Lessons learned from using Django as GraphQL backend

The good, the bad, the ugly

Thomas Aglassinger

2024-11-05

Lessons learned from using Django as GraphQL backend

The good, the ugly, the bad

Thomas Aglassinger, 2024-11-05

About me

Thomas Aglassinger

- MSc in information processing science
- 20+ IT experience in various sectors
- Founder Siisurit https://siisurit.com/
- Part-time employee https://www.providens.at/
- Casual freelancer https://aglassinger.at/
- Casual open source developer https://roskakori.at/

Agenda

- The good: Show it. 🥰
- The ugly: Show it and give suggestion on how to deal with it. 😤
- The bad: Show it and despair. 😭
- Summary and musings about the future 🤔

The good 🥰

GraphQL schema

- GraphQL has built in schema to support
  - Not an afterthought like REST with OpenAPI etc
  - graphene: Generate schema from Python code
- Explorers help to build queries
- IDEs validate query code while typing

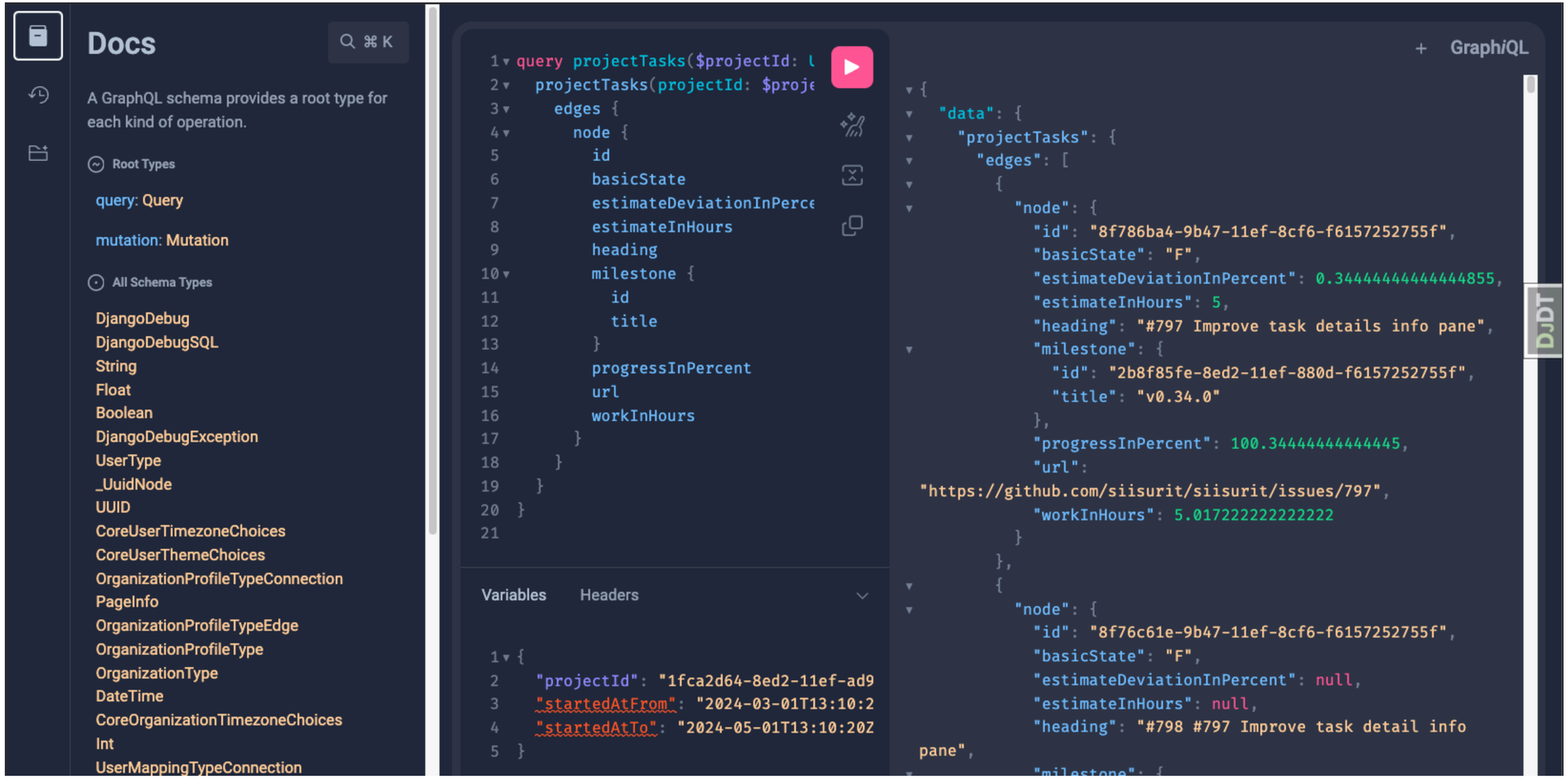
GraphQL schema explorers

Example: GitLab

- https://gitlab.com/-/graphql-explorer

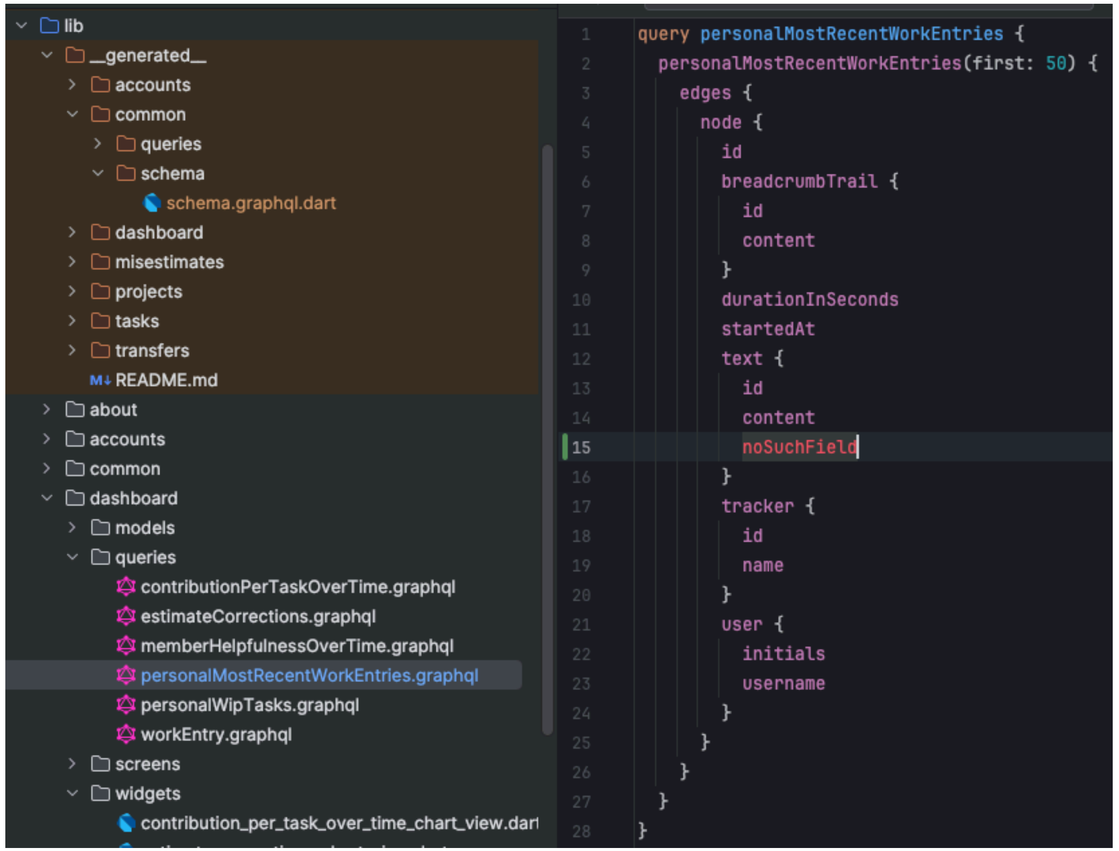
GraphQL schema explorers

Graphene: includes graphiql as Django view



GraphQL schema tool integration

- Schema can also be used to generate code to access data from other languages, for example: Dart.
- Android Studio can use schema to validate queries on the fly.



