# Model-Based Approach to Consume REST Services in Single Page Applications

Niklas Scholz - Final Presentation Master Thesis - 09.10.2017

Advisor: Adrian Hernandez-Mendez

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

# Agenda

Motivation and Problem

Research Questions

Background

Approach

- Reference Architecture
- Meta-Model
- Code Generation Process

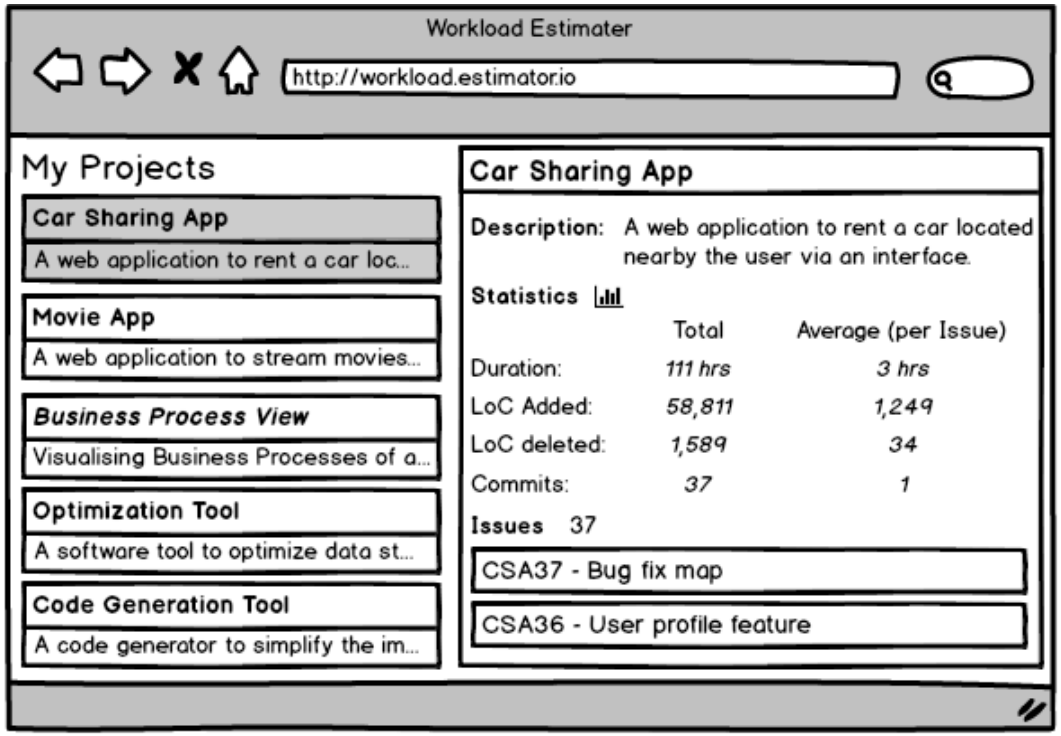
Live Demo

