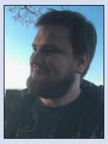
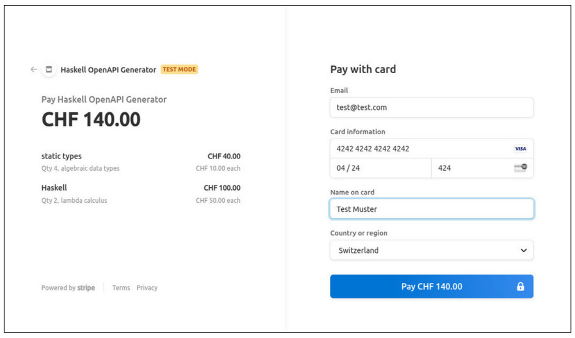
Joel
Fisch


Remo
Dörig

Graduate Candidates	Joel Fisch, Remo Dörig
Examiner	Prof. Dr. Farhad D. Mehta
Co-Examiner	Tom Sydney Kerckhove, CS Kerckhove, Zürich, ZH
Subject Area	Software

OpenAPI 3 code binding generator for Haskell

And its Application to Generate a Library for the Stripe Payment System


Haskell OpenAPI Generator TEST MODE

Pay Haskell OpenAPI Generator
CHF 140.00

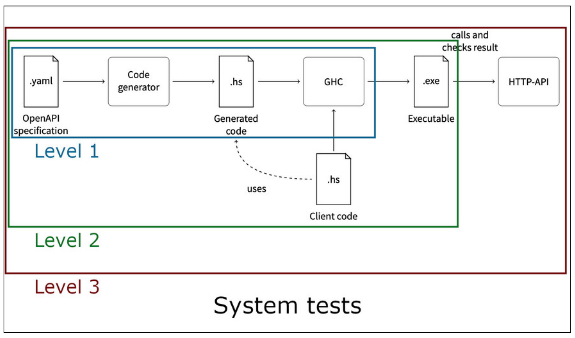
static types	CHF 40.00
Qty 4, algebraic data types	CHF 10.00 each
Haskell	CHF 100.00
Qty 2, lambda calculus	CHF 50.00 each

Powered by stripe | Terms Privacy

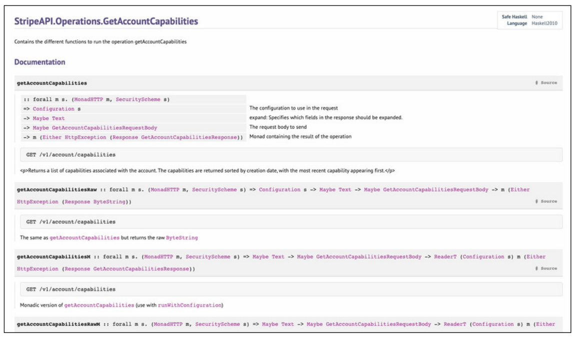
Pay with card

Email: test@test.com
Card information: 4242 4242 4242 4242 VISA, 04 / 24, 424
Name on card: Test Muster
Country or region: Switzerland

The generated Stripe library can be used to implement payments on a website.
Own presentment



The three system test levels depend on each other and add additional checks on every level.
Own presentment


StripeAPI.Operations.GetAccountCapabilities

Safe Haskell: None
Language: Haskell2010

Contains the different functions to run the operation getAccountCapabilities

Documentation

```
getAccountCapabilities
  :: forall m s. (MonadHTTP m, SecurityScheme s)
  => Configuration s                                             The configuration to use in the request
  -> Maybe Text                                                  expand: Specifies which fields in the response should be expanded.
  -> Maybe GetAccountCapabilitiesRequestBody                     The request body to send
  -> m (Either HttpException (Response GetAccountCapabilitiesResponse))   Monad containing the result of the operation
```

```
GET /v1/account/capabilities
```

<p>Returns a list of capabilities associated with the account. The capabilities are returned sorted by creation date, with the most recent capability appearing first.</p>

```
getAccountCapabilitiesRaw :: forall m s. (MonadHTTP m, SecurityScheme s) => Configuration s -> Maybe Text -> Maybe GetAccountCapabilitiesRequestBody -> m (Either HttpException (Response ByteString))
```

```
GET /v1/account/capabilities
```

The same as getAccountCapabilities but returns the raw ByteString

```
getAccountCapabilitiesM :: forall m s. (MonadHTTP m, SecurityScheme s) => Maybe Text -> Maybe GetAccountCapabilitiesRequestBody -> ReaderT (Configuration s) m (Either HttpException (Response GetAccountCapabilitiesResponse))
```

```
GET /v1/account/capabilities
```

Monadic version of getAccountCapabilities (use with runWithConfiguration)

```
getAccountCapabilitiesRawM :: forall m s. (MonadHTTP m, SecurityScheme s) => Maybe Text -> Maybe GetAccountCapabilitiesRequestBody -> ReaderT (Configuration s) m (Either
```

For the generated code, a documentation can be generated (one part of the Stripe documentation is visible above).
Own presentment

Initial Situation: OpenAPI 3 is a machine-readable specification standard which can be used to define the interface of (web-)services (also known as API). This allows clients to talk to the corresponding service and retrieve data which conforms to the OpenAPI specification of the service. OpenAPI is the de facto standard in the industry and many providers of APIs publish such a specification. Stripe, a big online payment provider, is such a company. As OpenAPI is formally specified, it is possible to generate code for client implementations. There are generators for many languages, but for some languages there are either none or none which are working. Haskell, a purely functional language, is one of the aforementioned languages. In addition, there is currently no working Haskell library for the newest version of the Stripe API specification. If a generator for OpenAPI 3 would exist, such a library could be implemented with low effort. Therefore, the goal of this thesis is to implement such a code generator which is able to transform an OpenAPI 3 specification into Haskell code, using Haskell as the implementation language as well.

Result: The previously mentioned code generator was implemented as part of this thesis. It is able to transform OpenAPI 3 specifications to working Haskell code which can be used by client applications to retrieve data from an API. As a second result, a Haskell library for Stripe was generated and published. This library can easily be updated in the future, if Stripe publishes a new specification for their API in form of OpenAPI 3. With this newly created library, a demo use case was implemented to showcase how the generated code can be used. Furthermore, to ensure the correct functionality of the generator, multiple types of automated tests were created. On the one hand, unit and property tests were used to check the correctness of single functions in the code, on the other hand, multiple levels of system tests were developed. As the generator is a command-line interface application, the system tests could check the generated code on three levels: on a first level, code generation and compilation are checked; on a second level, code integration with other client code using the generated code is ensured; on a third level, real HTTP calls are executed using the generated code. To allow users to discover the different functions and types generated, code documentation is generated using Haddock-comments directly in the code. These comments include meta information retrieved from the specification and can be extracted and viewed with the tool Haddock, used widely in the Haskell community.

Conclusion: Within this bachelor thesis, a code generator could be implemented which can be used to generate code bindings for Haskell. This allows to update the generated Stripe library (and other libraries using the generator) easily and with low maintenance effort in the future. Different members of the Haskell community either plan to use the library to implement commercial software or plan to join forces to create an even better version of the code generator.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz