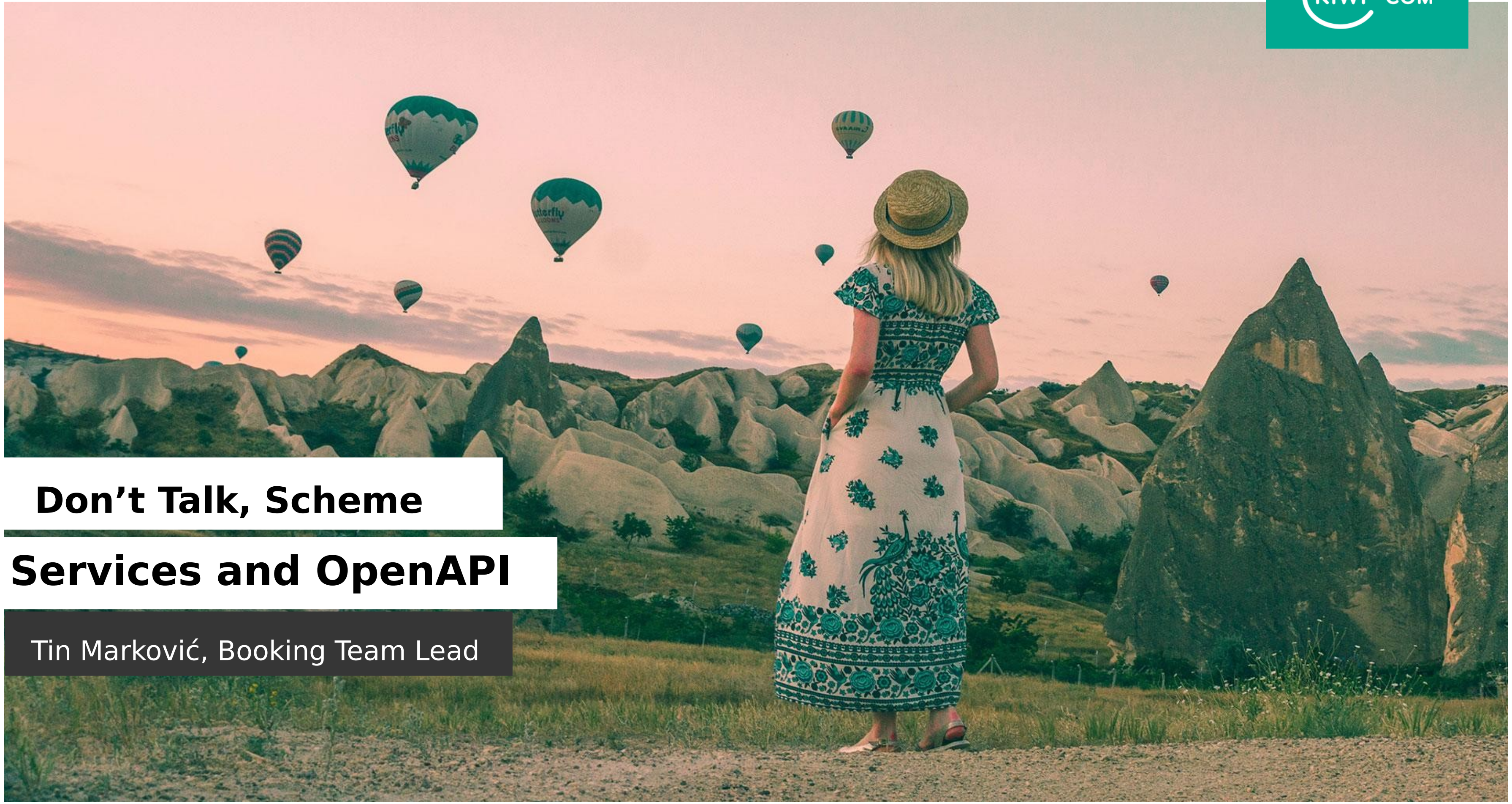

Don’t Talk, Scheme

Services and OpenAPI

Tin Marković, Booking Team Lead

Introduction

- Presentation and Examples
- Services are part of everyday work
- Architecture independent talk
- Very technical
- Principles apply outside of examples

Speaker

- https://tinthe.dev
- Tin
  - Team Lead at Kiwi.com
  - Software Architecture as passion
  - Experiences working big projects (edX, Texas U, Kiwi)
- Where do I fit in the Kiwi picture?