Evaluation



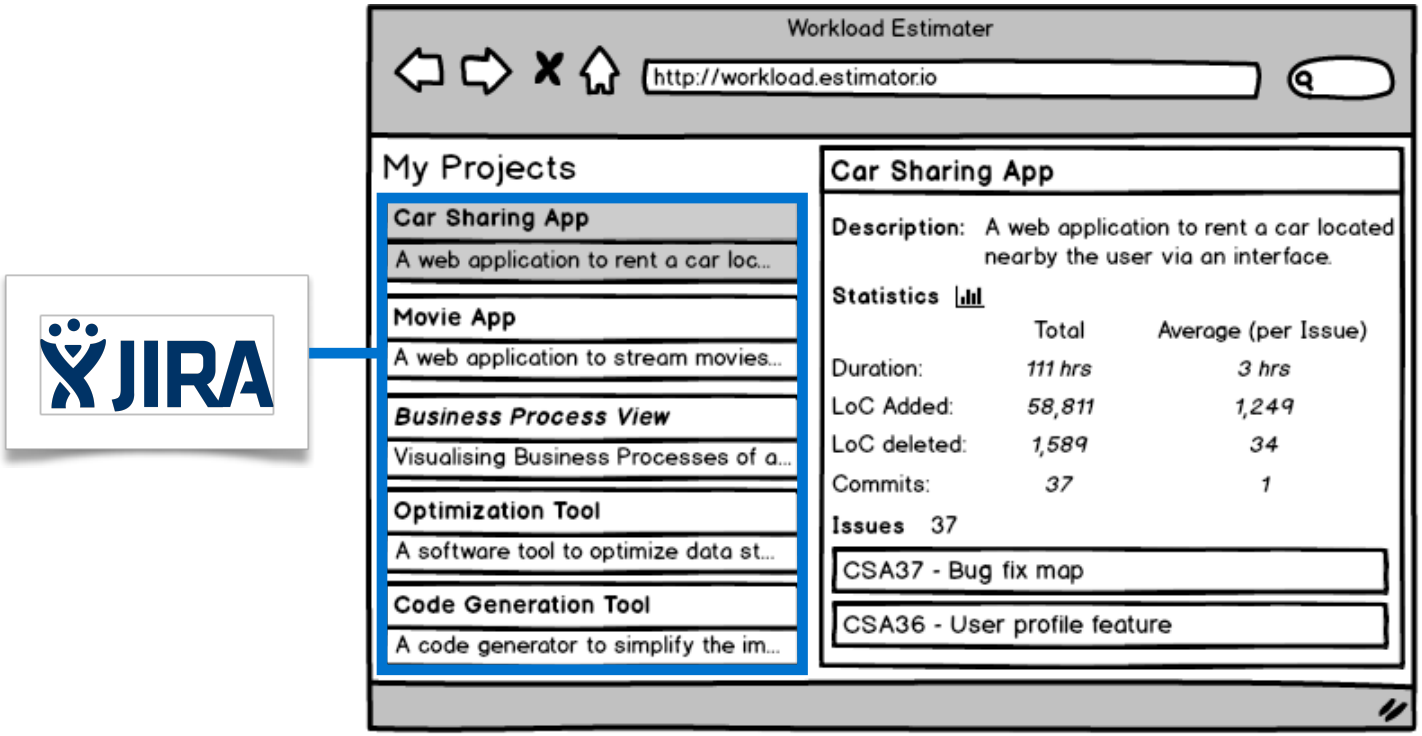
# Building an Application in 2017





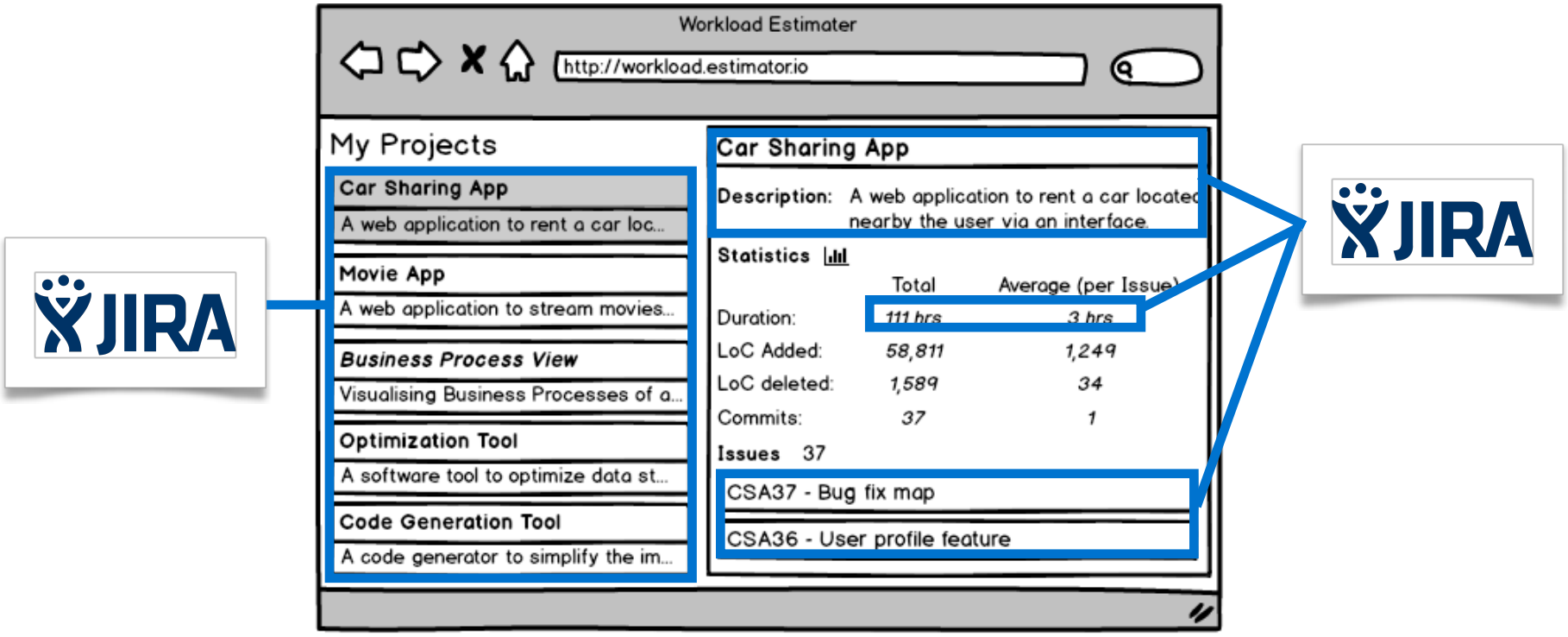
# Building an Application in 2017





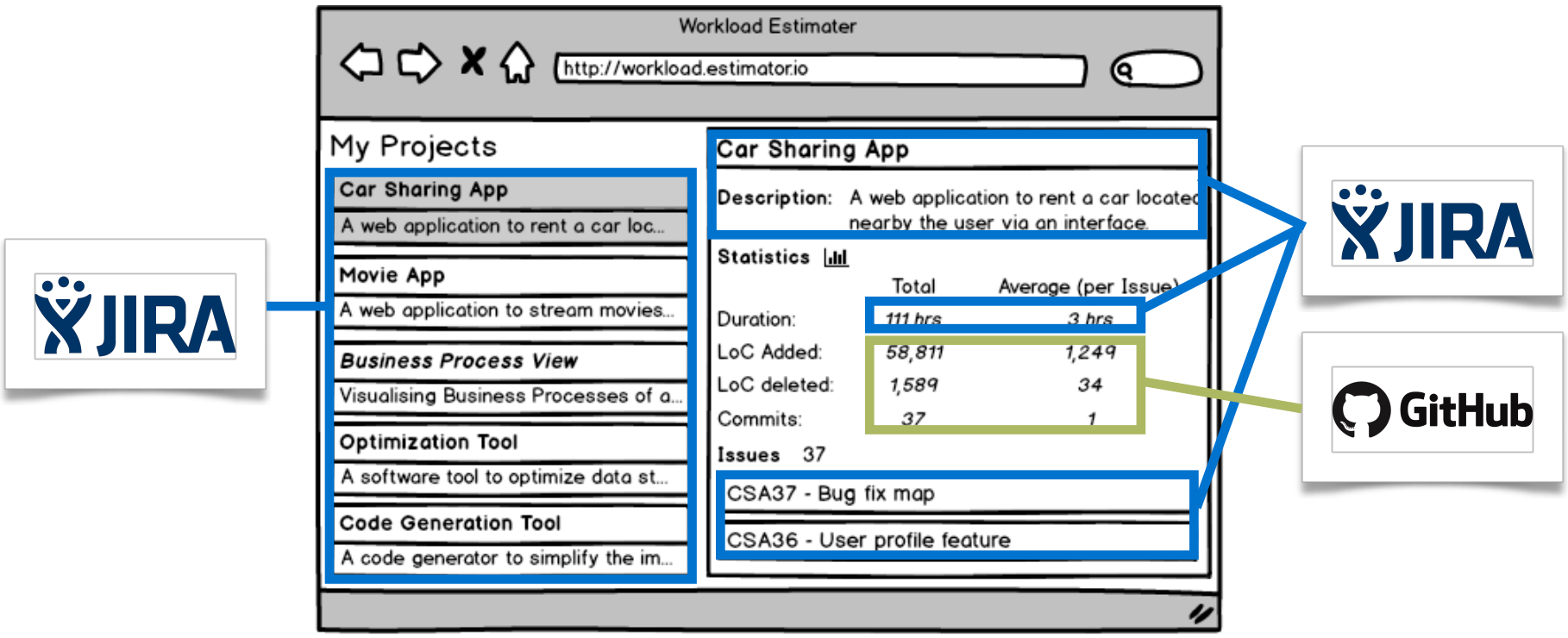
# Building an Application in 2017





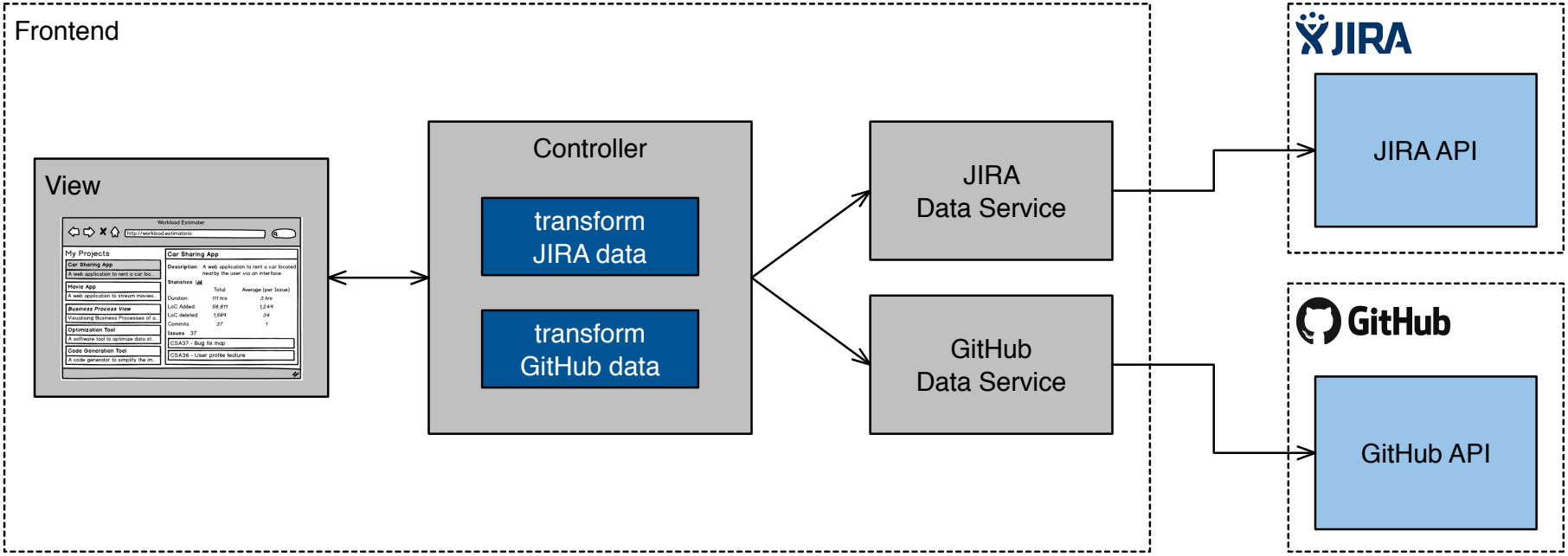
# Building an Application in 2017





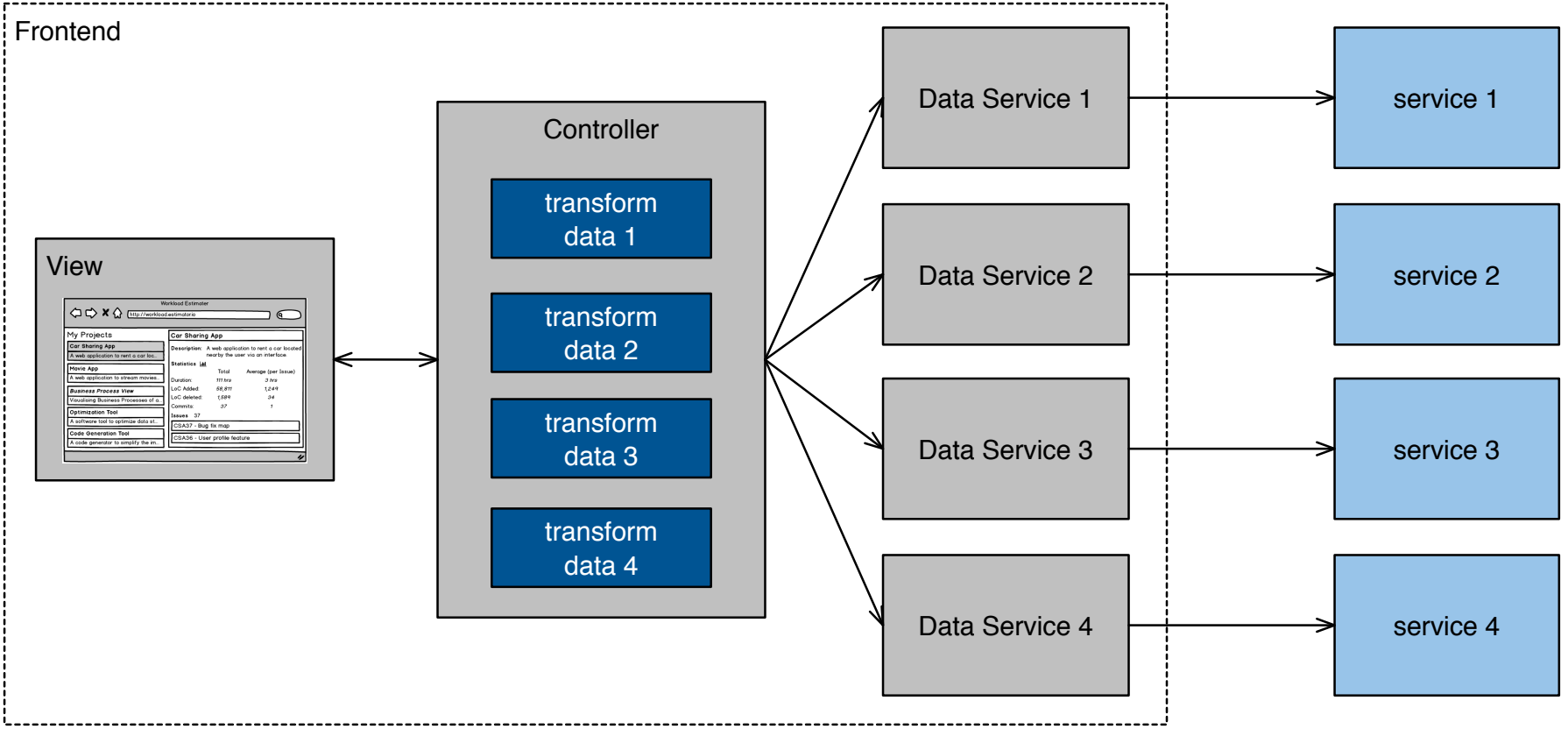
# Consuming APIs in Single Page Applications





