Modern Node.js API Development with LoopBack 4

Jake Brauchler & Craig Freeman
Kenzan

Agenda

- History
- What is it?
- Why should I use it?
- How do I use it?
  - Key Concepts
- Why Upgrade from 3.x to 4.x?
- What about GraphQL?
- Demo
- Going Further

In the beginning...

- Single-threaded
- Event-driven
- Fast
- `http` package

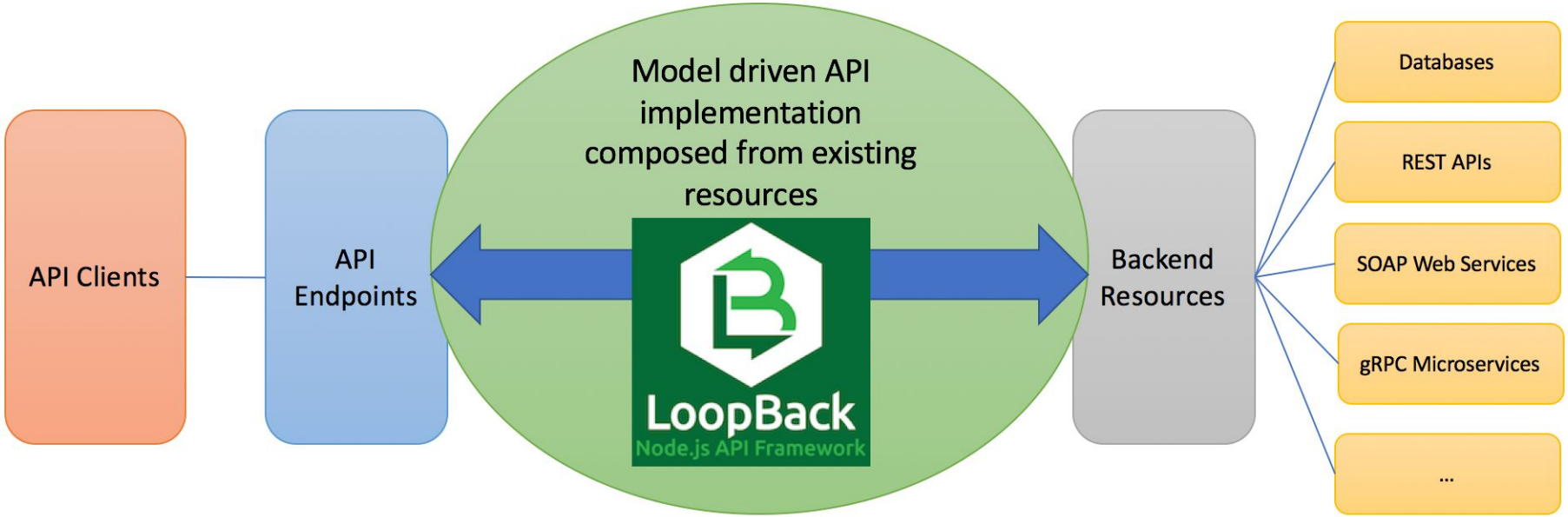
```
const http = require('http');
const port = 3000;

const requestHandler = (request, response) => {
 console.log(request.url);
 response.end('Hello Node.js Server!');
};

const server = http.createServer(requestHandler);

server.listen(port, (err) => {
 if (err) {
   return console.log('something bad happened', err);
 }

 console.log(`server is listening on ${port}`);
});
```

Then...

- Abstractions around `http` module
- Utility functions
- Faster
- No database

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (request, response) => {
 response.send('Hello from Express!');
});

app.listen(port, (err) => {
 if (err) {
    return console.log('something bad happened', err);
 }