Company: Kiwi.com

- Making travel better
- Our vision is to make travelling simple and accessible to everyone
- Stack: Python and friends :)
- Very technologically oriented

Abstract

- Services are developed by different people
- Communicating code is less safe than sharing it programmatically
- Protect yourself from misunderstandings and problems
- Keep many different services, but share important code
- Giant consistency and cooperation wins, easier maintenance

Overview: Presentation

- Schema and how to do it properly
- API-first and consistent, change-resistance
- Schema language and tooling
- Enforced or useless, Connexion
- Service based architectures, Flask
- Sharing schema, module approach
- Conclusions, how does it all work together

Overview: Examples

- Python, using Connexion (Flask + Swagger/OpenAPI)
- Birds as motif, watching and keeping
- Made up example, real use-cases

API

- Application Programming Interface
- Most often used in context of web
- API is a product, treat it as such

Schema

- The word schema comes from the Greek word *σχήμα (skhēma)*
- Means shape, or more generally, plan
- You wouldn't let a doctor "*wing*" your operation
- Plan first, do after
- Concisely shape ideas and structure APIs

Planning with Schema

- Adopt an API-first approach, scheme it out, then program
- Get early review feedback from peers and client developers
- Clear separation of WHAT vs. HOW concerns
- "agile" may be good, "reckless" is def. not

Schema in OpenAPI/Swagger

- OpenAPI formerly Swagger Specification
- Standard, language-agnostic interface to RESTful APIs
- Basic concepts:
  - parameters
  - paths
  - definitions
  - metadata/headers
  - request/response
  - references ($ref)

Example: Schema of our bird app

< show compiled and formatted schema >

Tooling of OpenAPI/Swagger

- Swagger UI
- Online resources
- Wide community, all over
- Connexion
- YAML or JSON

Why YAML

- More human approachable
- Less programmer noise
- Easy to use for tech writers
- Opinion/Taste

Connexion

- Framework on top of Flask
- Automatically handles HTTP requests based on OpenAPI/Swagger
- API described in YAML format

Flask

- Python library/framework
- Light, fast, minimally opinionated
- Rich tooling, flexibility

Connexion and Flask

- Allow many smaller and flexible services
- Allow for as little bloat as possible
- Connexion for parsing and working schema
- Simplicity allows for custom code

Enforced or useless

- Schema is a great design and plan
- Plans fall apart, documentation as well
- Programmaticaly enforce API schema
- Unless it passes, it doesn't work
- Forget about outdated schema

Example: App setup and framework

< overview of the simple app >

Schema sharing: Reasoning

- Use bundling code to reuse schema
- Reasons:
    - Having a separate repo/module
    - Ownership of API is more explicit
    - Review schema independent of code
    - Assure contract is intact

Schema sharing: Execution

- Bundling several schemata together
- Parsing them as a single entity
- Full link expansion (new!)

Example: Schema sharing

< demo of schema bundling >

Schema sharing: Future

- How to do even more?
- Multi-team access — generate code
- Separation of concerns
  - Keep schema versioned and secure
- Linting — speccy
  - Lint the schema, so it's up to standard
  - Share best patterns with the world/team

Product

- Multiple services using the same schema elements to work
- Bound via code integration and tests
- Important schema changes can be audited more carefully
- Terminology matches the actual production code

Example: Birdy Product

< demo the two applications >

< demo reusing returned objects >

Conclusions

- API and schema first, it's important part of product
- Force things via code, it will then self-validate
- Reuse and DRY terminology/schema, not just code
- Tight interface will pay for itself many times over

How to Start

- Call to Action: Use tools, expand them
- Adapt your applications tools for Schema:
    - OpenAPI or JSONschema or...
- A lot of it is process - establish it!
- Separate schema into separate repo and version it
- API specification from "BS" - Phil Sturgeon
    - https://blog.apisyouwonthate.com/creating-api-specifications-from-bulls-t-f5a54c005135

Resources

- Example application repo:
    - https://tinthe.dev/talks/schema-composition
    - https://github.com/TinMarkovic/dont_talk_scheme
- Keywords:
    - OpenAPI, Swagger, Connexion, Flask, Speccy
- Useful links:
    - blog.apisyouwonthate.com
    - opensource.zalando.com/restful-api-guidelines

Thanks!

ANY QUESTIONS?

You can find me at tinthe.dev