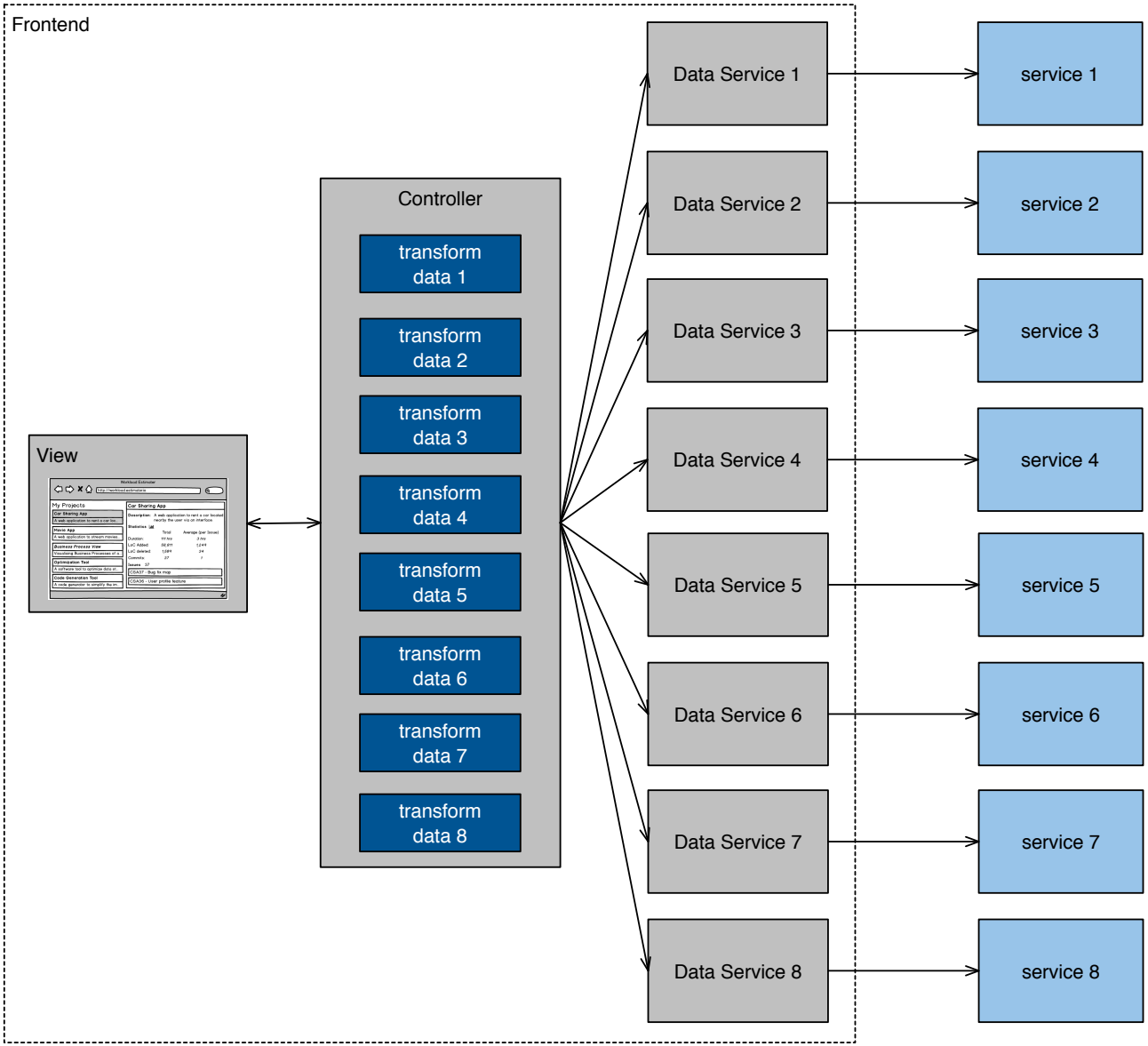
# Scaling up number of APIs to consume





# Scaling up number of APIs to consume





# Research Questions

**RQ1** **What is the state-of-the-art in model-based RESTful API integration?**



# Research Questions

**RQ1** **What is the state-of-the-art in model-based RESTful API integration?**

**RQ2** **How does a model-based approach for RESTful API integration in single page applications look like?**



# Research Questions

**RQ1** What is the state-of-the-art in model-based RESTful API integration?

**RQ2** How does a model-based approach for RESTful API integration in single page applications look like?

**RQ3** What are the benefits and limitations of a model-driven approach for RESTful API integration?



# Analysing related work

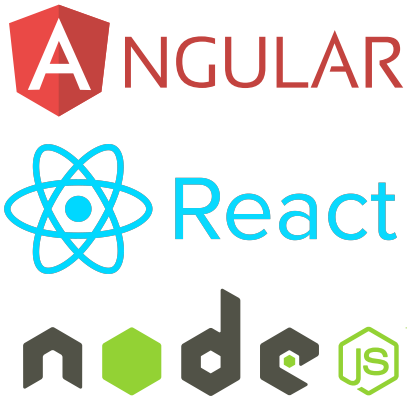
- Lanthaler and Gütl: Hydra [1]
  - RESTful API description language
- Rossi: Model-driven REST API development [2]
  - Constricting model in UML and transforming it to RAML
- Modelling REST with the Eclipse Modelling Framework (EMF)
  - Ed-Douibi et al.: Ready-to-run web APIs out of data models [3]
  - Haupt et al.: Generate REST compliant services [4]
- Bonifacio et al.: NeoIDL [5]
  - domain specific language to specify RESTful APIs



# Analysing the state-of-the-art

**Web development frameworks**




**API Design Guidelines**


**API Description**


**API Management Tools**



# Process Steps

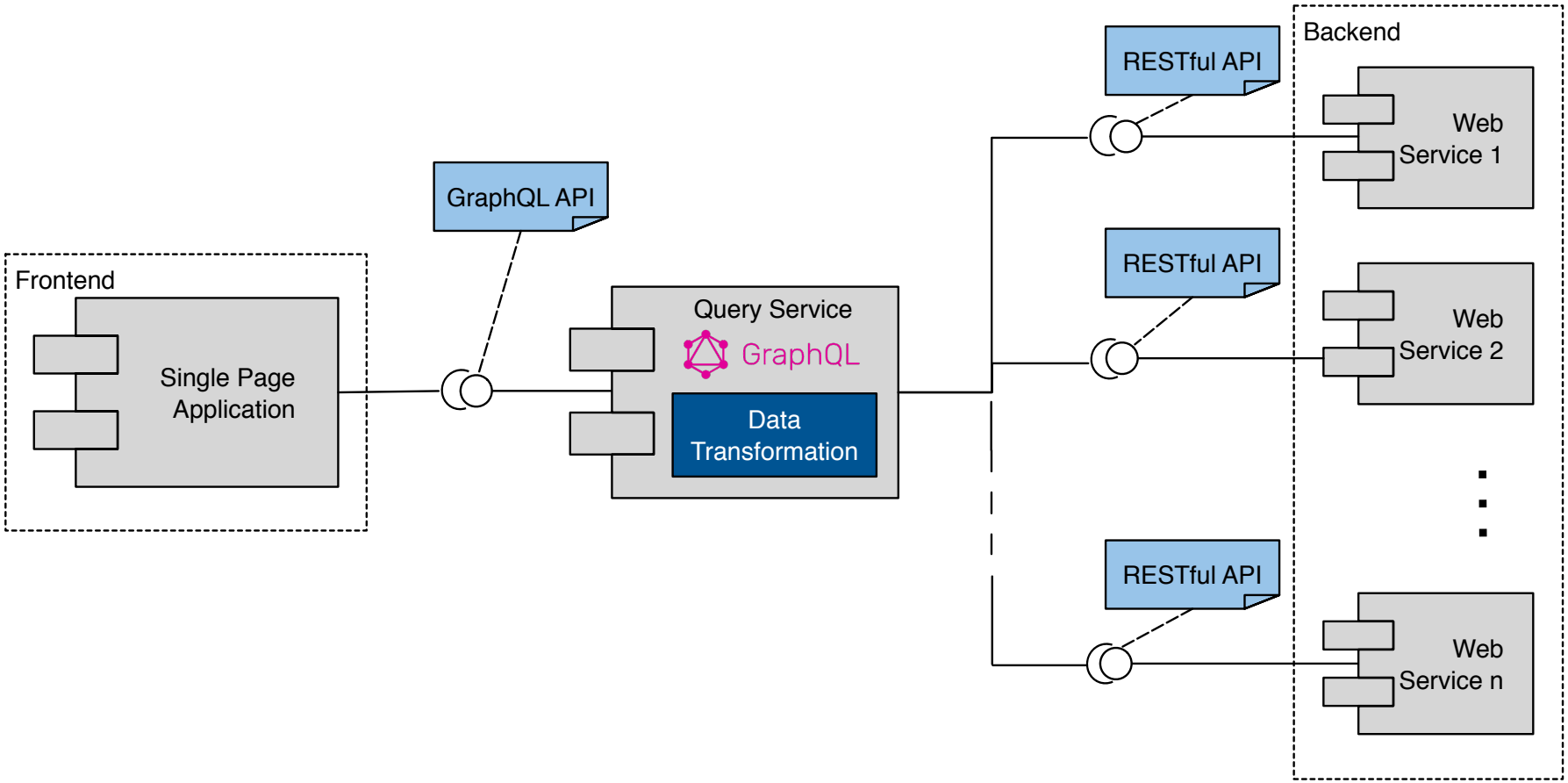
## Reference Architecture

- Architecture to reduce complexity in client
- shift responsibility of RESTful API consumption outside of client



# Reference Architecture



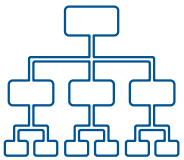


# Process Steps

**Reference Architecture**

- Architecture to reduce complexity in client
- shift responsibility of RESTful API consumption outside of client



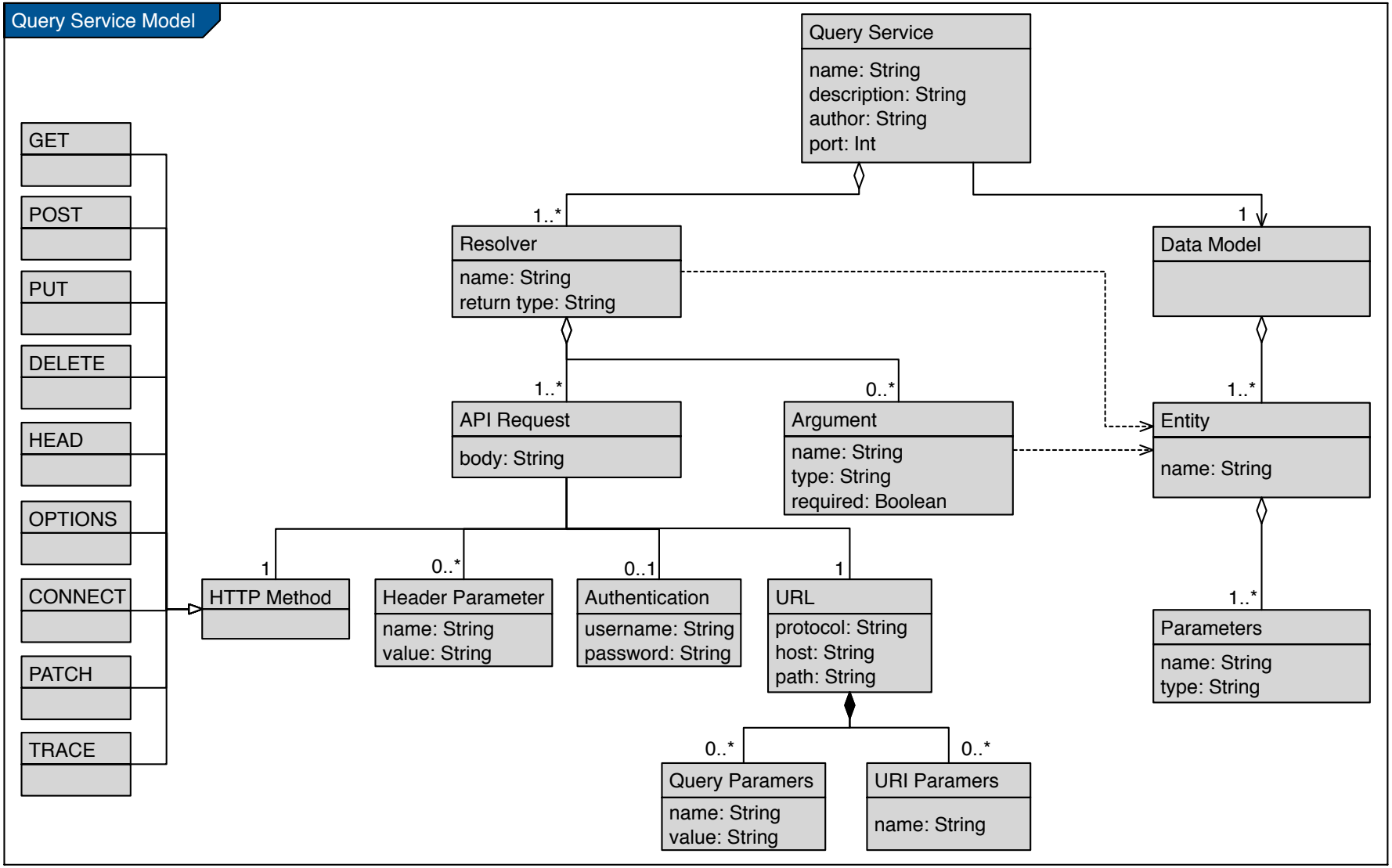
**Meta-Model**

- Establishing meta-model describing consumption of RESTful services



# Meta-Model



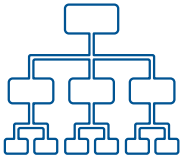


# Process Steps

**Reference Architecture**

- Architecture to reduce complexity in client
- shift responsibility of RESTful API consumption outside of client



**Meta-Model**

- Establishing meta-model describing consumption of RESTful services

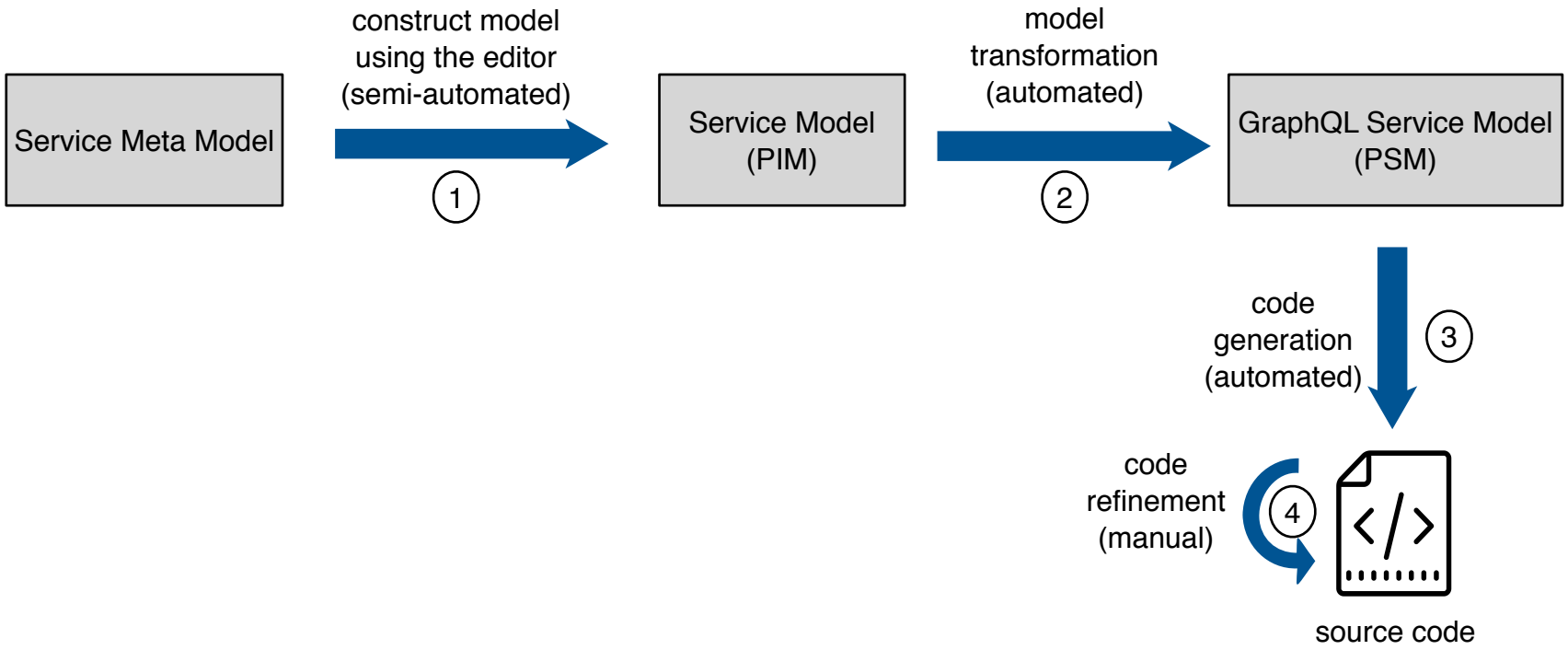
**Generation Process**

- Development of model-driven process to automatically generate RESTful API consumption
- Based on meta-model



# Code Generation Process



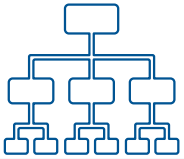


# Process Steps

**Reference Architecture**

- Architecture to reduce complexity in client
- shift responsibility of RESTful API consumption outside of client



**Meta-Model**

- Establishing meta-model describing consumption of RESTful services

**Generation Process**

- Development of model-driven process to automatically generate RESTful API consumption
- Based on meta-model

**Imple-mentation**

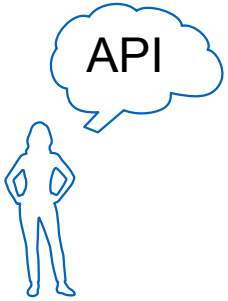
- Implementation of process as a web application
- React app



# Code Generation Tool - Workflow

**Developer**



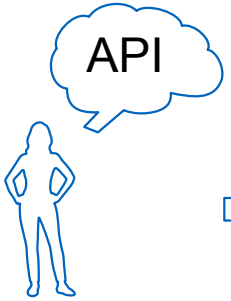

1. What data and where to get it from?

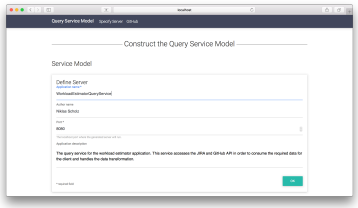


# Code Generation Tool - Workflow

**Developer**




1. What data and where to get it from?

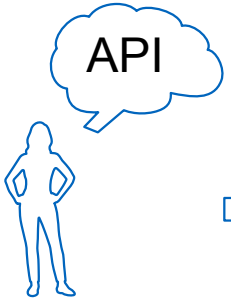
2. Construction of model



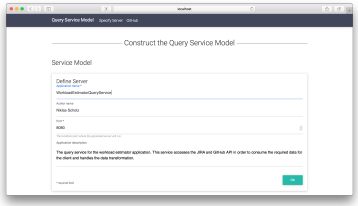
 

# Code Generation Tool - Workflow

**Developer**




1. What data and where to get it from?




2. Construction of model

generation of query service

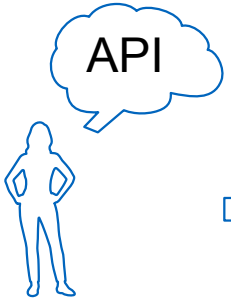
3. Manual code refinement

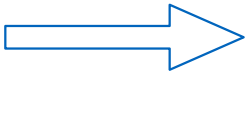


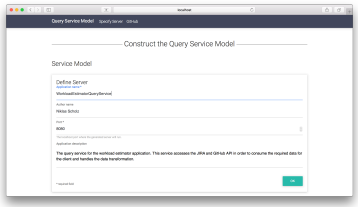
 

# Code Generation Tool - Workflow

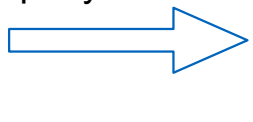
## Developer






generation of query service

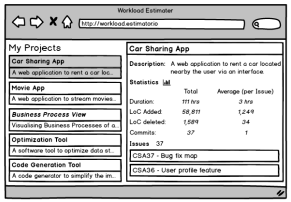


1. What data and where to get it from?

2. Construction of model

3. Manual code refinement

## UI Designer



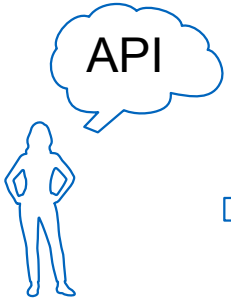

4. Design UI

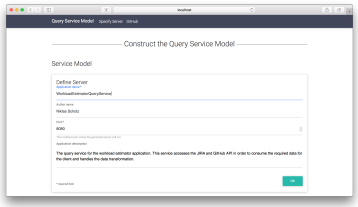


# Code Generation Tool - Workflow

## Developer



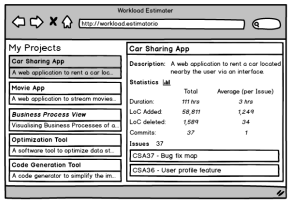

generation of query service

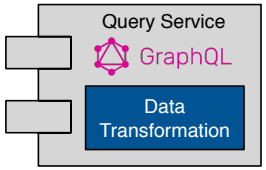
1. What data and where to get it from?

2. Construction of model

3. Manual code refinement

## UI Designer



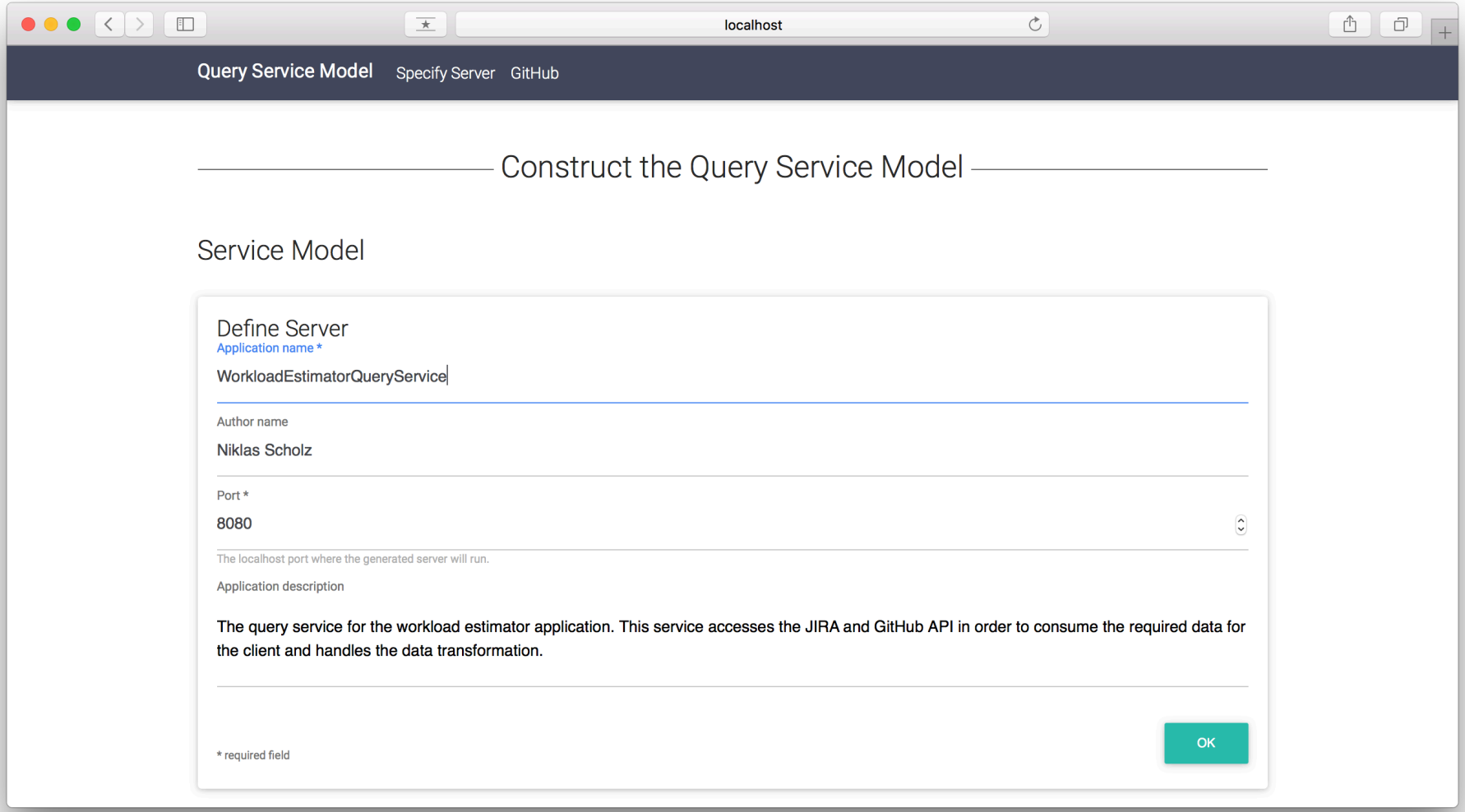

4. Design UI

5. Access query service to receive data



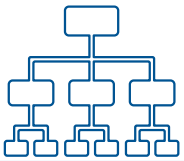
# Code Generation tool - Live Demo



# Process Steps

## Reference Architecture

- Architecture to reduce complexity in client
- shift responsibility of RESTful API consumption outside of client

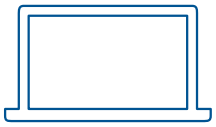


## Meta-Model

- Establishing meta-model describing consumption of RESTful services

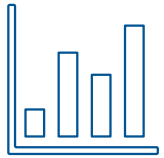
## Generation Process

- Development of model-driven process to automatically generate RESTful API consumption
- Based on meta-model



## Imple-mentation

- Implementation of process as a web application
- React app



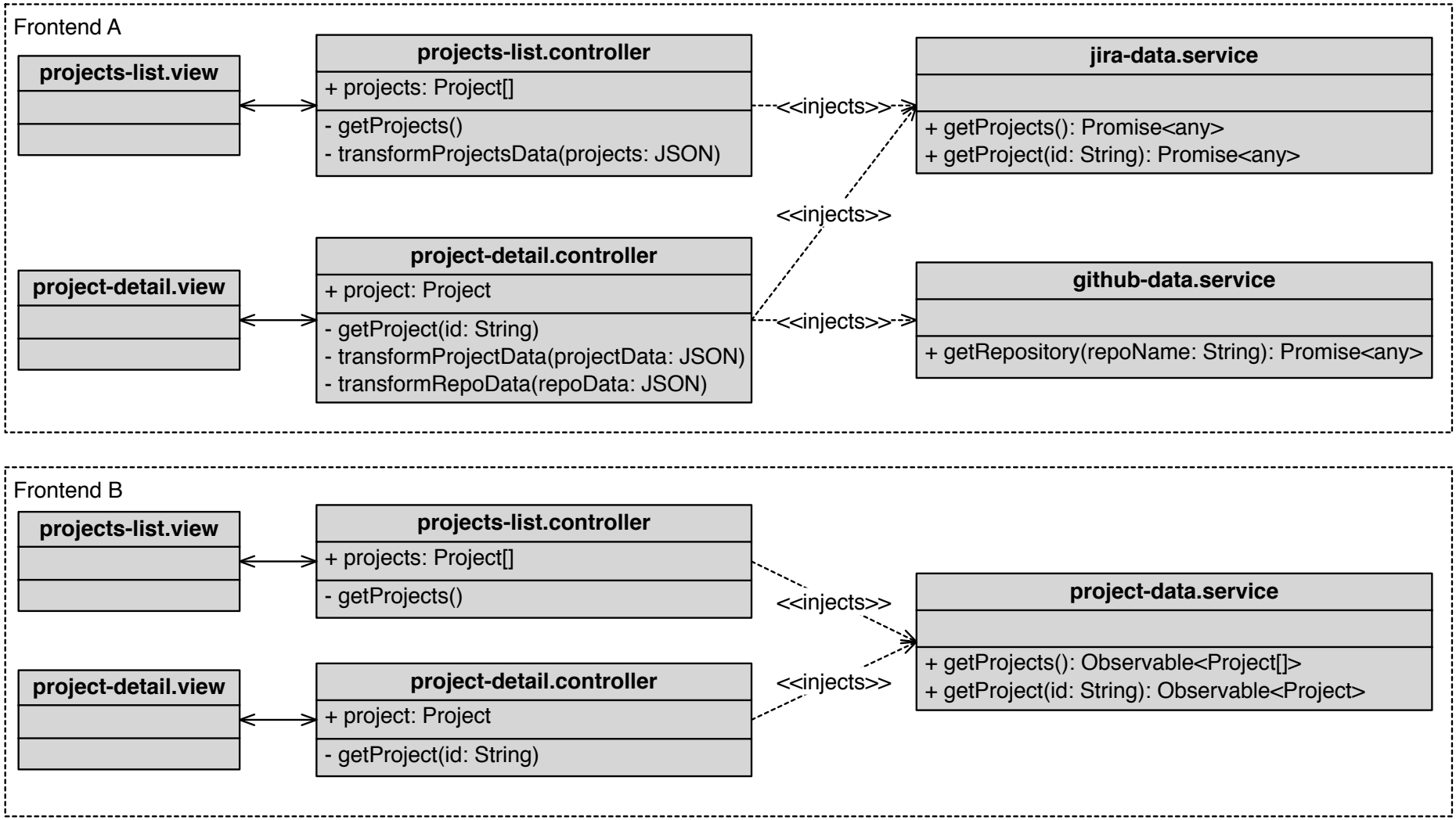
## Evaluation

- development of two example apps with tool
- comparison of two frontends (one with tool one without)