 console.log(`server is listening on ${port}`);
});
```

LoopBack


API Clients
API Endpoints
Model driven API implementation composed from existing resources
LoopBack
Node.js API Framework
Backend Resources
Databases
REST APIs
SOAP Web Services
gRPC Microservices
...

What is LoopBack?

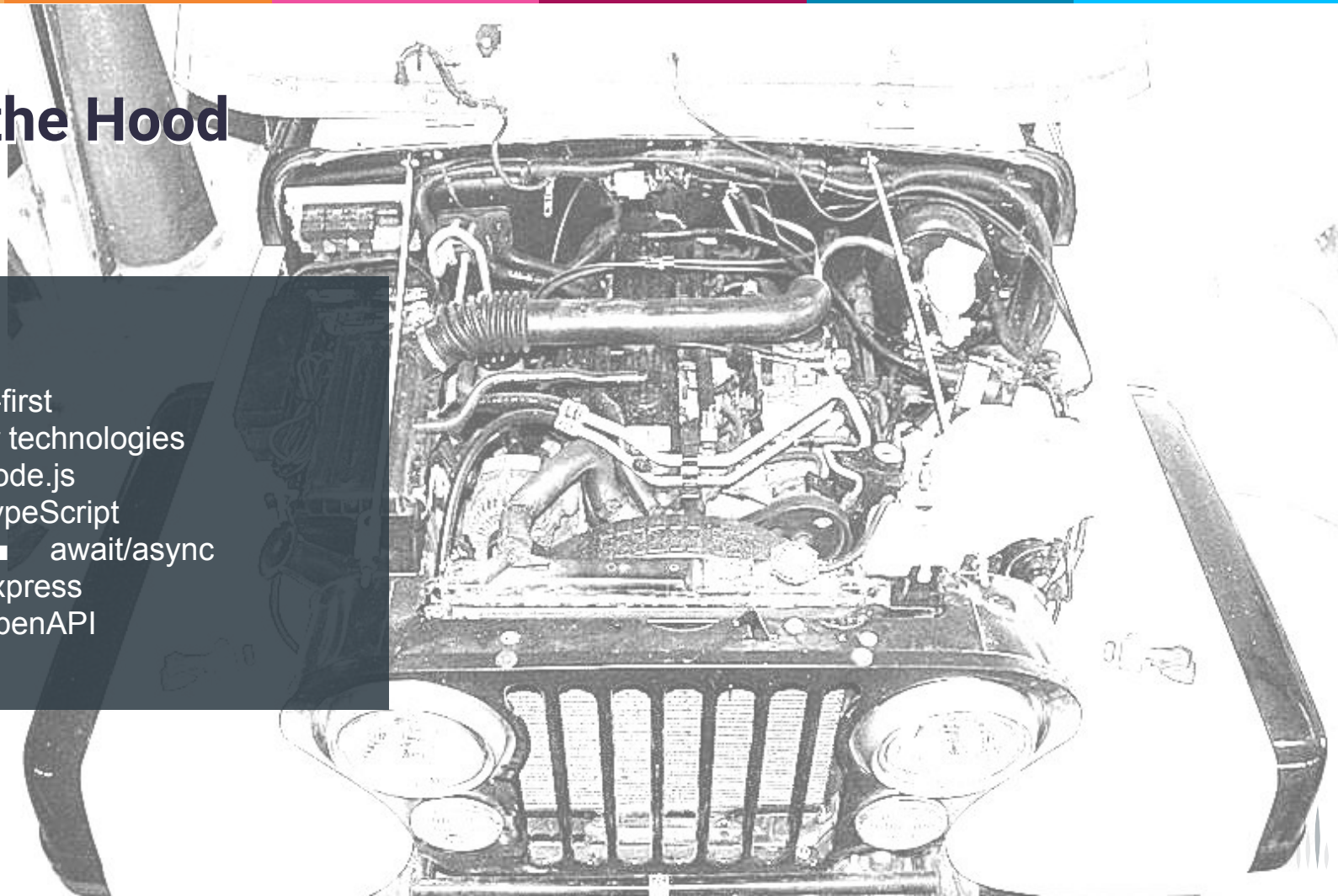
A framework to help quickly create and manage REST APIs as microservices using the OpenAPI standard.

Connect web clients to data and services.

Defines the models being used on either end of the request to be the source of truth.

Current version: 4.x

Under the Hood



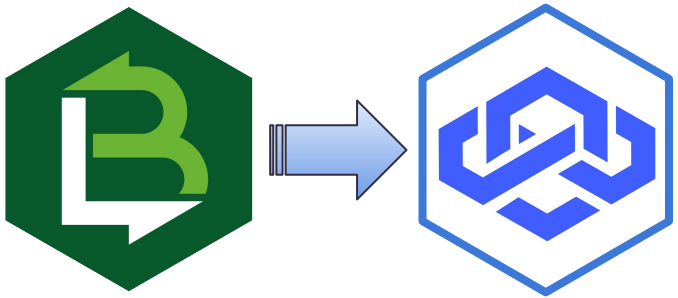
- MVC
- Testing-first
- Familiar technologies
  - Node.js
  - TypeScript
    - await/async
  - Express
  - OpenAPI

Why Should I Use LoopBack?

1. One source of truth
2. API composition layer
