TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Fredo Erxleben

# Design and Implementaion of a Web-Based RC3E User Frontend

Status Report // Dresden, April 12, 2018

# Contents of this Talk

Section 1

# Introduction

# The Core Issue

RC3E ( **R**econfigurable **C**ommon **C**loud **C**omputing **E**nvironment )

- System for distribution of FPGA ressources to remote users
- Actual ressources get abstracted as virtual FPGAs
- Controlled by a human administrator, who...
  - ...runs selected scripts on the host
  - ...manages users by hand
  - ...has to schedule ressources
  - ...does the programming of vFPGAs and relays the results

## Project Aim

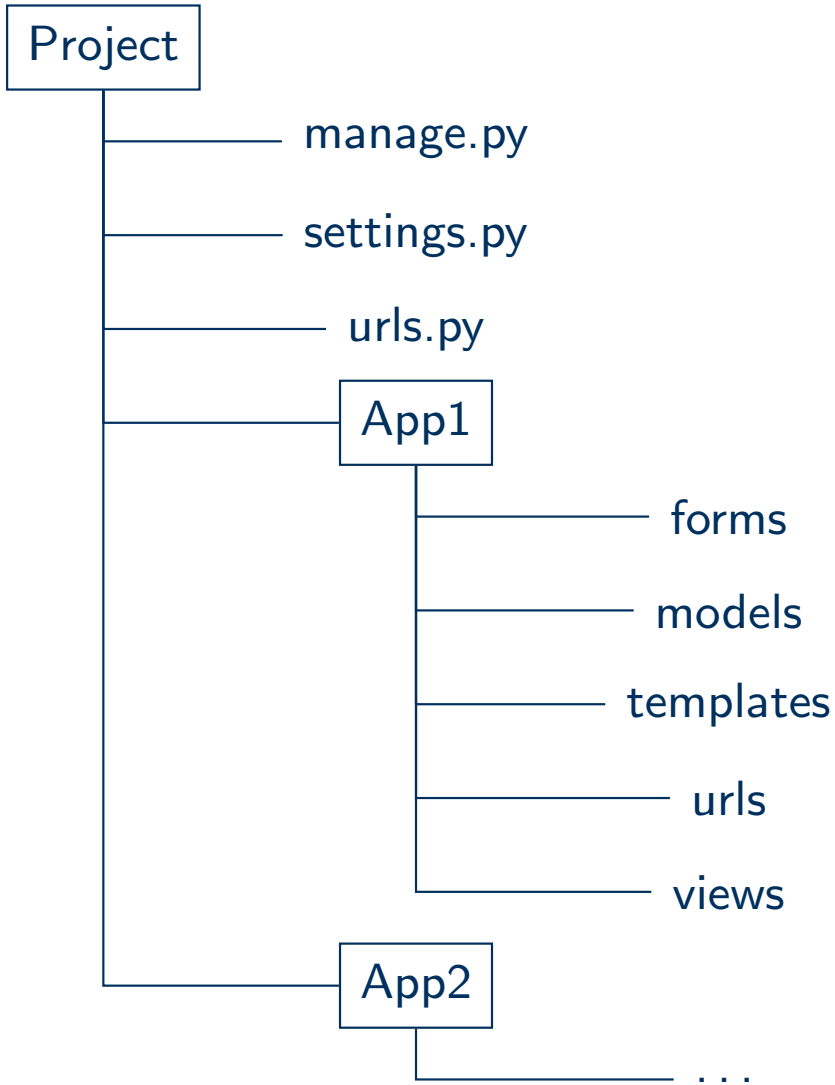
Create a web-frontend that can automatize these tasks.

