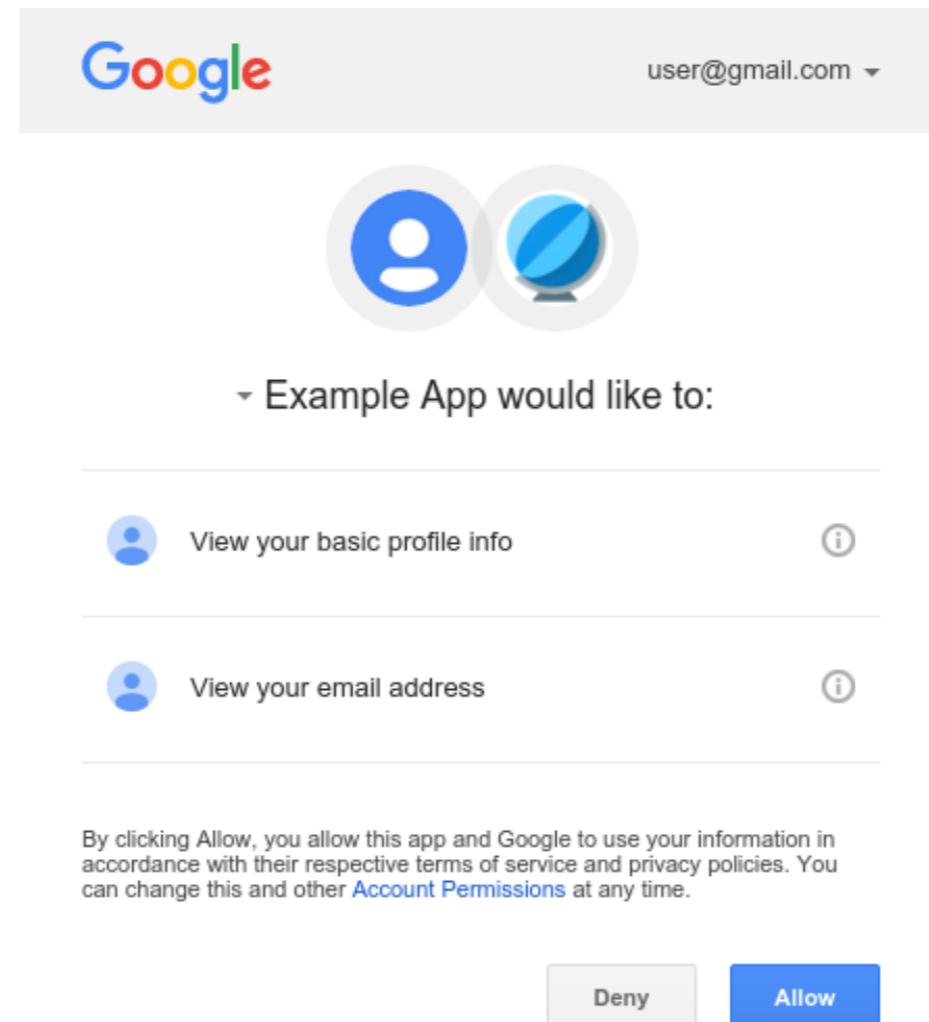
- Let's consider updating our Todos app so that it can automatically put calendar events on my Google Calendar



Prof LaToza

How does Todos tell Google that it's posting something for Prof LaToza?  
Should Prof LaToza tell the Todos app his Google password?