3. Strong typing
4. OpenAPI standard
5. Extensibility
6. Modern JavaScript

Why Upgrade from 3.x to 4.x?

- Built for microservices
- Code responsible for data access and manipulation is separated from the code responsible for implementing the REST API
- Models divided into controllers, repositories, models, and services
- Better promises support
- IoC Container with dependency injection

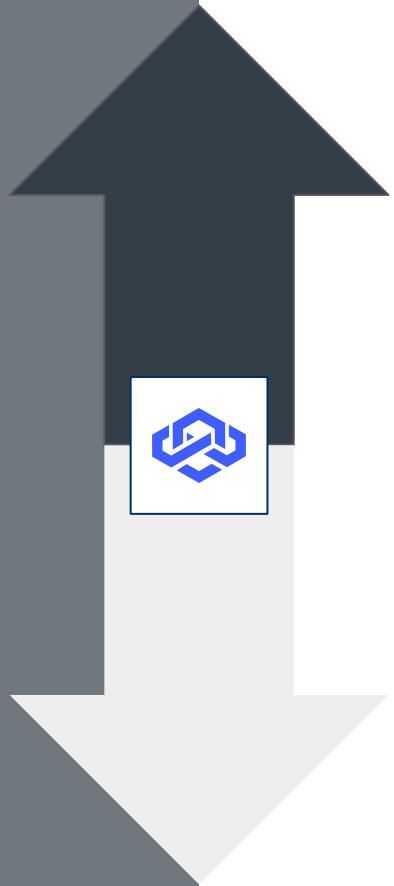


How Do I Use LoopBack?

Bottom-Up

Programmatic, code-first approach.

Export an OAS



Use the CLI, scaffold app (or extension) and generate artifacts.

Top-Down

How Do I Use LoopBack?

Outside-In

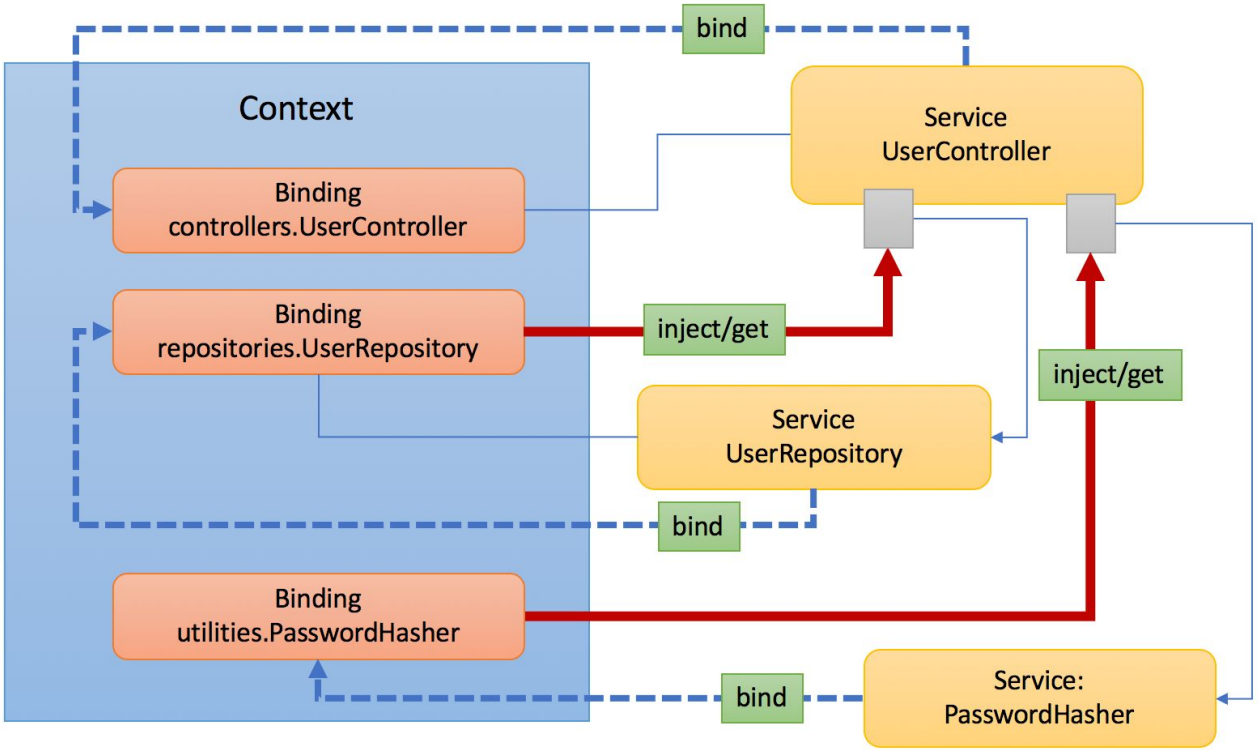
Export API's OAS spec

Inside-Out

Import OAS spec to create API artifacts

Key Concepts

Context



Key Concepts

Dependency Injection