# What is Django?

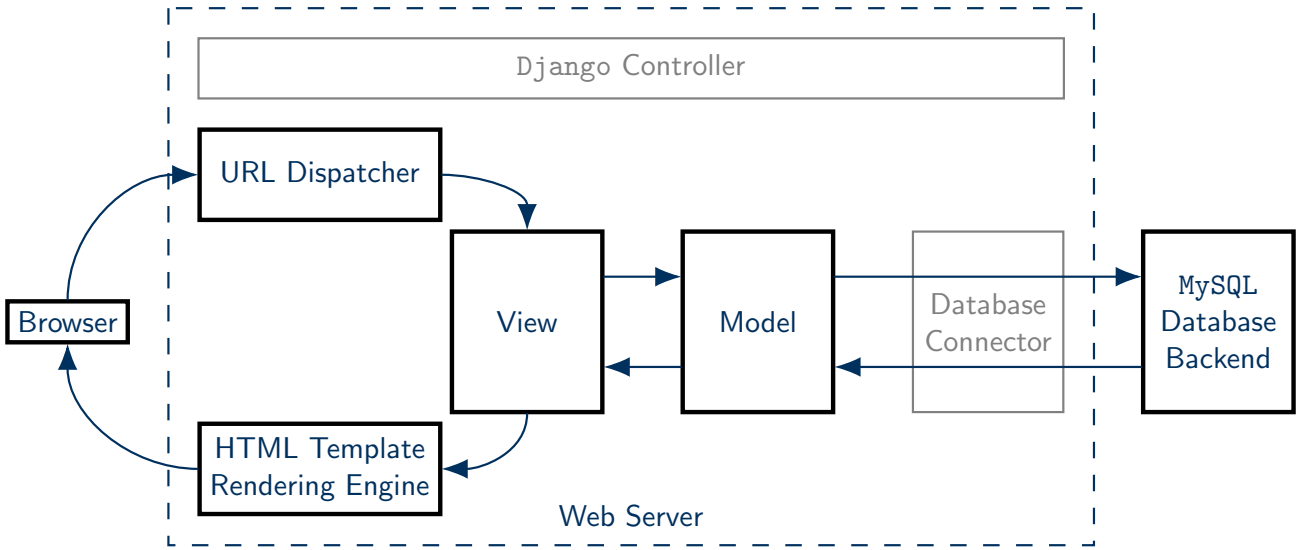
- Python-based framework for web applications
- Architecture is **M**odel **T**emplate **V**iew
  - Variation of MVC pattern
- Focus on reuseability, rapid prototyping and the *DRY*-principle[1]
- Huge amount of out-of-the-box features

[1] *DRY* = Don't repeat yourself

# Architecture of a `Django` Project



# Operation Principle of a Django-App



# What is REST?

**RE**presentational **S**tate **T**ransfer

- Architecture principle for distributed web applications
- Allow clients to access a textual representation of a ressource on the host
- Set of available operations on the ressources are well-defined by a simple interface

**Benefit**

Easy to parse for machine-to-macine communication.
Useful for interaction with the RC3E-server.