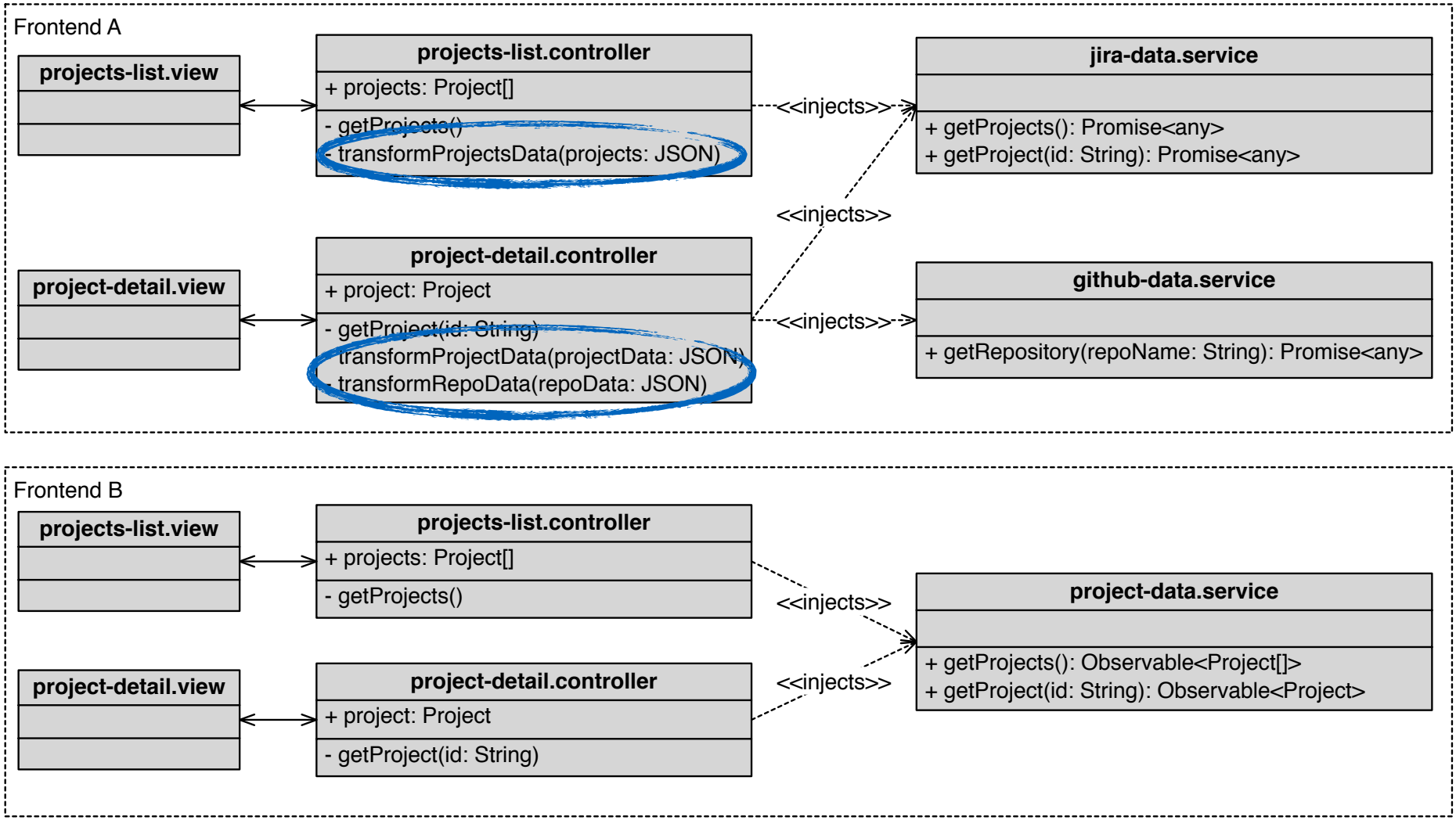
# Evaluation - Comparing Two Frontends



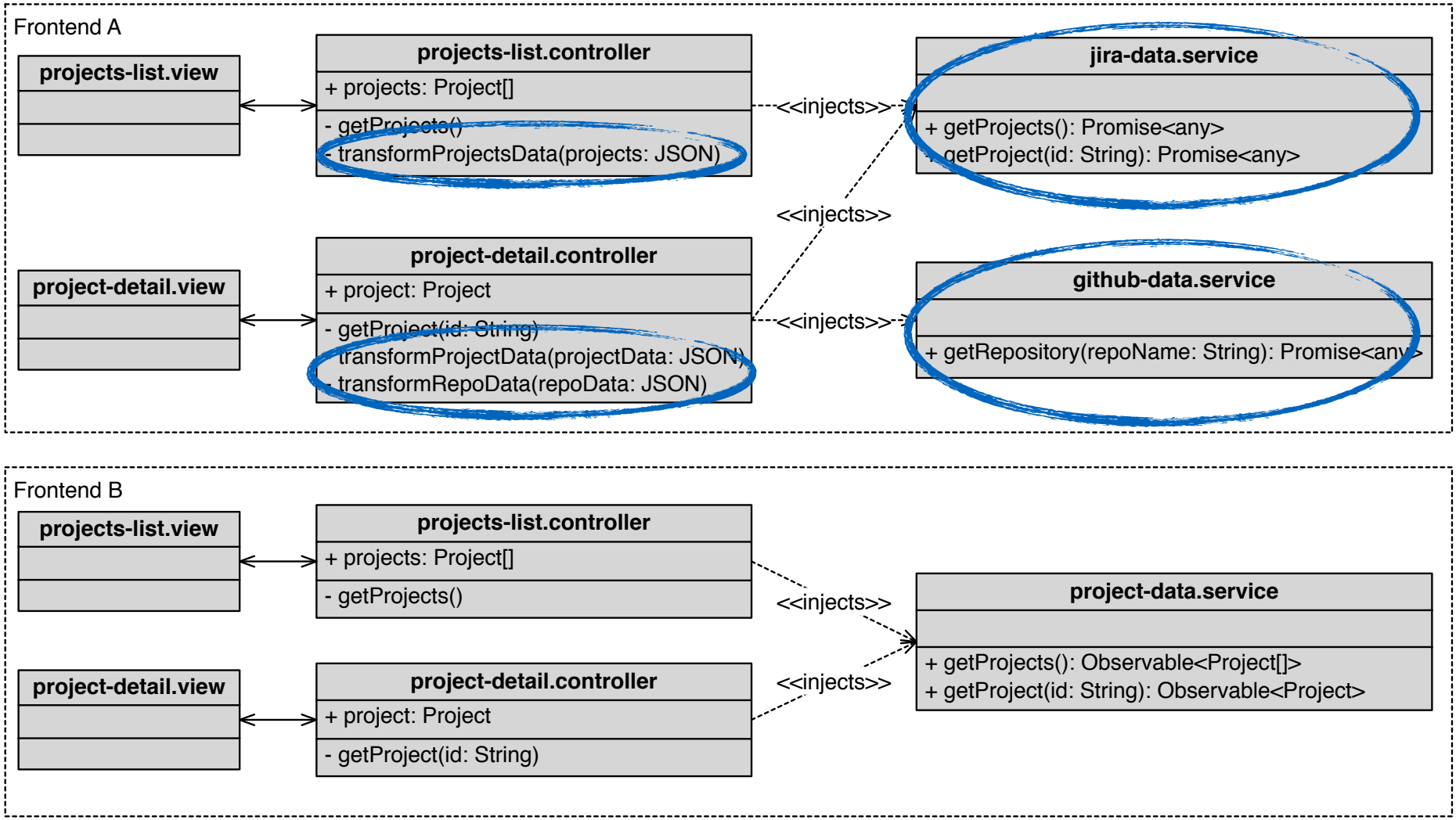


# Evaluation - Comparing Two Frontends



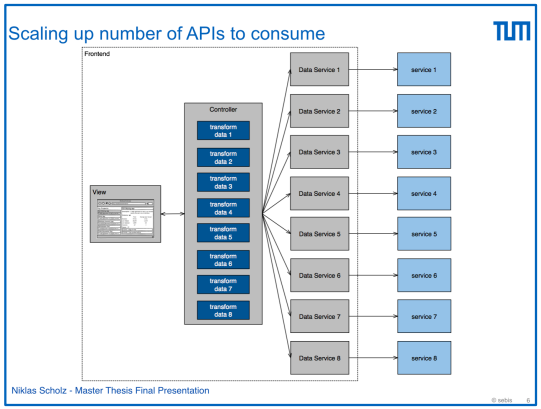
# Evaluation - Comparing Two Frontends





# Conclusions

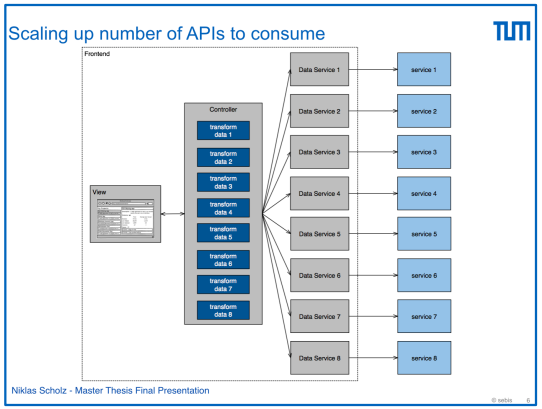



Consuming multiple APIs leads to complexity in the client

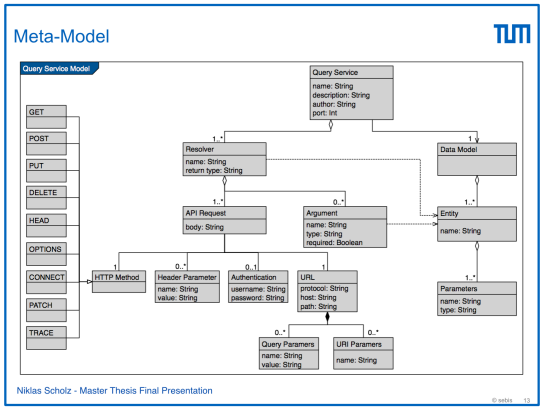


# Conclusions

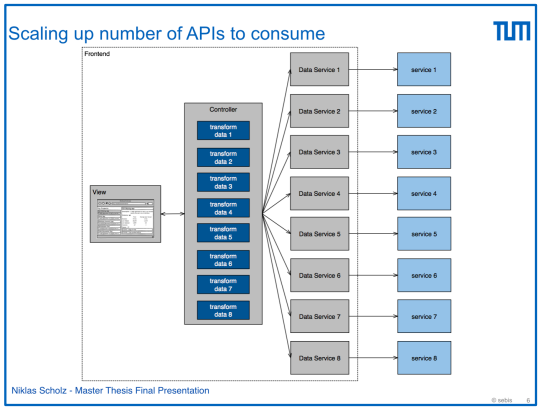



Consuming multiple APIs leads to complexity in the client

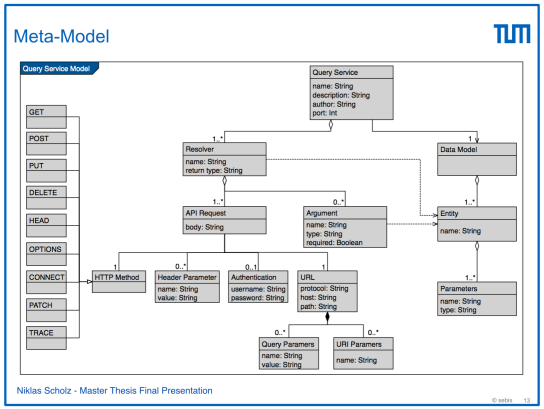



The consumption of RESTful services can be modelled

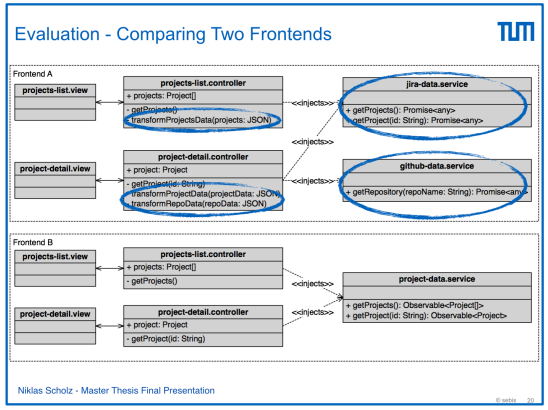
# Conclusions




Consuming multiple APIs leads to complexity in the client

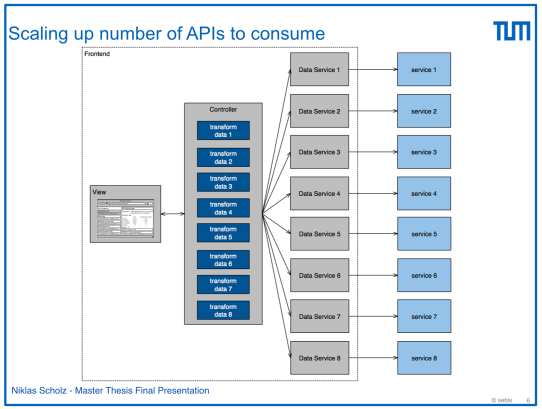



The consumption of RESTful services can be modelled

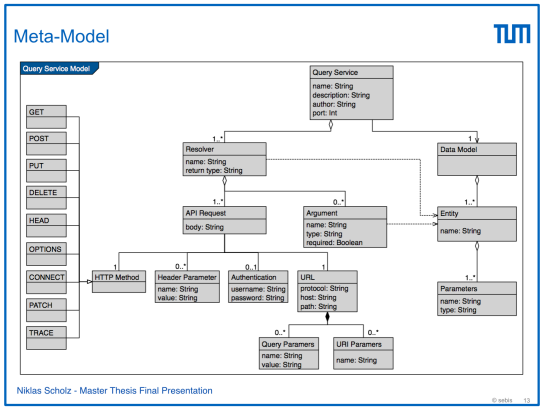



Approach reduces complexity in client

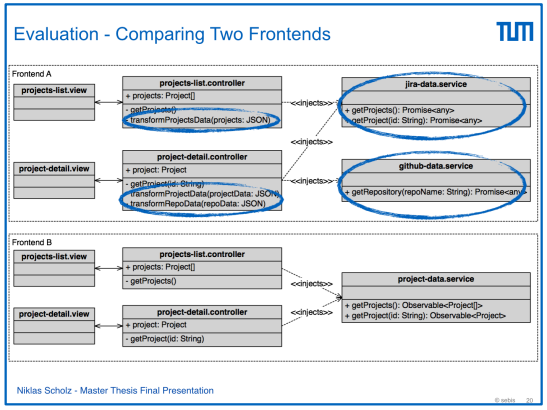
# Conclusions



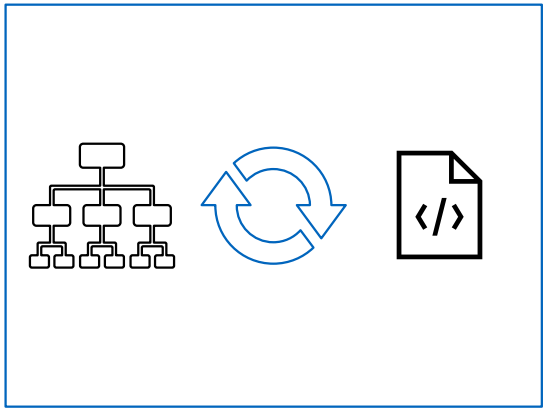

Consuming multiple APIs leads to complexity in the client




The consumption of RESTful services can be modelled

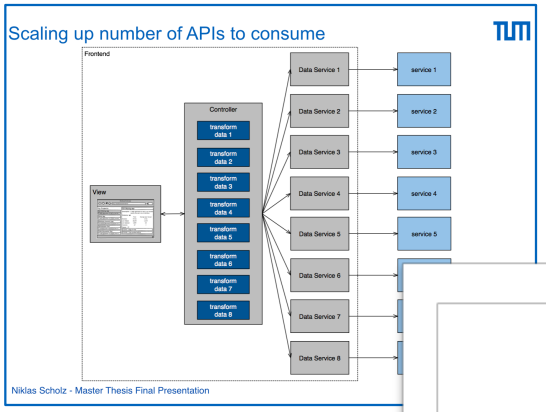



Approach reduces complexity in client

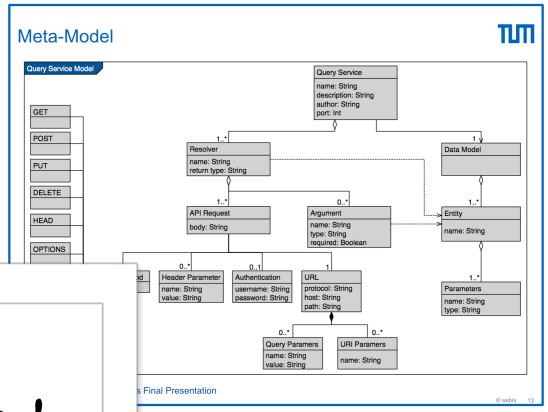


Limitation: Roundtrip Engineering

