# servant

Defining web APIs at the type-level

February 1, 2016

Introduction

### TYPE-LEVEL DSLs?

- · (Uninhabited) types with attached "meaning"
- The Expression Problem (Wadler 1998)
- · API representation and interpretation are separated
- APIs become first-class citizens

## HASKELL EXTENSIONS

- TypeOperators
- DataKinds
- TypeFamilies

#### A SERVANT EXAMPLE

#### **COMPUTED TYPES**

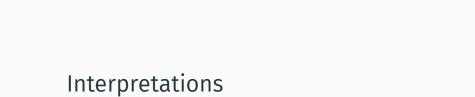
```
type TaggedPubs = "tagged" :> Capture "tag" Text :> ...
taggedPubsHandler :: Server TaggedPubs
taggedPubsHandler tag = ...
```

#### **COMPUTED TYPES**

```
type TaggedPubs = "tagged" :> Capture "tag" Text :> ...

taggedPubsHandler :: Server TaggedPubs
taggedPubsHandler tag = ...

Server TaggedPubs ~
Text -> EitherT ServantErr IO [Pub]
```



#### **SERVANT-SERVER**

The one everyone is interested in!

- · Based on WAI, can run on warp
- Interprets combinators with a simple HasServer c class
- · Easy to use!

#### HASSERVER ...

#### SERVER EXAMPLE

```
type Echo = Capture "echo" Text :> Get [][PlainText] Text
echoAPI :: Proxy Echo
echoAPI = Proxy

echoServer :: Server Echo
echoServer = return
```

#### **SERVANT-CLIENT**

- · Generates Haskell client functions for API
- Same types as API specification: For RPC the whole "web layer" is abstracted away
- Also easy to use!

# SERVANT-DOCS, SERVANT-JS ...

Many other interpretations exist already, for example:

- · Documentation generation
- Foreign function export (e.g. Elm, JavaScript)
- Mock-server generation

# Demo

# Conclusion

#### DRAWBACKS

- Haskell has no custom open kinds (yet)
- · Proxies are ugly
- Errors can be a bit daunting

# QUESTIONS?

Ølkartet: github.com/tazjin/pubkartet Slides: github.com/tazjin/servant-presentation

@tazjin