# Example for a RESTful API Call

## Request:

```
fredo in ~ >  http GET localhost:8000/rest_api/producers/
```

## Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
[More http header stuff]
[
    {
        "id": 1,
        "name": "Xilinx"
    },
    {
        "id": 2,
        "name": "Altera"
    },
    {
        "id": 3,
        "name": "Intel"
    }
]
```

Section 2

# Data Model

# Preface

The data model is the backbone of the project.

- Must be represented by a relational DB
- Fine granularity to improve changeability

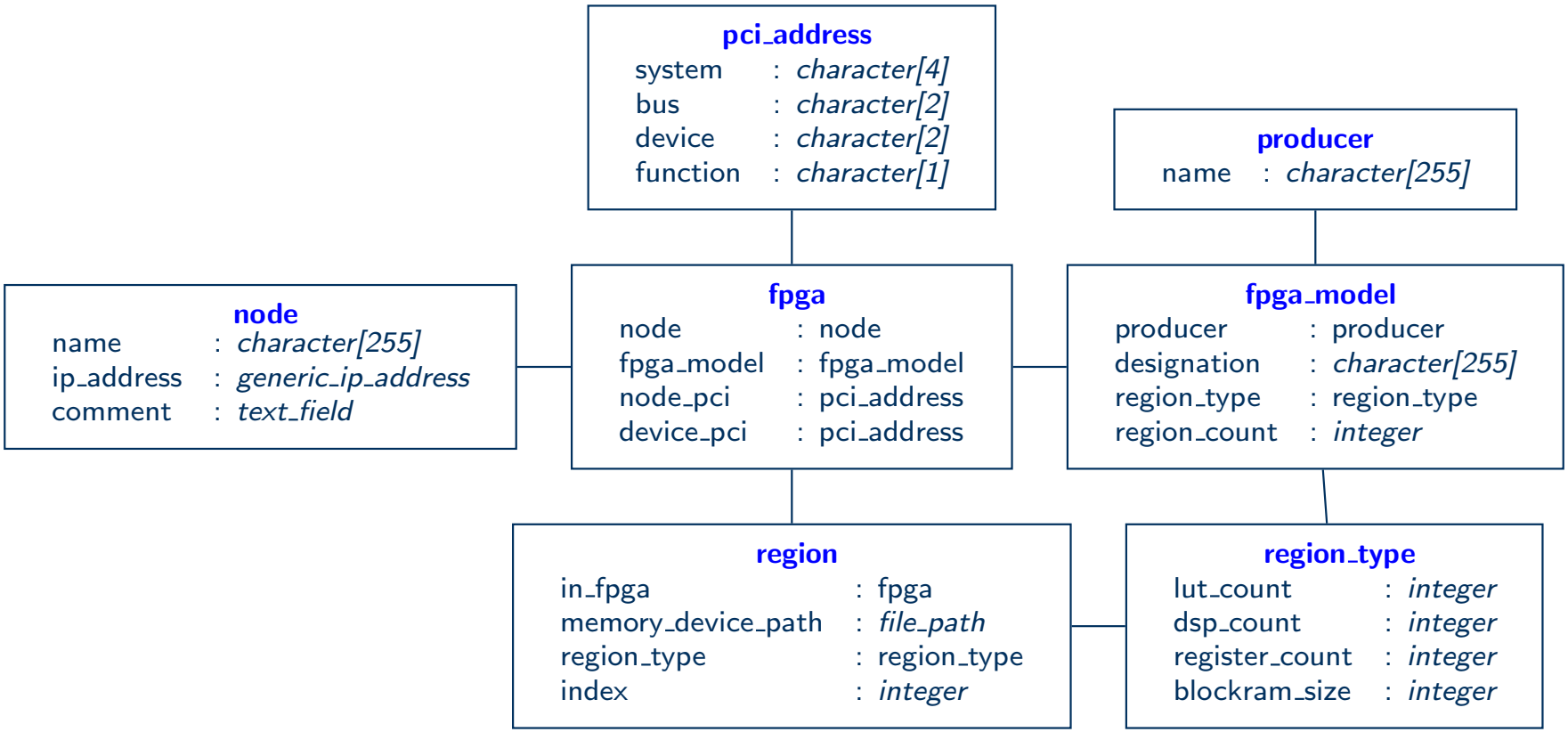
**Important!**

FPGAs are assumed to be homogenous.

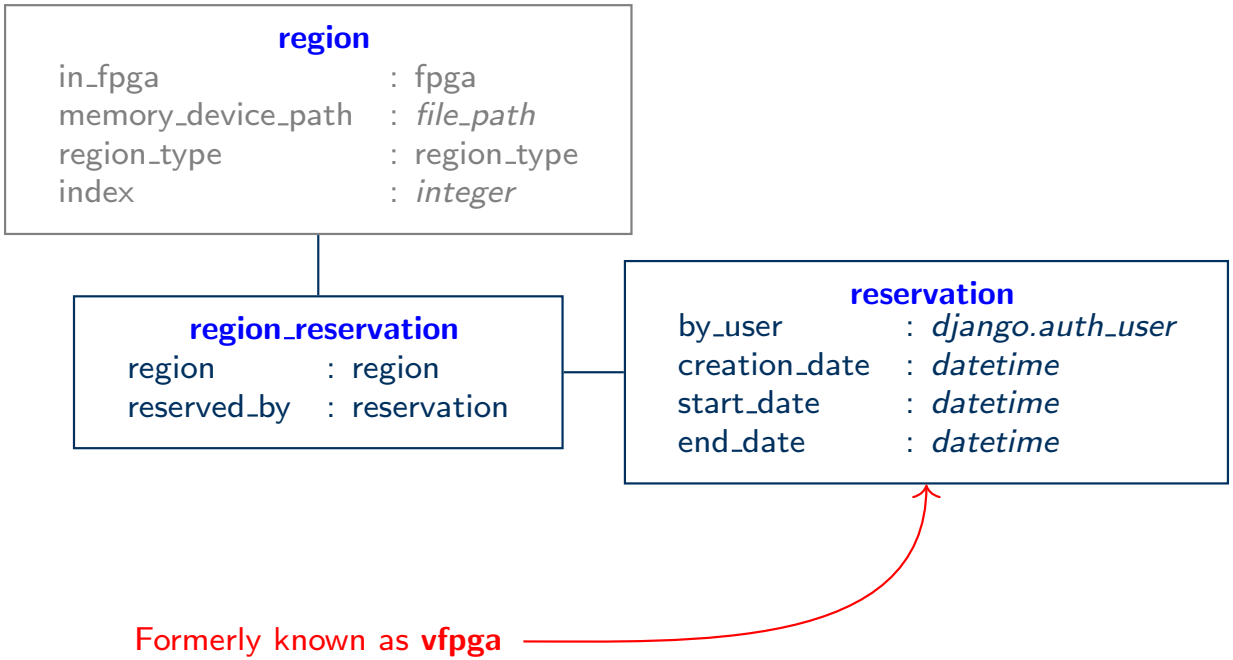
**Note**

Multiple development iterations introduced some technical debt.
But: It can be purged by refactoring cycles.