GraphQL: Query only the fields you need

- Example:
  - GitLab issue has many fields.
  - Typically you only need a few.
  - No need to transfer the entire Issue object like it is common with REST APIs.

```
{
  project(fullPath: "gitlab-org/graphql-sandbox") {
    name
    issue(iid: "2") {
      title
    }
  }
}
```

GraphQL: More than JSON types

- (Mostly) standardized types for
  - date time
  - int
  - UUID
  - ...

GraphQL: Paging via connections

First page

```
{
  project(fullPath: "gitlab-org/graphql-sandbox") {
    issues(first: 100) {
      pageInfo {
        endCursor
        hasNextPage
      }
      nodes {
        title
      }
    }
  }
}
```


```
{
  "data": {
    "project": {
      "issues": {
        "pageInfo": {
          "endCursor": "eyJjcmVhd...",
          "hasNextPage": true
        },
        "nodes": [
          {
            "title": "More issues for everyone!"
          },
          {
            "title": "Have some tea"
          },
          {
            "title": "Spider issue"
          }
        ]
      }
    }
  }
}
```

GraphQL: Paging via connections

Next page

```
{
  project(fullPath: "gitlab-org/graphql-sandbox") {
    issues(
      first: 3,
      after: "eyJjcmVhd..."
    ) {
      pageInfo {
        endCursor
        hasNextPage
      }
      nodes {
        title
      }
    }
  }
}
```


```
{
  "data": {
    "project": {
      "issues": {
        "pageInfo": {
          "endCursor": "eyJjcmVhd...",
          "hasNextPage": true
        },
        "nodes": [
          {
            "title": "..."
          },
          {
            "title": "Musical issue"
          },
          {
            "title": "Incendio!"
          }
        ]
      }
    }
  }
}
```

The ugly 

GraphQL error handling 1/2

- Everything is HTTP status 400 (bad request)
- If you attempt to access something you should not, exclude it from response (lists) / have null-response (single item)
- Error details are stored in standardized "errors" object
- See also: https://ariadnegraphql.org/docs/error-messaging

```
{
    "errors": [
        {
            "message": "Cannot do something.",
            "locations": [
                {
                    "line": 1,
                    "column": 21
                }
            ]
        }
    ]
}
```

GraphQL error handling 2/2

- "The errors key is, by design, supposed to relay errors to **other developers working with the API**. Messages present under this key are technical in nature and shouldn't be displayed to your end users."
- "Instead, you should **define custom fields** that your queries and mutations will include in result sets, to relay eventual errors and problems to clients, like this: ..."
- See also: https://ariadnegraphql.org/docs/error-messaging
- Yes, but what about queries? 🤔

```
type Mutation {
  login(
    username: String!,
    password: String!
  ): LoginResult!
}

type LoginResult {
  error: String
  user: User
}
```

Authentication with JWT

JSON web tokens

- Popular with many GraphQL APIs and documentations
- Messy and complicated
  - Permanent token
  - Temporary token that expires after a few minutes
  - Regularly request new temporary token using permanent token

Authentication with JWT

Authentication with JWT

JSON web tokens

- See also: https://blog.ploetzli.ch/2024/should-i-use-jwt-for-authentication/
- "JWT as authentication tokens are constructed for Google/Facebook scale environments, and absolutely no one who is not Google/Facebook needs to put up with the ensuing tradeoffs."

Authentication with JWT

JSON web tokens

- If you must:
  - JSONWebTokenMiddleware: https://github.com/flavors/django-graphql-jwt/blob/main/graphql_jwt/middleware.py
  - Django GraphQL JWT: https://django-graphql-jwt.domake.io/

Why not just use HTTP bearer token?

- Indeed. Just do that.
- All necessary components are already part of Django REST framework
  - Beef up with Knox: https://jazzband.github.io/django-rest-knox/
- Requires a little fiddling to make work with GraphQL
  - See: "Token Authentication and Authorization with GraphQL and Django", https://danielwelch.github.io/django-graphql-token-auth.html

Addendum: JWT can also store permissions

- Pros
  - No permissions needed in the database
  - Can be combined with single sign on, e.g. AD contains permission details and generated JWT from that
- Cons
  - increased network traffic
  - permission changes only become active after the token expires

Thanks Florian, for pointing this out! 👍

GraphQL IDs

- Every GraphQL node must have a unique ID
  - ...that is unique even compared to other types
- Standard solution: Combine type name and int ID from DB, then do base64
- Result: Constant encode/decode to get ID actually needed in a specific context
- Possible alternative approach: Use UUID
  - Consider using UUIDv7 = time + random
- Graphene: Not sure how to do this properly, using Monkey Patch for now 🙈

Security

- GraphQL queries get mapped to Django queries
- Django queries technically can access all data.
- Possibly not every user should see all data.
- You need to limit the Django query for every GraphQL query yourself.
- Graphene: For types, you can specify which Django objects are allowed.
- Most of the responsibility to get this right is with you.

The bad 😭

Graphene error handling

One AssertionError to rule them all

- Graphene uses AssertionError to communicate that a query failed for some reason. 🤪
- If you use assert internally without a specific message, and it fails, you get an "unknown error" with a spurious location. 😭

```
def resolve_something(root, info):
    x = ...
    assert x > 0  # Fails!
    ...
```

```
errors=[{
  'message': 'An unknown error occurred.',
  'locations': [{'line': 2, 'column': 3}],
  'path': ['something']
}]
```

Graphene handling of N+1 queries

...or the lack of thereof

- Strawberry has an optimizer extension that automatically deals with N+1 queries
- Graphene doesn't
  - Instead, your "resolvers" that need to be manually implemented for every object type that can cause N+1 queries
  - Documentation is lacking good, actionable examples
  - Mostly blogs, some of them behind Medium paywall
- Me: Kind-of-solved by mostly querying IDs only and caching entire objects

N+1 queries

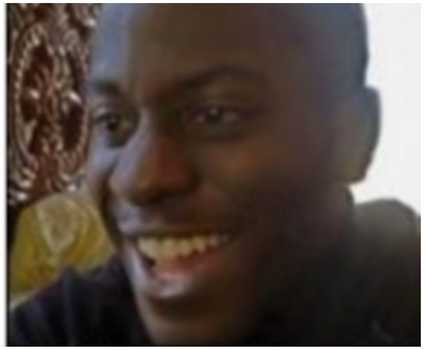
- Nested queries:
  - 1 SQL to obtain 1st level
  - N SQL to obtain each record of next level
- Examples:
  - Project has 1 organization
  - Issue has N labels
  - Issue has 1 or N assignee(s)

```
query projectsById($ids: [UUID]) {
  projectsById(ids: $ids) {
    id
    code
    name
    organization {
      id
      name
    }
  }
}
```

GraphQL filtering

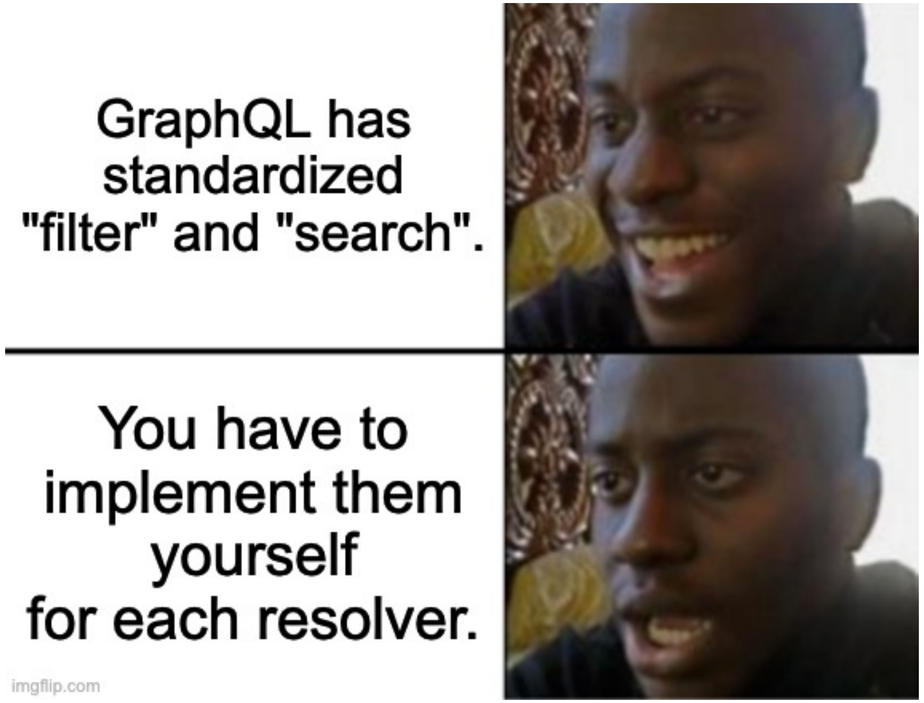
Think SQL-Where

GraphQL has
standardized
"filter" and "search".



GraphQL filtering

Think SQL-Where



- See for example: https://www.howtographql.com/graphql-python/7-filtering/
- Maybe I'm missing something? 🤔

Summary

Summary

About GraphQL

- GraphQL solves some of the headaches of REST APIs
- GraphQL adds intrinsic complexity.
- GraphQL errors can be hard to track.
  Sometimes requires debugging of library code, which is unusual with REST.
- GraphQL is dumb unless you help it (N+1 queries)
- GraphQL IDs cam be annoying, consider using UUID.

Summary
GraphQL and Django

- Avoid JWT if possible, use DRF tokens instead.
- Graphene has good parts and shaky parts.
- Strawberry might be better suited.

Summary

GraphQL and Django

- Avoid JWT if possible, use DRF tokens instead.
- Graphene has good parts and shaky parts.
- Strawberry might be better suited.

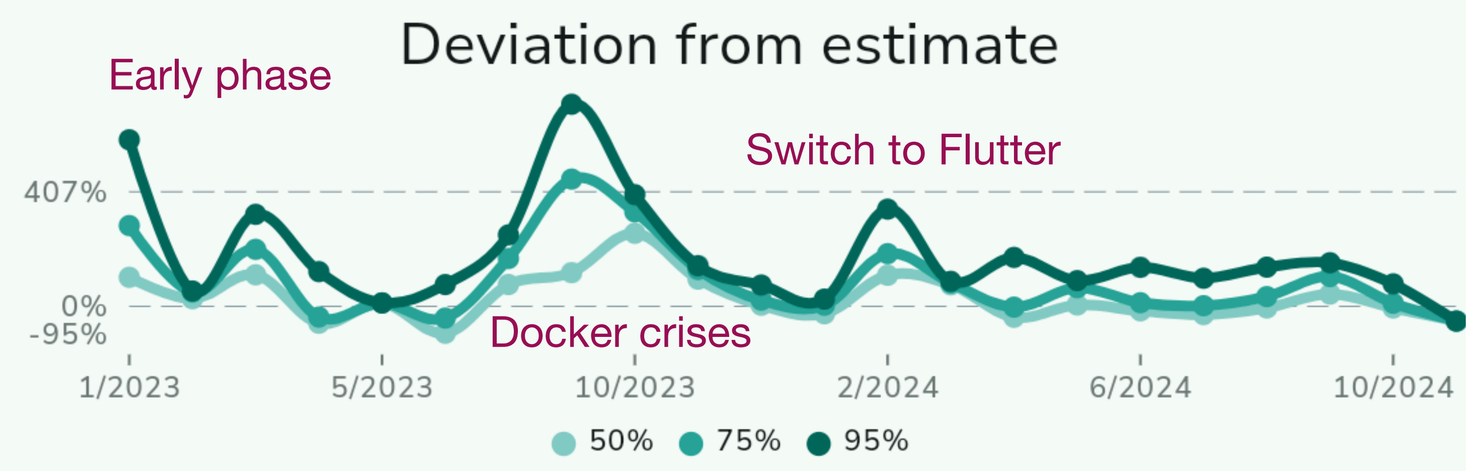
Ask for early access: hello@siisurit.com

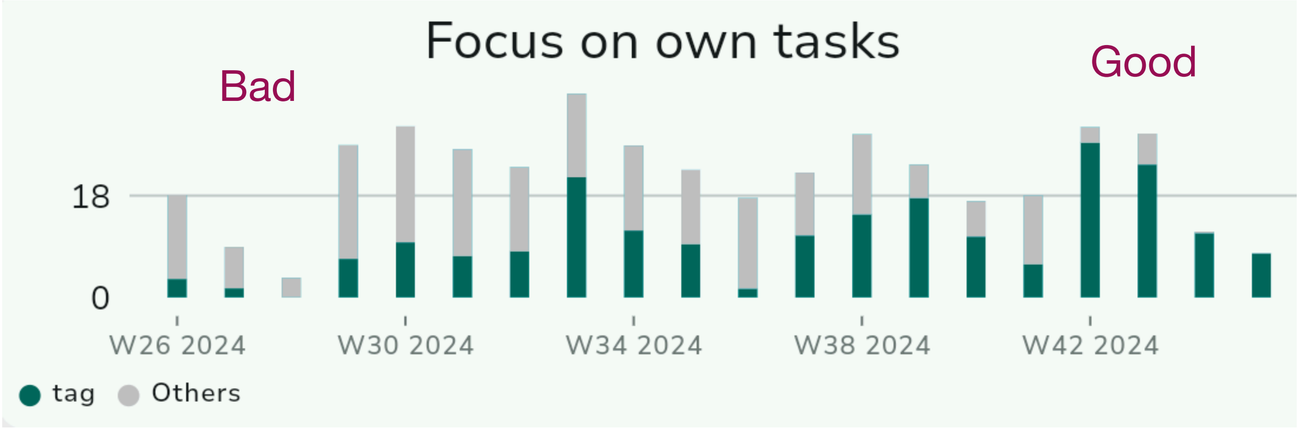
https://siisurit.com/

Appendix

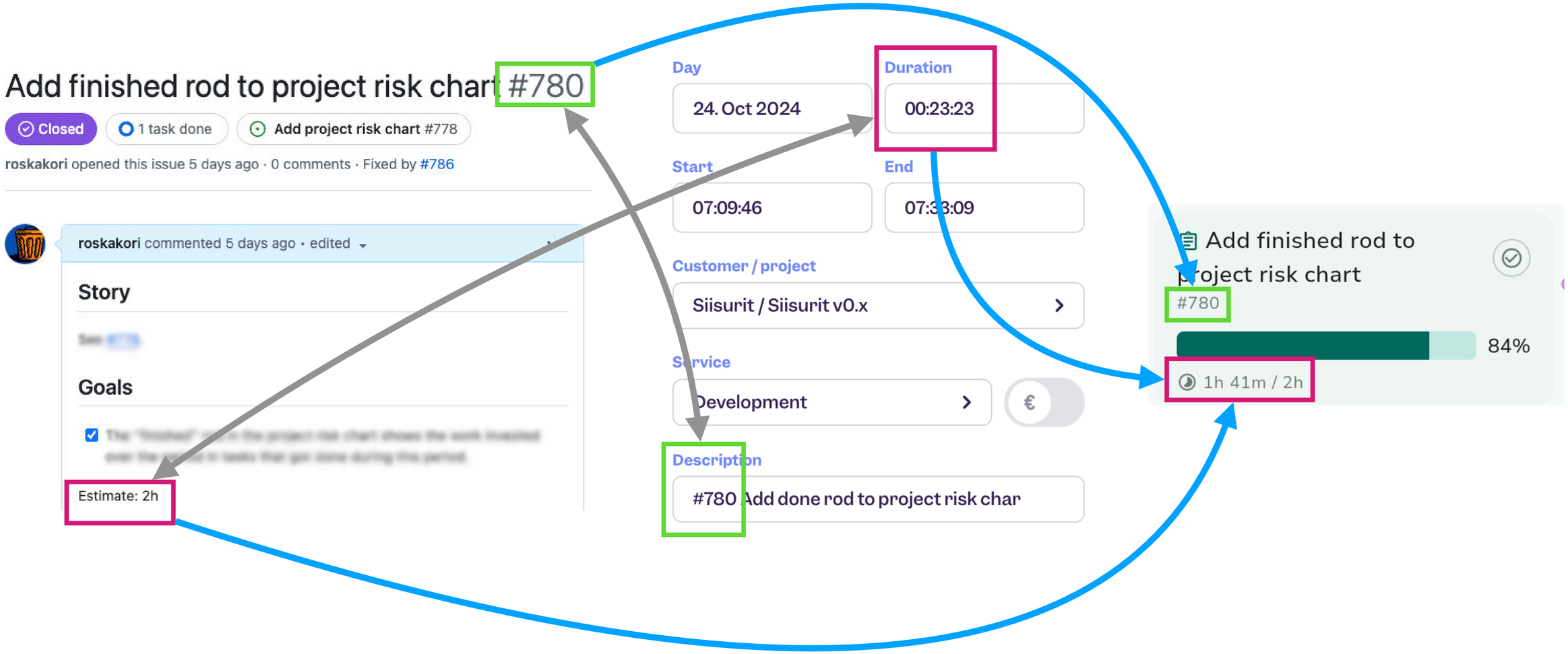
The following slides a taken from a talk given at the GrazReq#3 requirements engineering meetup. For the full set, see https://github.com/roskakori/talks/blob/master/graz_req/Transparently%20dealing%20with%20shifting%20requirements.pdf