# We've got something for that...

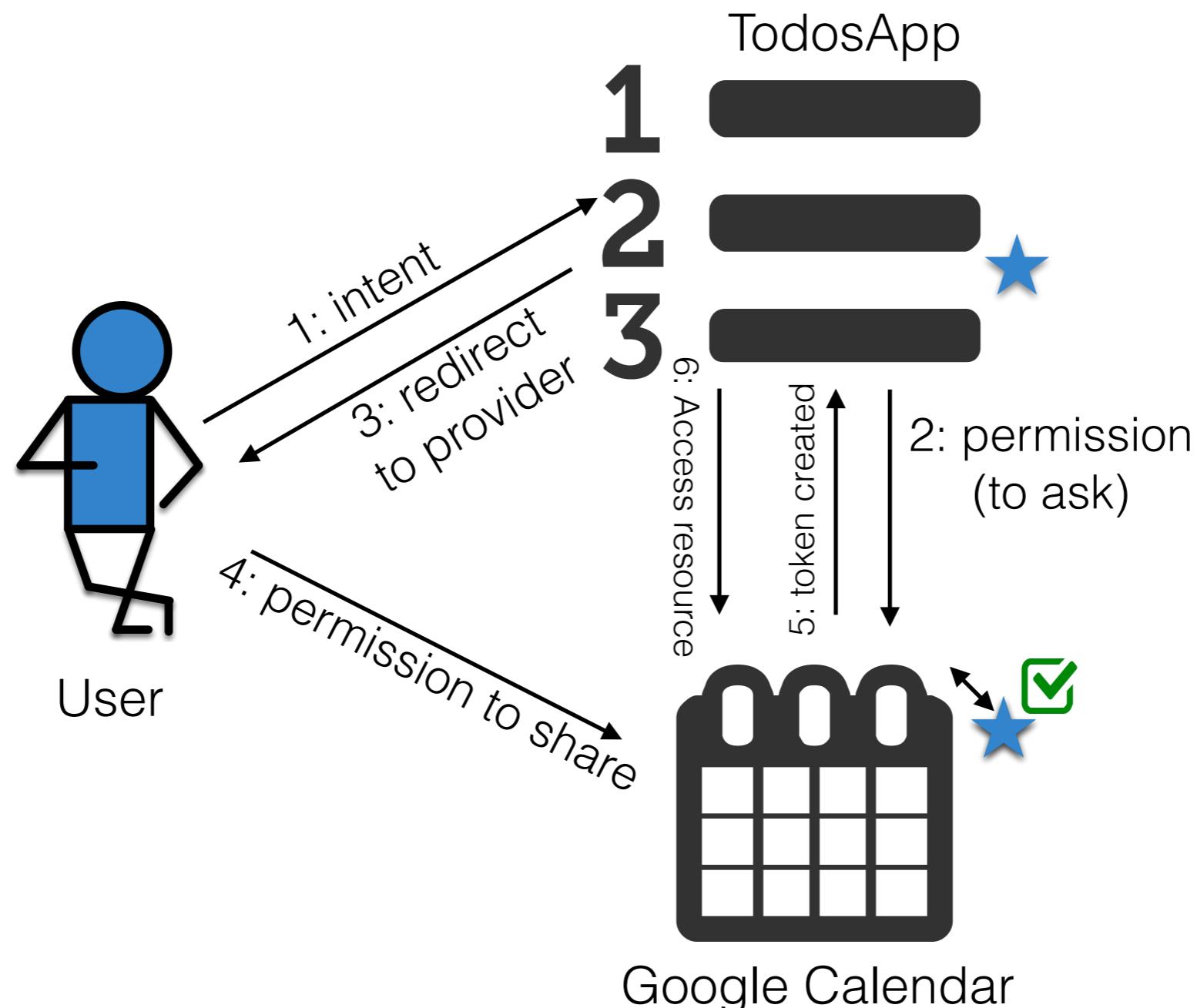


# OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” **without** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof LaToza,” “Todos App,” “Google Calendar”
- Service provider issues a **token** on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user
- Protocol could be considered a conversation...

# An OAuth Conversation

Goal: TodosApp can post events to User's calendar.  
TodosApp never finds out User's email or password



# Tokens?

A token is a **secret value**. Holding it gives us access to some privileged data. The token identifies our users and app.

Example token:

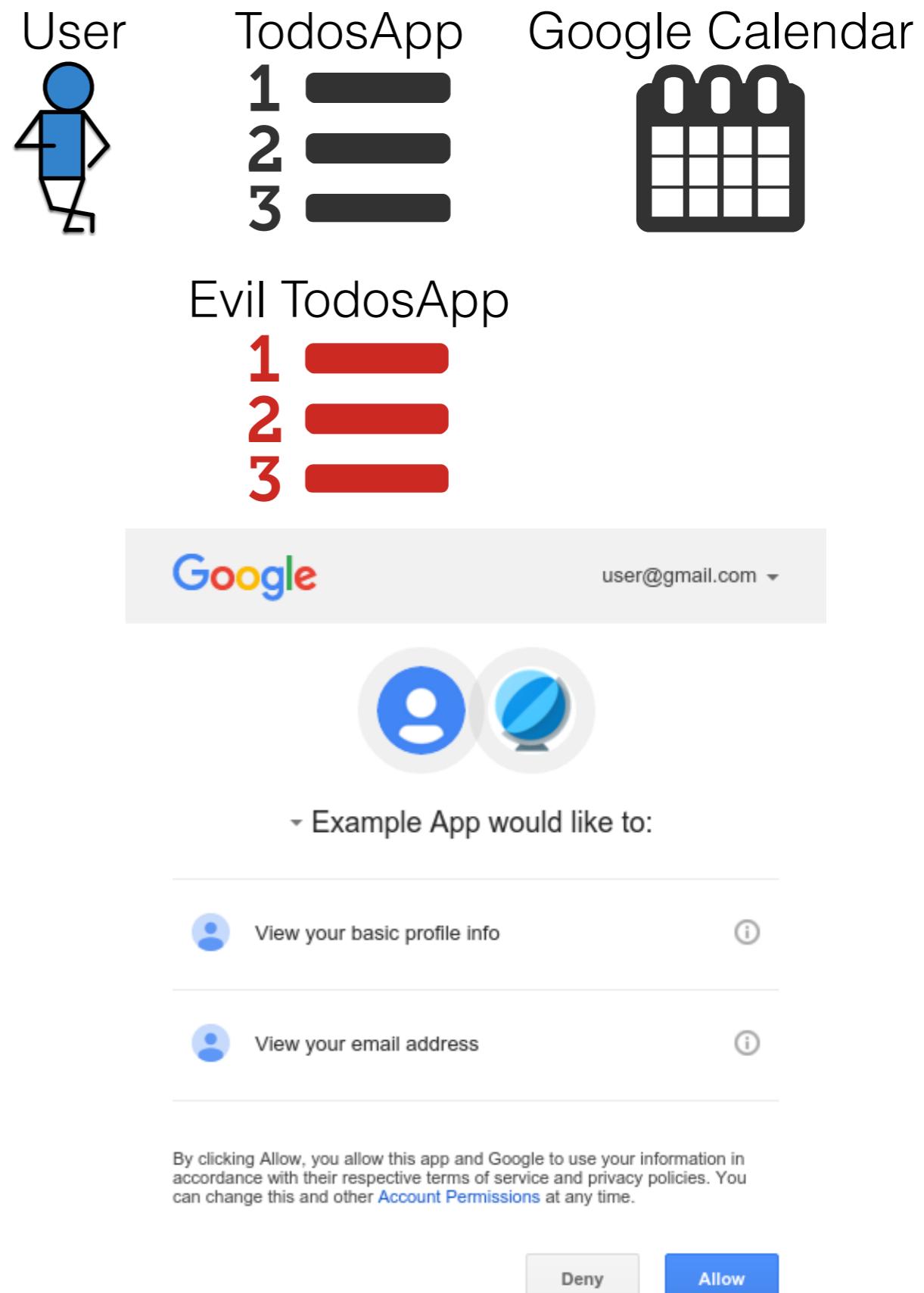
```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImU3Yjg2NjFjMGUwM2Y3ZTk3NjQyNGUxZWFiMzI50WIxNzRhNGVlNWUiifQ.eyJpc3Mi0iJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vYXV0aGRlbW8tNzJhNDIiLCJuYW1lIjoiSm9uYXRoYW4gQmVsbCIsInBpY3R1cmUi0iJodHRwczovL2xoNS5nb29nbGV1c2VyY29udGVudC5jb20vLW0tT29jRlU1R0x3L0FBQUFBQUFBQUFBQUFBQUgwL0JVV2t0NkRtTVJrL3Bob3RvLmpwZyIsImF1ZCI6ImF1dGhkZW1vLTcyYTQyIiwiYXV0aF90aW1lIjoxNDc3NTI5MzcxLCJ1c2VyX2lkIjoiSk1RclFpdTlTUlRkeDY0YlR5Z0EzeHhEY3VIMiIsInN1YiI6IkpNUXJRaXU5U1JUZHg2NGJUeWdBm3h4RGN1SDIiLCJpYXQiOjE0Nzc1MzA40DUsImV4cCI6MTQ3NzUzNDQ4NSwiZW