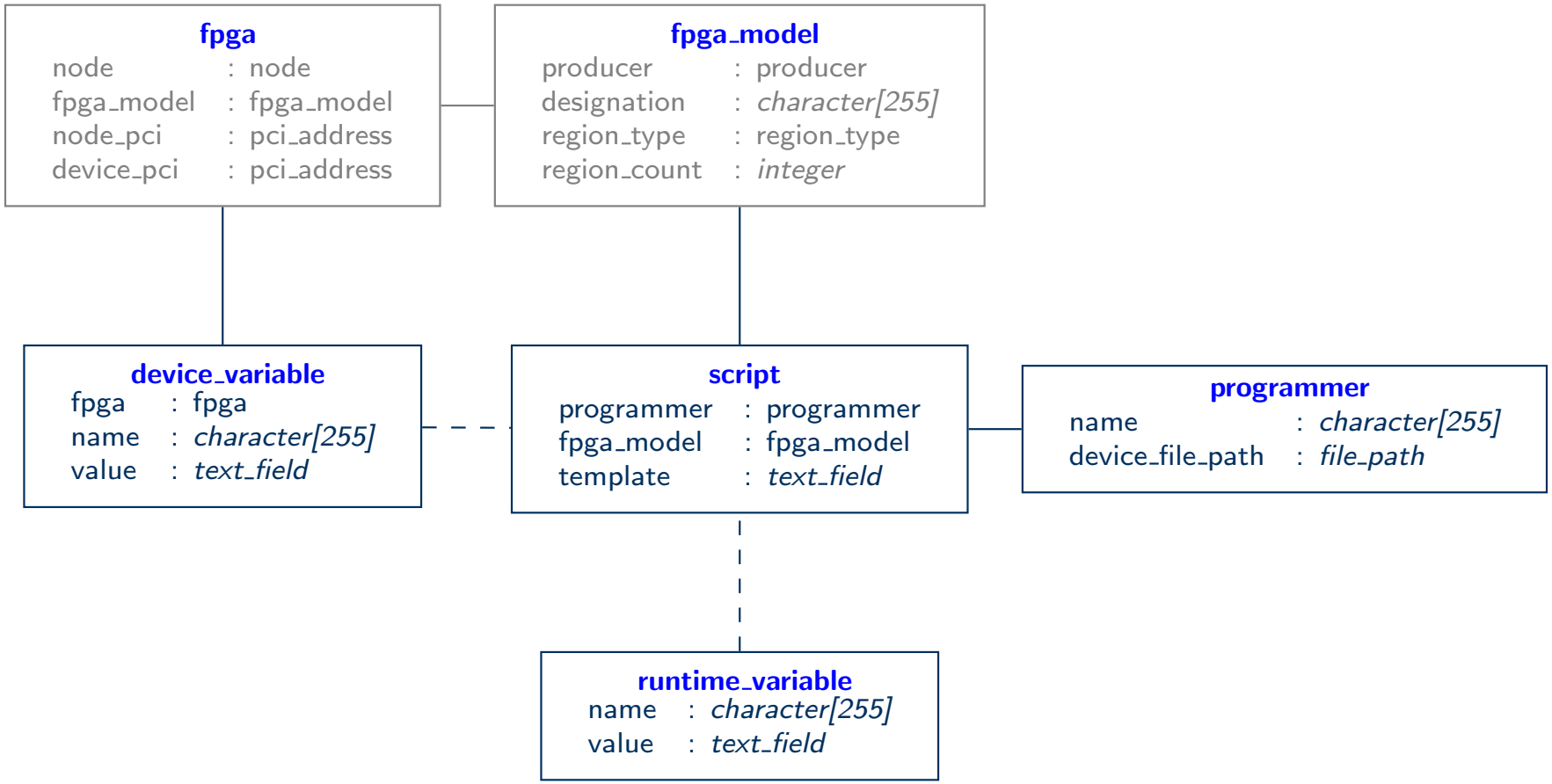
# Data Model Details I



# Data Model Details II



# Data Model Details III



Section 3

# Selected Issues

# Basic Principle of User Interaction I

Boil down all interaction to the following operations:

- *List* objects of a type
- *Show* object details
- *Create* objects
- *Delete* objects

**Note**

Allowing to modify existing objects has severe implications!
(What happens when you change the region type/ count of an FPGA?)

# Basic Principle of User Interaction II

**Issue**

URLs and internal project structure have to consistently represent this principle.

**Issue**

Look and feel of the HTML pages should be uniform.

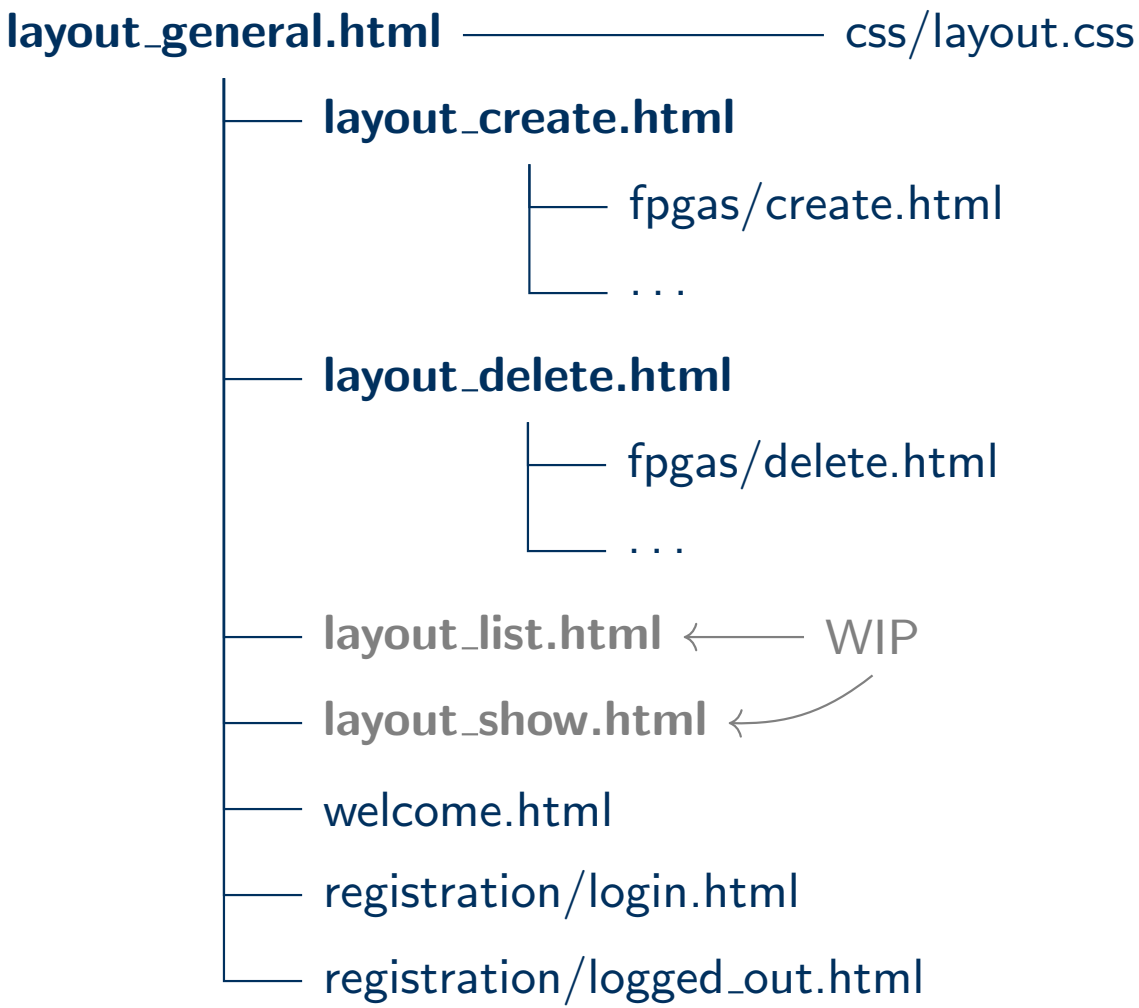
**Solution**

Use `Djangos` HTML template inheritance.

**Note**

Even special pages can profit from inheriting layout templates.

# HTML Template Structure



# Automating URL-Binding Generation

## Issue

Generating all the URL-bindings for the *List/Show/Create/Delete* pages by hand is tedious and error-prone.

## Solution

Write generator functions for the common case and only deal with the special cases by hand.

## Note

Generator functions also check if the required templates are present.

(PS: `https://www.xkcd.com/917/`)

# Supporting Multiple APIs I

**Issue**

Need to support a REST and a web API

- Both use the same protocol (HTTP/ HTTPS)
- HTTP request types *GET*, *POST* used by web API
- REST will use these as well

How to figure out what to do with an HTTP request?

# Supporting Multiple APIs II

## Idea

Give each API a different URL namespace.

Web-API:

```
GET    /web_api/producers/create/
POST   /web_api/producers/create/
```

REST-API:

```
GET    /rest_api/producers/
POST   /rest_api/producers/
GET    /rest_api/producers/1/
DELETE /rest_api/producers/1/
```

# Supporting Multiple APIs III

**Issue**

The REST-framework alters the `Django`-apps structure.

**Solution**

Split the project into one app per API.

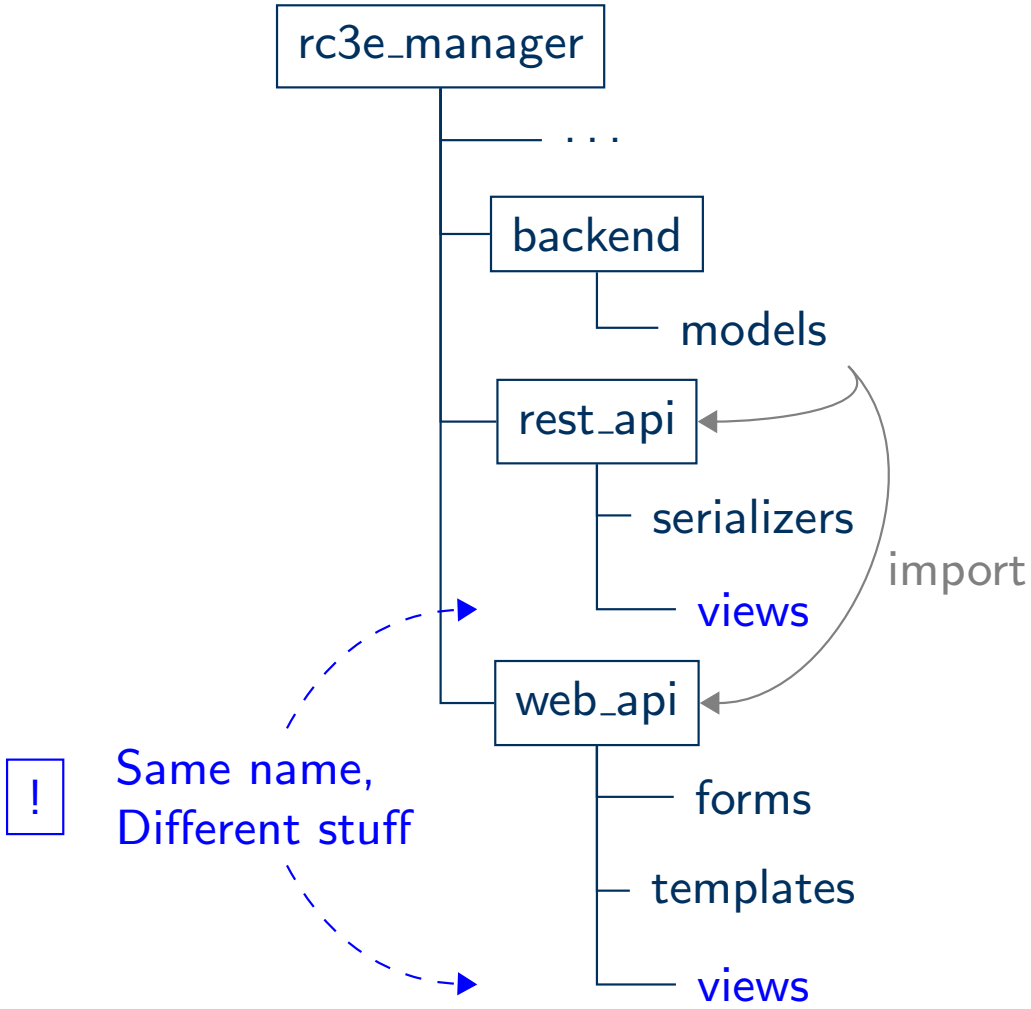
**Next Issue**

Both apps have to operate on the exact same model.

**Solution**

Split off the model into a separate app as well.

# Supporting Multiple APIs IV



# Scheduling of Reservations I

## User Input

- Region type
- Region count (consecutive)
- Desired start and end of reservation

# Scheduling of Reservations II

*View* generates and issues `SQL` query from user input.

**Database Action**

1. Filter for FPGAs with correct region type
2. Filter regions without reservation in desired timeframe
3. Sort remaining regions by index

Result type: `{FPGA: [Region]}`

# Scheduling of Reservations III

*View* then processes the query result.

## Pseudocode