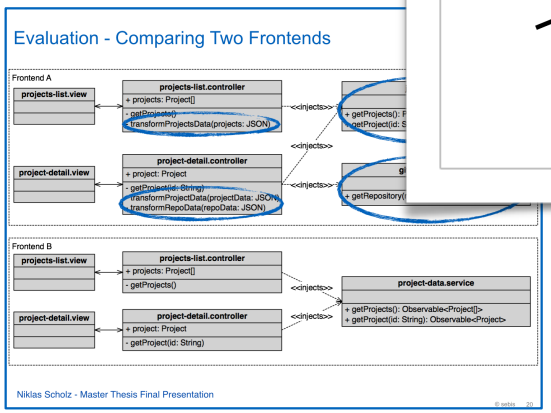
# Conclusions



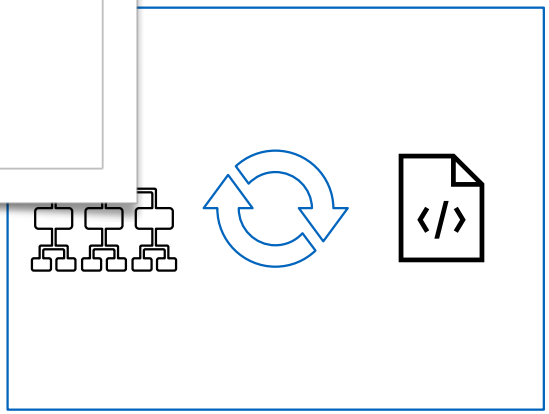

Consuming multiple AP
complexity in the




sumption of RESTful
es can be modelled


Approach reduces complexity in client



Limitation: Roundtrip Engineering



# References


[1] Markus Lanthaler and Christian Gütl. 'Hydra: A vocabulary for hypermedia-driven web APIs'. In: *CEUR Workshop Proceedings* 996 (2013). issn: 16130073.

[2] Davide Rossi. 'UML-based Model-Driven REST API Development'. In: *Proceedings of the 12th International Conference on Web Information Systems and Technologies, Vol 1 (WEBIST)* (2016), pp. 194–201. doi: 10.5220/0005906001940201.

[3] Hamza Ed-Douibi et al. 'EMF-REST: Generation of RESTful APIs from Models'. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (2015), pp. 1446–1453. doi: 10.1145/2851613.2851782.

[4] Florian Haupt et al. 'A model-driven approach for REST compliant services'. In: *Proceedings - 2014 IEEE International Conference on Web Services, ICWS 2014* (2014), pp. 129–136. doi: 10.1109/ICWS.2014.30.

[5] Rodrigo Bonifacio et al. 'NeoIDL: A Domain-Specific Language for Specifying REST Services'. In: *International Conference on Software Engineering and Knowledge Engineering* (2015), pp. 613–618. doi: 10.18293/SEKE2015-218.