1haWwi0iJqb25iZWxsd2l0aG5vaEBnbWFpbC5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmlyZWJhc2Ui0nsiaWRlbnRpdGllcyI6eyJnb29nbGUuY29tIjpbIjEw0TA0MDM1MjU3NDMxMjE1NDIxNiJdLCJlbWFpbCI6WyJqb25iZWxsd2l0aG5vaEBnbWFpbC5jb20iXX0sInNpZ25faW5fcHJvdmlkZXIi0iJnb29nbGUuY29tIn19.rw1pPK377hDGmSaX31uKRphKt4i79aHjceepnA8A2MppBQnPJlCqmgSapxs-Pwmp-1Jk382VooRwc8TfL6E1UQUl65yi2aYYzSx3mWMTwtPTHTkMN4E-GNprp7hX-pqD3PncBh1bq1dThPNyjHLp3CUlPPO_QwaAeSuG5xALhzfYkvLSINTy4FguD9vLHydpVHWscBNCDHAC0qSeV5MzUs6ZYMnBIitFhbkak6z50ClvxGTGMhvI8m11hIHdWgNGnDQNNNoosiifzlwMqDHiF5t3K0L-mxtcNq33TvMAc43JElxnyB4g7qV2hJI0y4MLtLxphAfCeQZA3sxGf7vDXBQ
```