```
foreach fpga:
|   foreach region:
|   |   if region is not consecutive:
|   |   |   reset markers // deal with fragmentation
|   |
|   |   mark region
|   |
|   |   if enough regions marked:
|   |   |   break all // Got enough reservation candidates
|
|   reset markers // Reservations across FPGAs are not allowed

reserve candidates // If there are any
// Inform User
```

# Scheduling of Reservations IV

## Notes

- *First fit* scheduling does not use ressources equally
- Fragmentation may become an issue in real-life use
- Reservations are not supposed to be deleted (accounting) which may slow down the DB after many reservation cycles

It works for the scale of RC3E but has optimization potential.

Section 4

# Results and Future Work

# Summary of Current State

- Data model has been developed
- Technology research is completed
- Model is implemented
- Web-API is implemented
- Currently known issues have been solved

# Future Work

Immediate tasks:

- Finish transition to new HTML template structure
- Finish implementation of REST-API

Whishlist:

- Subsystem for uploading designs, running the FPGA programming scripts and operation feedback
- Subsystem for statistics and accounting
- Logging RC3E interactions to DB

**Ultimative Goal**

Connect the frontend with the live RC3E-system.

Section 5

# Appendix

# Further Reading I

## RC3E

*RC3E: Reconfigurable Accelerators in Data Centres and Their Provision by Adapted Service Models*, Oliver Knodel, Patrick Lehmann, Rainer G. Spallek, 2016
`https://ieeexplore.ieee.org/abstract/document/7820030/`

*Rekonfigurierbare Hardwarekomponenten im Kontext von Cloud-Architekturen*, Dissertation of Oliver Knodel, 2018
Publication pending

# Further Reading II

## REST

*Architectural Styles and the Design of Network-based Software Architectures*, Dissertation of Roy Fielding, 2000
`https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`

# Further Reading III

## Django

*Official Documentation*

https://docs.djangoproject.com/en/2.0/

*Django-REST*

http://www.django-rest-framework.org/

# Last Slide

Thank you for your attention.
Questions?


| | |
|---|---|
| **Office:** | TUD, APB 1095 |
| **Phone:** | TUD-38456 |
| **E-Mail:** | fredo.erxleben@tu-dresden.de |
| **Project Git:** | `http://github.com/VLSI-EDA/rc3e-manager` |


PS: We are looking for students to continue working on this project.

Bachelor thesis, anyone? ;)