Example Flutter charts built with GraphQL





Continuous task and time tracking



#778 Add project risk chart

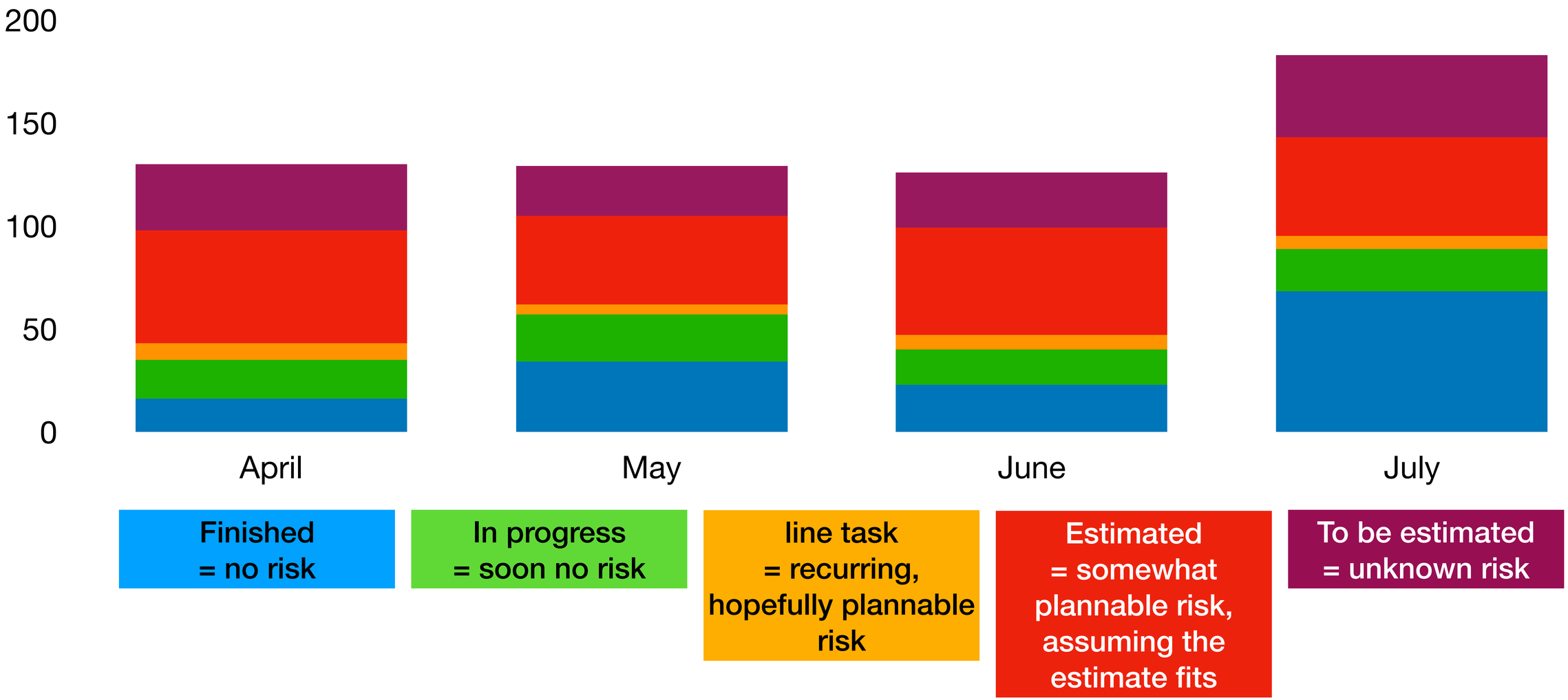
User story

"As project admin, I want to know how much risk concerning work [in hours] is still left in the project,

so that I can decide if mitigation is needed."

Recap: #778 Add project risk chart

UI draft for project risk development chart



Identify risky parts and split them into tasks

Part	What needs to be done?	Risks	Estimate / Action
Finished rod	Database query for chart data		2h
In progress rod	Database query for chart data		2h
Line task rod	How to detect line task?	No clear requirements	Omit for now, need to specify
Estimated rod	Database query for estimate - progress; needs some logic	Database query possible, or loop needed?	5h / Expect to fiddle
To be estimated rod	Compute with heuristics. Which? Use median for now.	Median is not standard SQL. How to compute in ORM?	5h / Research median
Add chart to UI	Common frontend things, data model for chart	How to represent fixed ID not from database?	4h / Explore ID options

How did the estimates turn out?

low risk

Add project risk chart
#778
66%
39m / 1h

Add dummy project risk chart
#779
98%
3h 54m / 4h

Add finished rod to project risk chart
#780
84%
1h 41m / 2h

Add in-progress rod to project risk chart
#781
101%
2h / 2h

medium risk

Add to-estimate rod to project risk chart
#783
76%
3h 47m / 5h

Add estimated rod to project risk chart
#782
56%
2h 49m / 5h

high risk

Add "to estimate" chart to project details screen
#793
11m

Add line tasks rod to risk development chart
#796
6m

Hide to-estimate rod in risk development chart
#792
55%
16m / 30m

Added tasks