Decoded:

```
{  
  "iss": "https://securetoken.google.com/authdemo-72a42",  
  "name": "Thomas LaToza",  
  "picture": "https://lh5.googleusercontent.com/-m-0ocFU5GLw/AAAAAAAIAI/AAAAAAAHA0/BUWkN6DmMRk/photo.jpg",  
  "aud": "authdemo-72a42",  
  "auth_time": 1477529371,  
  "user_id": "JMQrQiu9SRTdx64bTygA3xxDcuH2",  
  "sub": "JMQrQiu9SRTdx64bTygA3xxDcuH2",  
  "iat": 1477530885,  
  "exp": 1477534485,  
  "email": "latoza@gmail.com",  
  "email_verified": true,  
  "firebase": {  
    "identities": {  
      "google.com": ["109040352574312154216"],  
      "email": ["latoza@gmail.com"]  
    },  
    "sign_in_provider": "google.com"  
  },  
  "uid": "JMQrQiu9SRTdx64bTygA3xxDcuH2"  
}
```

# Trust in OAuth

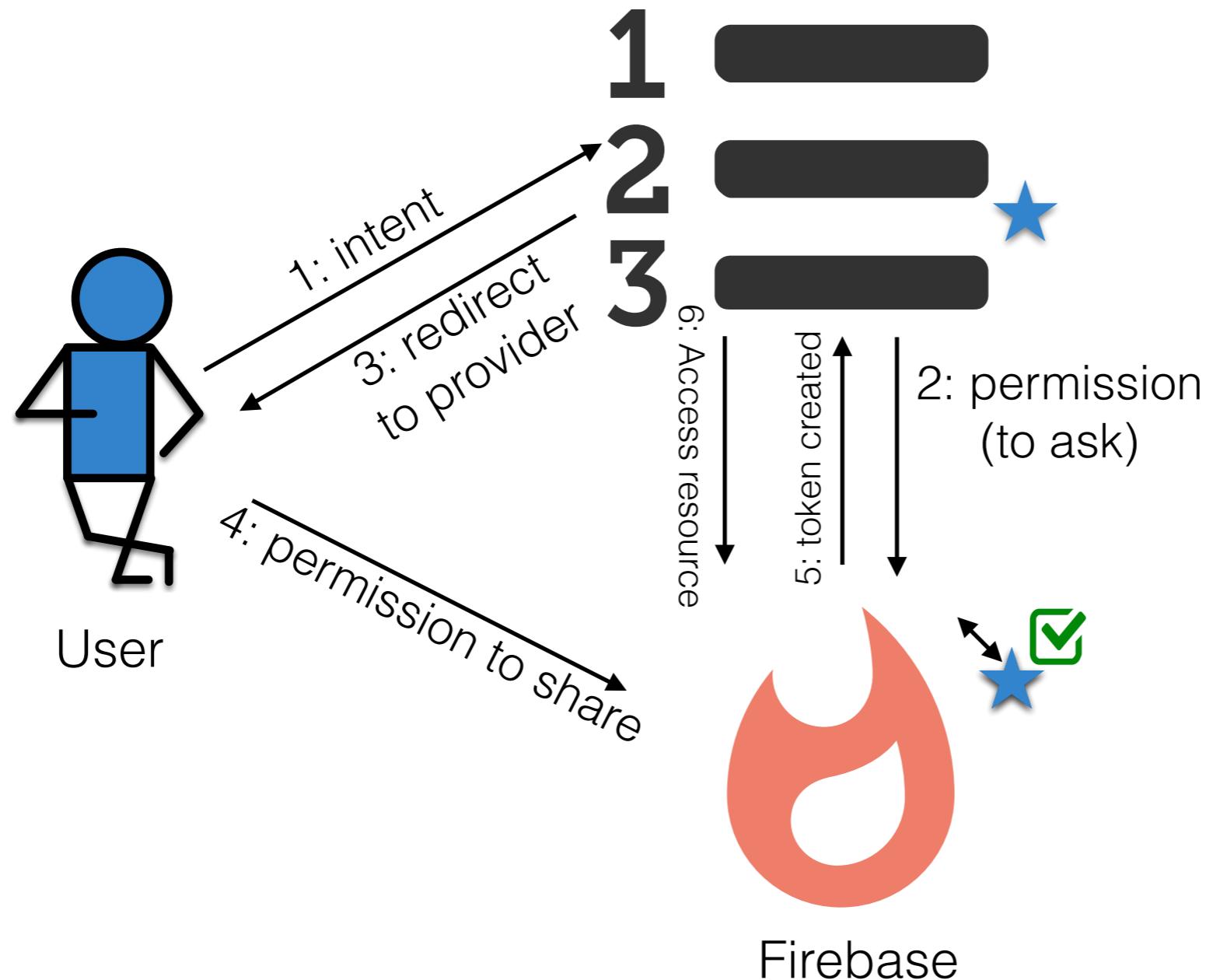
- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide
  - ... they were the one who clicked the link after all



# Authentication as a Service

- Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service
- Why?
  - Security
  - Reliability
  - Convenience
- We can use OAuth for this!
- We’re going to use Firebase’s authentication API in our homework this week