```
// src/application.ts
import {Application} from '@loopback/core';
import * as fs from 'fs-extra';
import * as path from 'path';

export class MyApp extends RestApplication {
 constructor() {
   super();
   const app = this;
   app
     .bind('logger.widget')
     .to(logInfo);
 }
}
```

```
// src/controllers/widget.controller.ts
import {inject} from '@loopback/context';

export class WidgetController {
 // injection for property
  @inject('logger.widget');
 private logger: Function;

 constructor(
   // etc...
 ) {}
 // etc...
}
```

Key Concepts

Application, Server

Application - Stage; setting up components, controllers, servers, bindings. Extend, customize context.

Server - Start/stop/listening on a port.

Use `@loopback/rest` package for a default REST server

```
export class WidgetApplication extends Application {
 constructor() {
   super({
     // RestServer configuration
     rest: {
       port: 8080,
       apiExplorer: {
         url: 'https://petstore.swagger.io',
         httpUrl: 'http://petstore.swagger.io',
       },
     },
   });

   const app = this; // For clarity.
   // You can bind to the Application-level context here.
   // app.bind('foo').to(bar);
   app.component(RestComponent);
   app.controller(UserController);
   app.controller(ShoppingCartController);
 }
```

Key Concepts

Controllers, Routes

Handle incoming API requests

Business logic

Mapping between API spec and an operation

```
import {HelloRepository} from '../repositories';
import {HelloMessage} from '../models';
import {get, param, HttpErrors} from '@loopback/rest';
import {repository} from '@loopback/repository';


export class HelloController {
  constructor(
    @repository(HelloRepository) protected repo: HelloRepository
  ) {}


  @get('/messages')
  async list(
    @param.query.number('limit') limit = 10
  ): Promise<HelloMessage[]> {
    return await this.repo.find({limit});
  }
}
```

Key Concepts

Repositories

Interface that provide data access operations (ex: CRUD) of a domain model against the underlying database or service