# Backup - Advantages

**Lightweight Client**

- Reducing complexity on client-side
- GraphQL queries

**Focus on important steps**

- not carried away with implemen-tation details
- Focus on UI

**Advantages of MDSD**

- consistency
- reusability
- development speed
- manageability of complexity



# Backup - Limitations

**Roundtrip Engineering**

- Model not being stored
- Apply API changes manually

**Manual Code Refinement**

- Data transformation not modelled
- Developer has to add code manually

**Level of Abstraction**

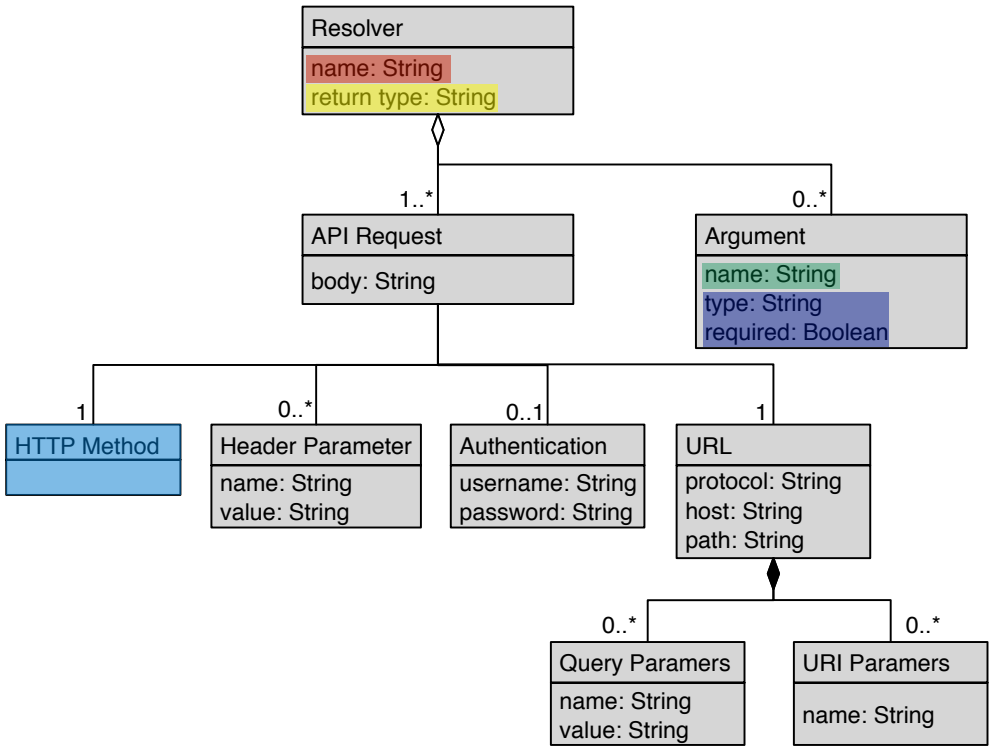
- only RESTful APIs
- model not entirely generic



# Backup - Model Transformation Process

Resolver Model




GraphQL Schema

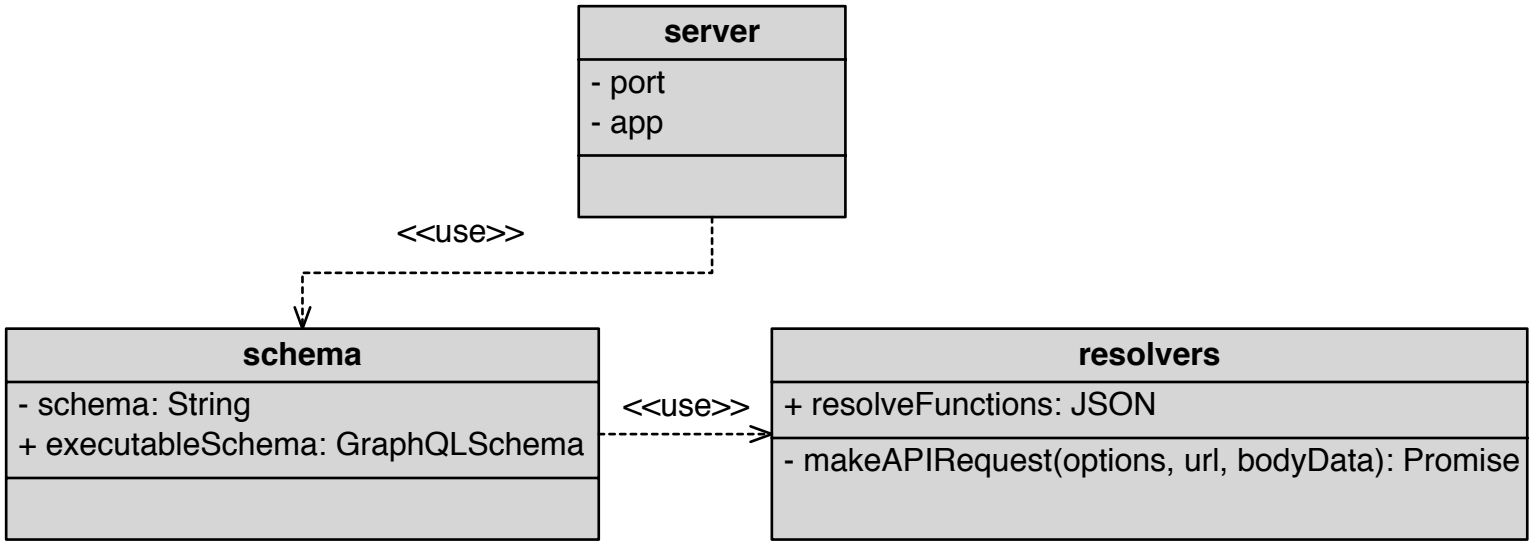
```
#the schema allows the following query:
type Query {
    projects: [Project]
    project(projectId: String!): Project
}

#the schema allows the following mutations:
type Mutation {
    deleteProject(projectId: String!): Project
}
```

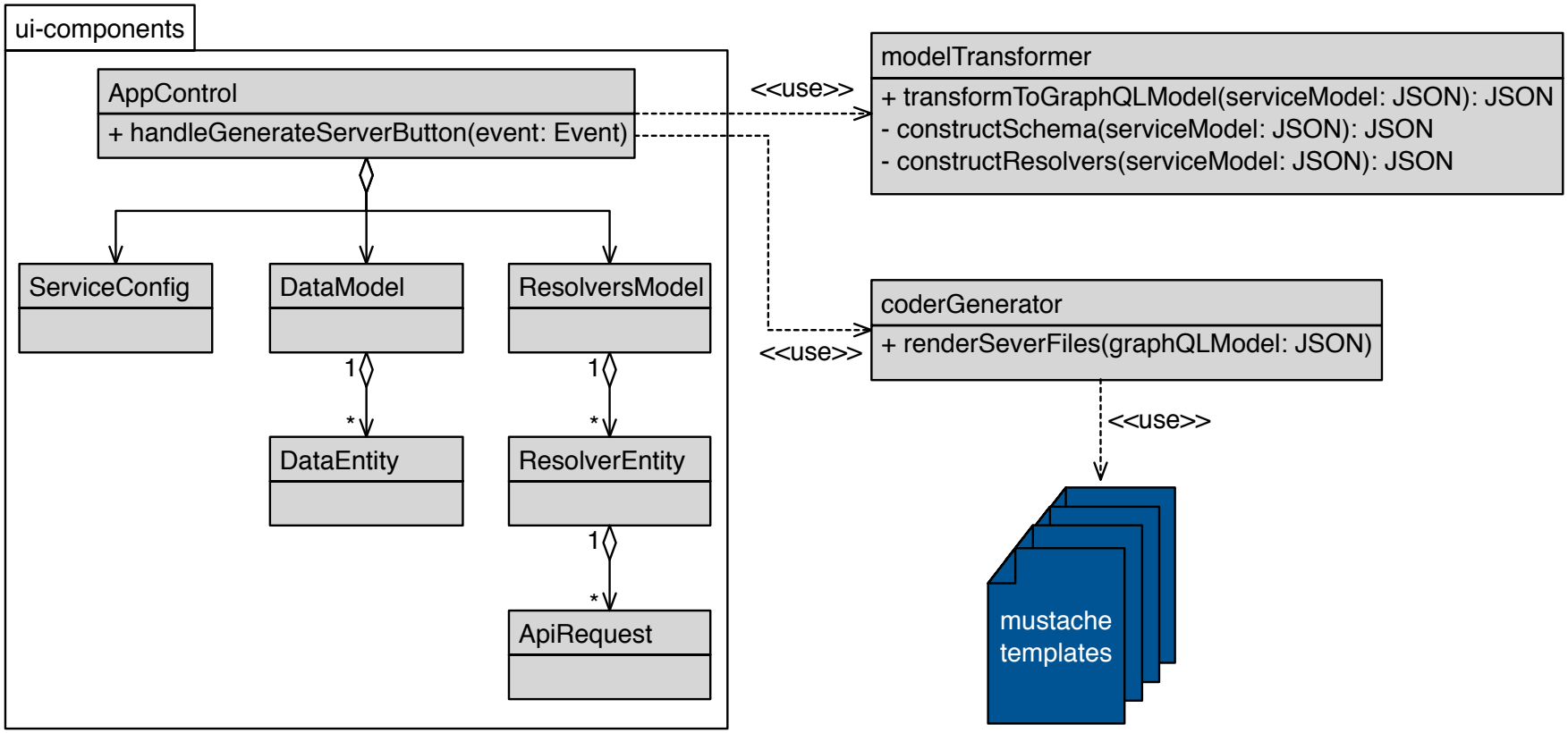


# Backup - Query Service Architecture



# Backup - Code Generation Tool Architecture





# Backup - GraphQL Schema Example

```
type Project {
    projectId: String
    name: String
    description: String
    duration: Float
    nrCommits: Int
}

#the schema allows the following query:
type Query {
    projects: [Project]
    project(projectId: String!): Project
}

#the schema allows the following mutations:
type Mutation {
    deleteProject(projectId: String!): Project
}
```