# Using an Authentication Service

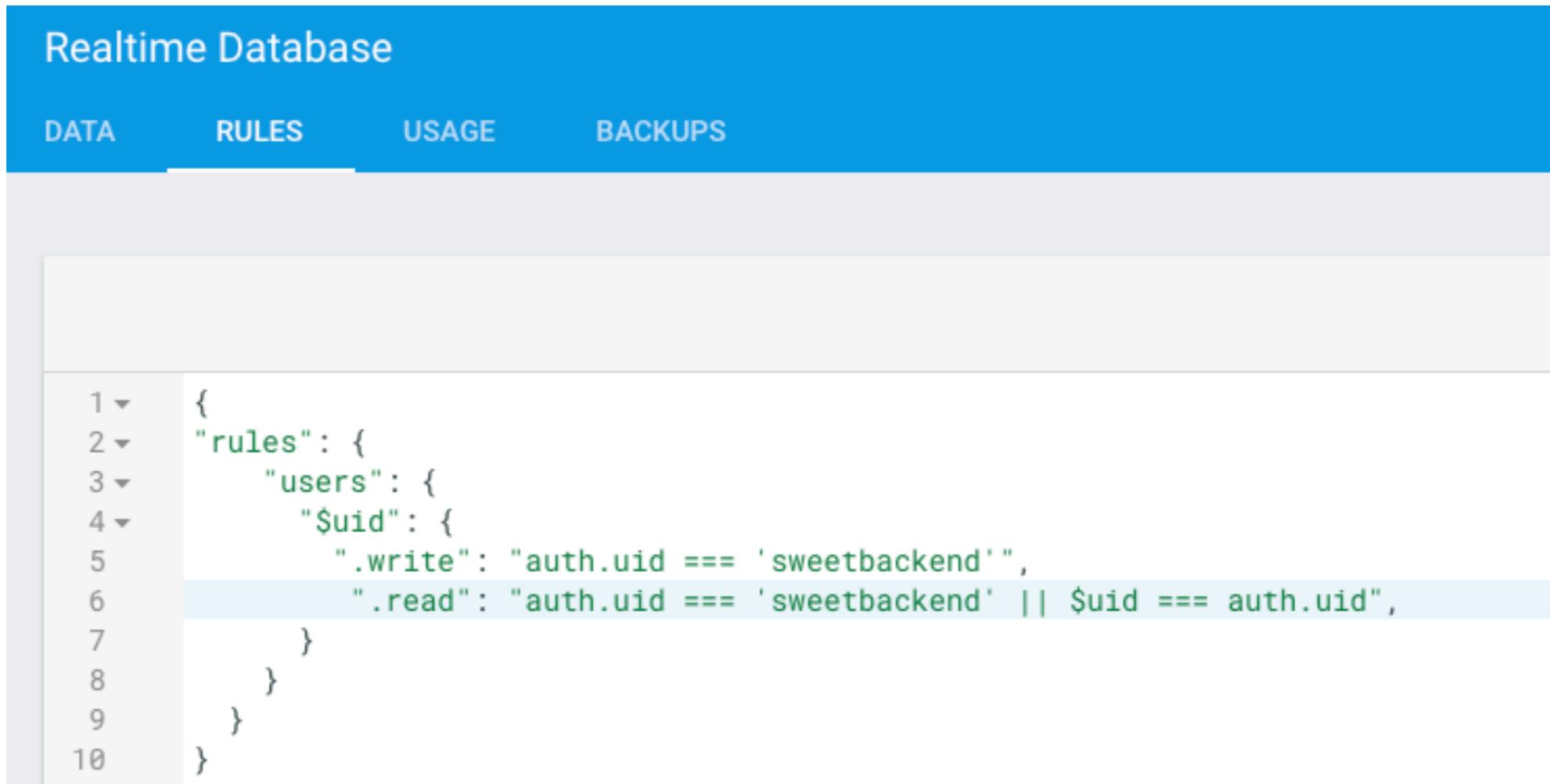


# Firebase Authentication

- Firebase provides an entire suite of authentication services you can use to build into your app
- Can either use “federated” logins (e.g. login with google, facebook, GitHub credentials) or simple email/password logins. Use whichever you want.
- Getting started guide: <https://github.com/firebase/FirebaseUI-Web>
- For backend: <https://firebase.google.com/docs/auth/server/verify-id-tokens>
- Firebase handles browser local storage to track that the user is logged in across pages (woo)

# Firebase Authentication

- We can update our firebase database rules so that:
  - Every user gets their own area
  - No user can read another user's area
  - Only our sweet backend server (with a private key called “sweetbackend” can write)