```
export interface CrudRepository<T extends ValueObject | Entity>
  extends Repository<T> {
    create(
      dataObject: DataObject<T>,
      options?: Options
    ): Promise<T>;
    createAll(
      dataObjects: DataObject<T>[],
      options?: Options
    ): Promise<T[]>;
    find(
      filter?: Filter<T>,
      options?: Options
    ): Promise<T[]>;
    updateAll(
      dataObject: DataObject<T>,
      where?: Where<T>,
      options?: Options,
    ): Promise<Count>;
```

Key Concepts

Models

How to define schemas for business objects

(ex: customer, address, order).

Contain properties for name, type, and other constraints.

Used for data exchange.

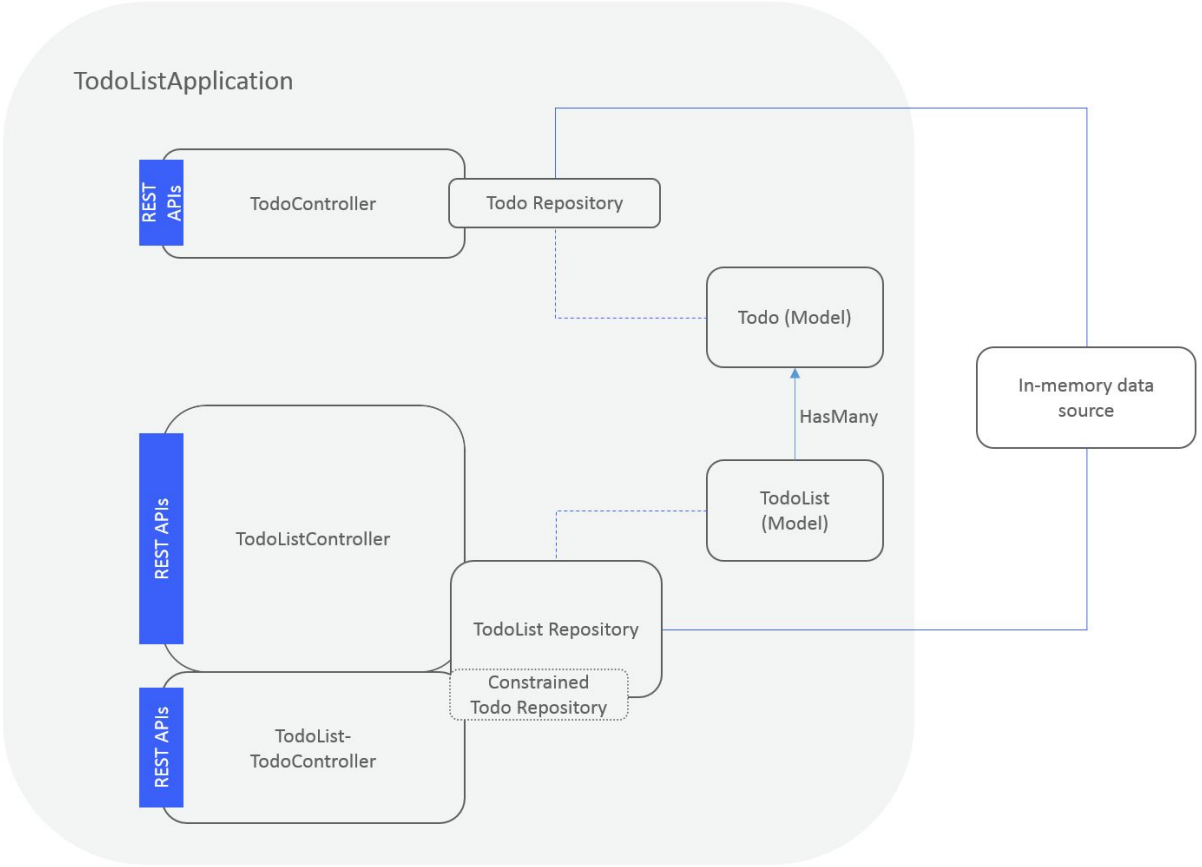
Value object (many) vs entity (unique)

```
import {
  model,
  property
} from '@loopback/repository';


@model()
export class Customer {
  @property()
  email: string;

  @property()
  isMember: boolean;

  @property()
  cart: ShoppingCart;
}
```

Key Concepts

- **DataSources:** A named configuration for a Connector instance that represents data in an external system.
- **Components**: Ways for 3rd parties to extend LoopBack as a self-contained package.
- **Services**: Interact with existing REST APIs, SOAP Web Services, and other forms of services/microservices
- **Sequences**: Extension of middleware

High-Level View



CLI

Projects
- Application
- Extension

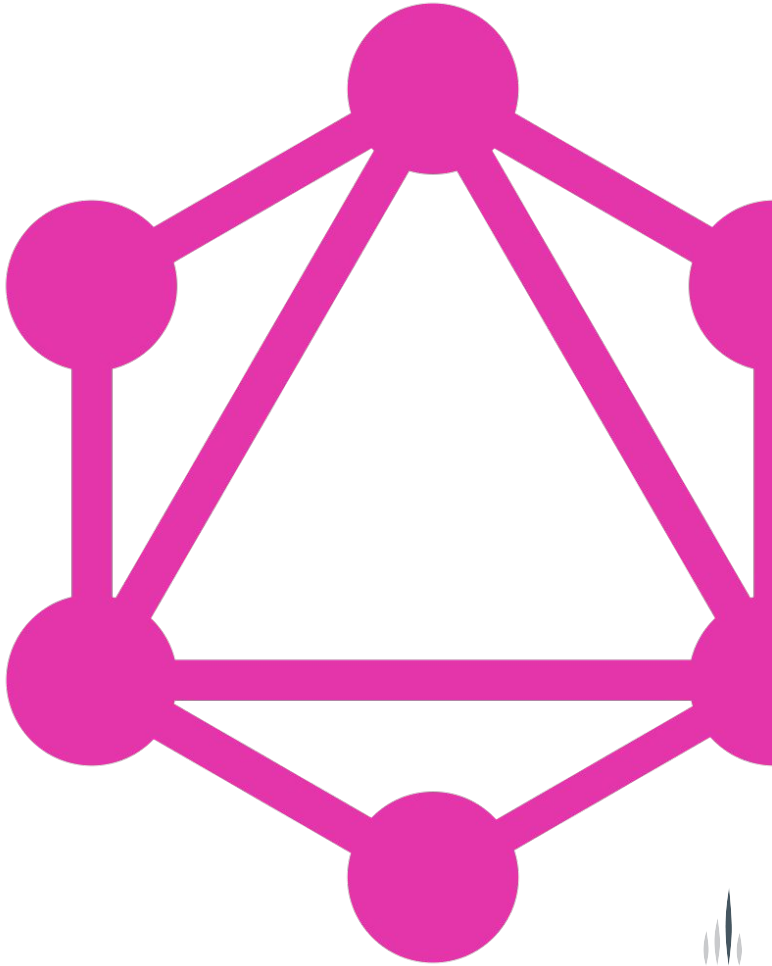
Artifacts
- Controller
- Datasource
- Model
- Repository
- Service
- OpenAPI