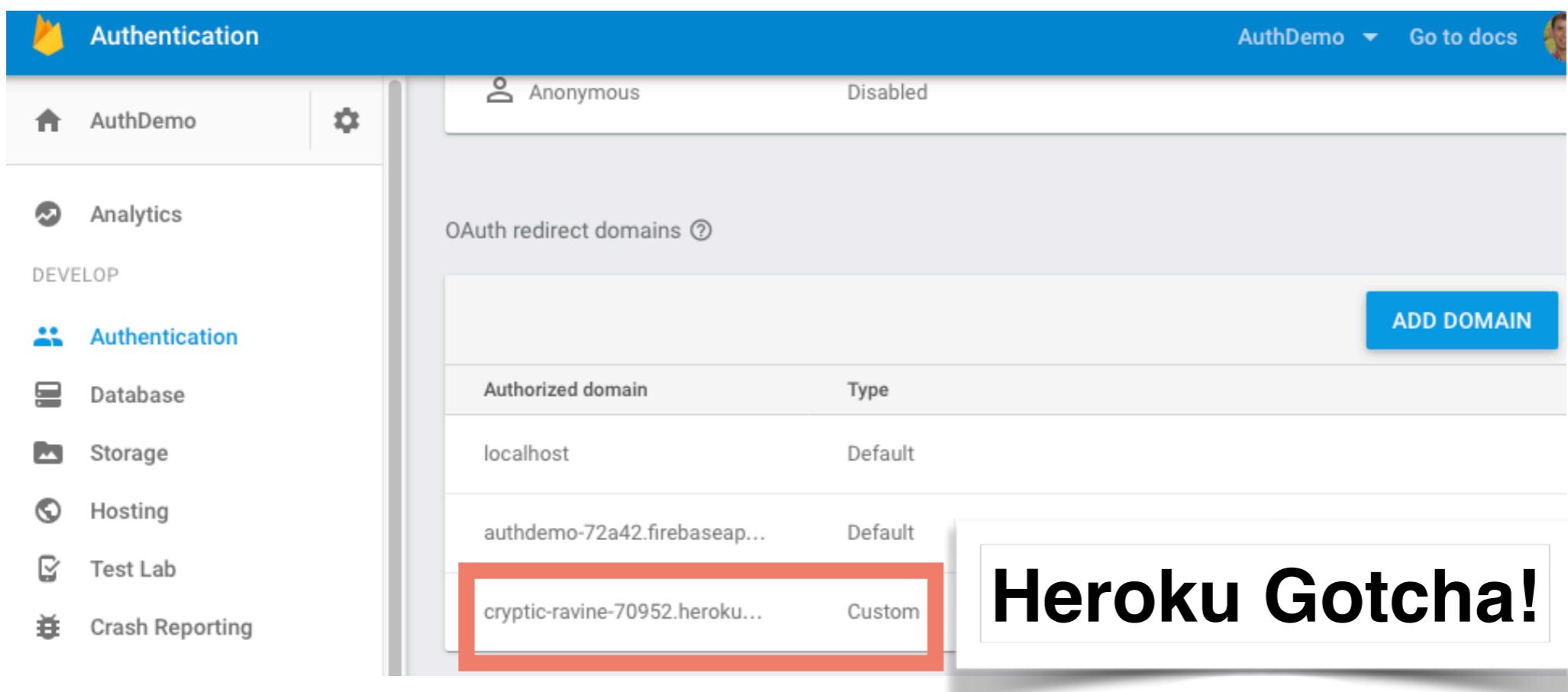
The screenshot shows the Firebase Realtime Database console. The top navigation bar is blue with the text "Realtime Database" and tabs for "DATA", "RULES", "USAGE", and "BACKUPS". The "RULES" tab is selected. Below the tabs is a large text area containing the database rules. The rules are defined as a JSON object with the following structure:

```
1  {
2    "rules": {
3      "users": {
4        "$uid": {
5          ".write": "auth.uid === 'sweetbackend'",
6          ".read": "auth.uid === 'sweetbackend' || $uid === auth.uid",
7        }
8      }
9    }
10 }
```

The ".read" rule at line 6 uses a logical OR operator (||) to allow both the backend server ("sweetbackend") and the user's own UID to read data. The ".write" rule at line 5 restricts writes to only the backend server.

# OAuth + Heroku

- Normally we set up our OAuth server so that it will only send the result of a login back to our own server
- Could be a security vulnerability if someone else could get it



Authentication

AuthDemo Go to docs

Anonymous Disabled

OAuth redirect domains

ADD DOMAIN

Authorized domain	Type
localhost	Default
authdemo-72a42.firebaseio...	Default
cryptic-ravine-70952.herokuapp...	Custom

**Heroku Gotcha!**

# Demo

- In our demo, we'll add Google login to our todo app
- We'll let Firebase manage client side logins
- When the client talks to our backend, it will send its token

# Exit-Ticket Activity

Go to [socrative.com](https://socrative.com) and select “Student Login”

Class: SWE432001 (Prof LaToza) or SWE432002 (Prof Bell)

ID is your [@gmu.edu](mailto:@gmu.edu) email

1: How well did you understand today's material

2: What did you learn in today's class?

For question 3:

Do you think your app could benefit from microservices?

**You may not submit this activity if you are not present in lecture.**

**Doing so will be considered academic dishonesty.**