```
> npm install -g @loopback/cli
> lb4
> lb4 controller
```

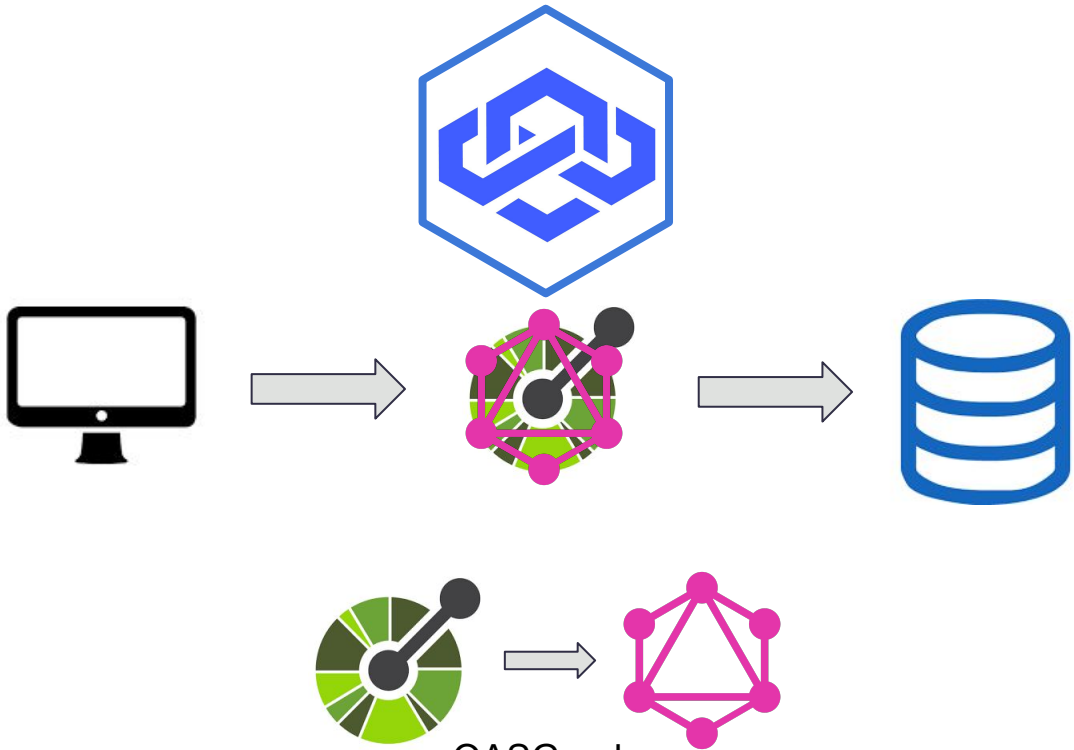
What About GraphQL?

GraphQL is a query language for APIs

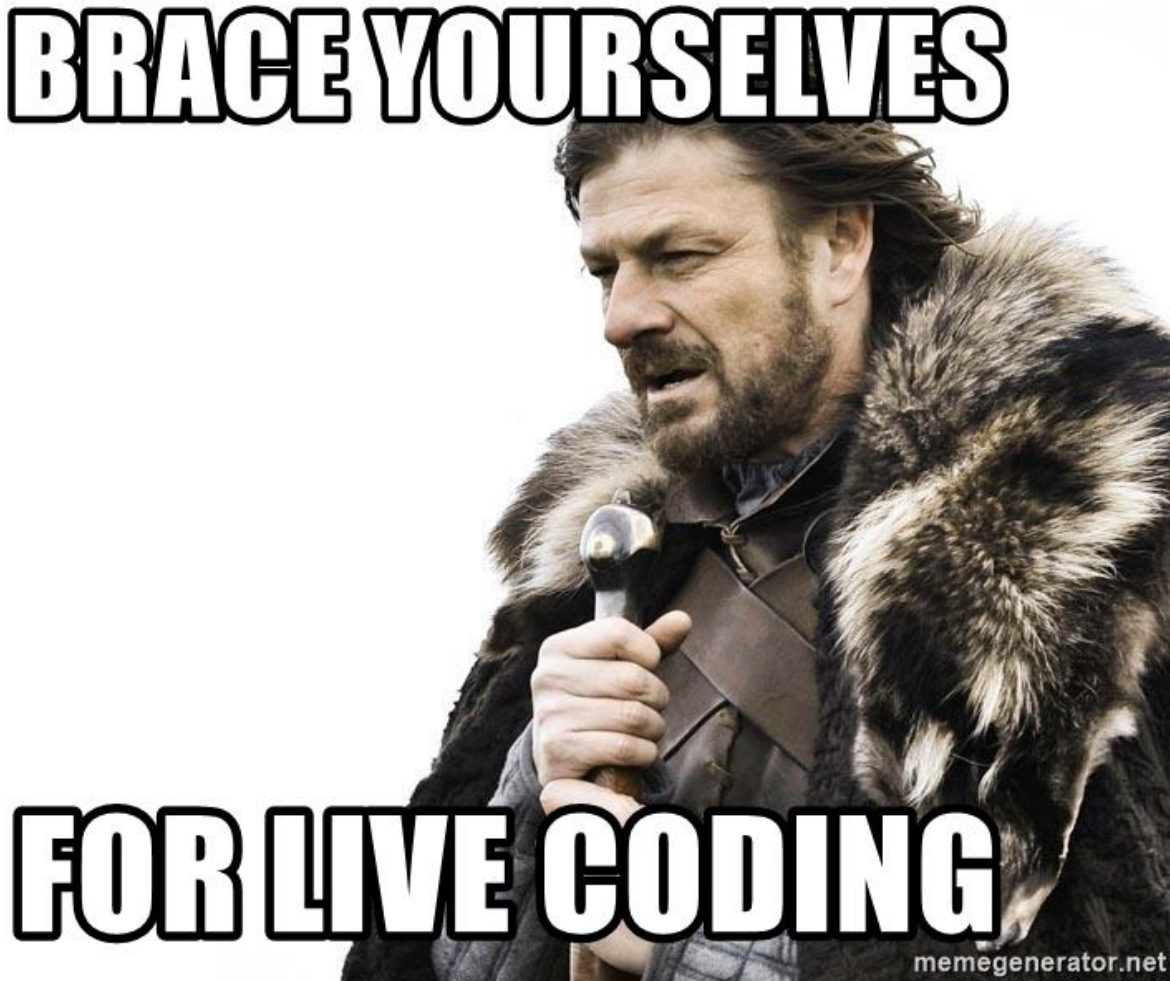
- No over-fetching of data
- Get many resources in a single request
- Community support
- Language/protocol-agnostic
- Strongly-typed schema



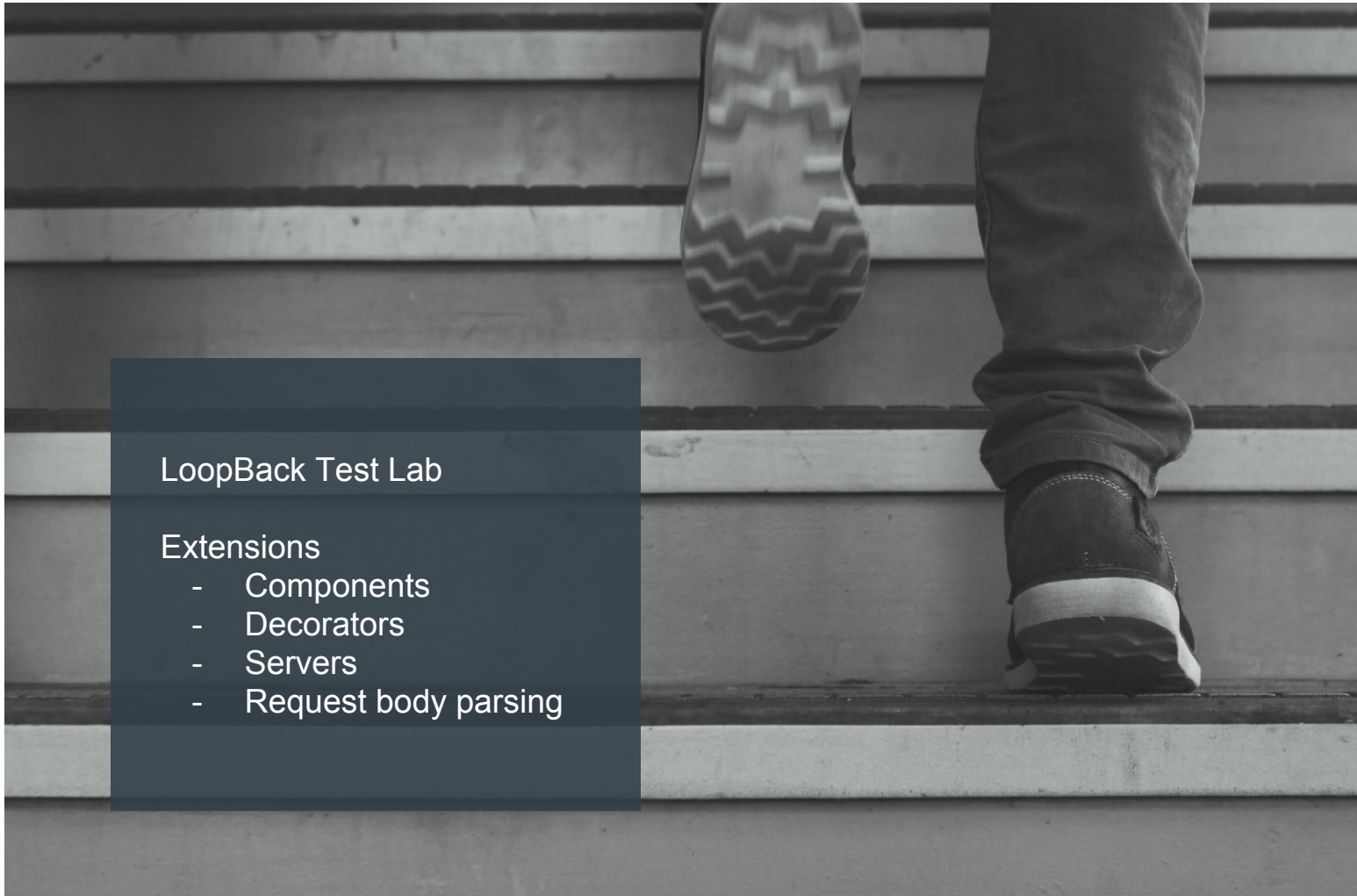
What About GraphQL?



Demo Time!



Going Further



LoopBack Test Lab

Extensions
- Components
- Decorators
- Servers
- Request body parsing

Recap

- History
- LoopBack is a declarative framework to help quickly compose and manage REST APIs as microservices using the OpenAPI standard.
- One source of truth to connect web clients to data and services.
- Key components
- GraphQL, OpenAPI specs, and OASGraph

Modern Node.js API Development with LoopBack 4

Jake Brauchler
jbrauchler@kenzan.com

Craig Freeman
cfreeman@kenzan.com
@craigfreeman

Resources:

- LoopBack
- v3 to v4 Migration
- LoopBack Test Lab
- Demo code