GraphQL integration with Flutter

Options explored and lessons learned

Thomas Aglassinger

2024-11-20

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

- 🎓 GraphQL primer and backend exploration
- 🔀 GraphQL - The good the bad the ugly
- 💡 How I use GraphQL - in action
- 🤔 Summary and musings about the future

What is GraphQL?

What is GraphQL?

- A data query and manipulation language for APIs
- Open source, developed by Facebook
- Addresses some issues REST APIs tend to have.
- Somewhat "SQL for APIs", but has graphs instead of tables.
- Great introduction and tutorial: https://graphql.org/learn/

GraphQL - 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

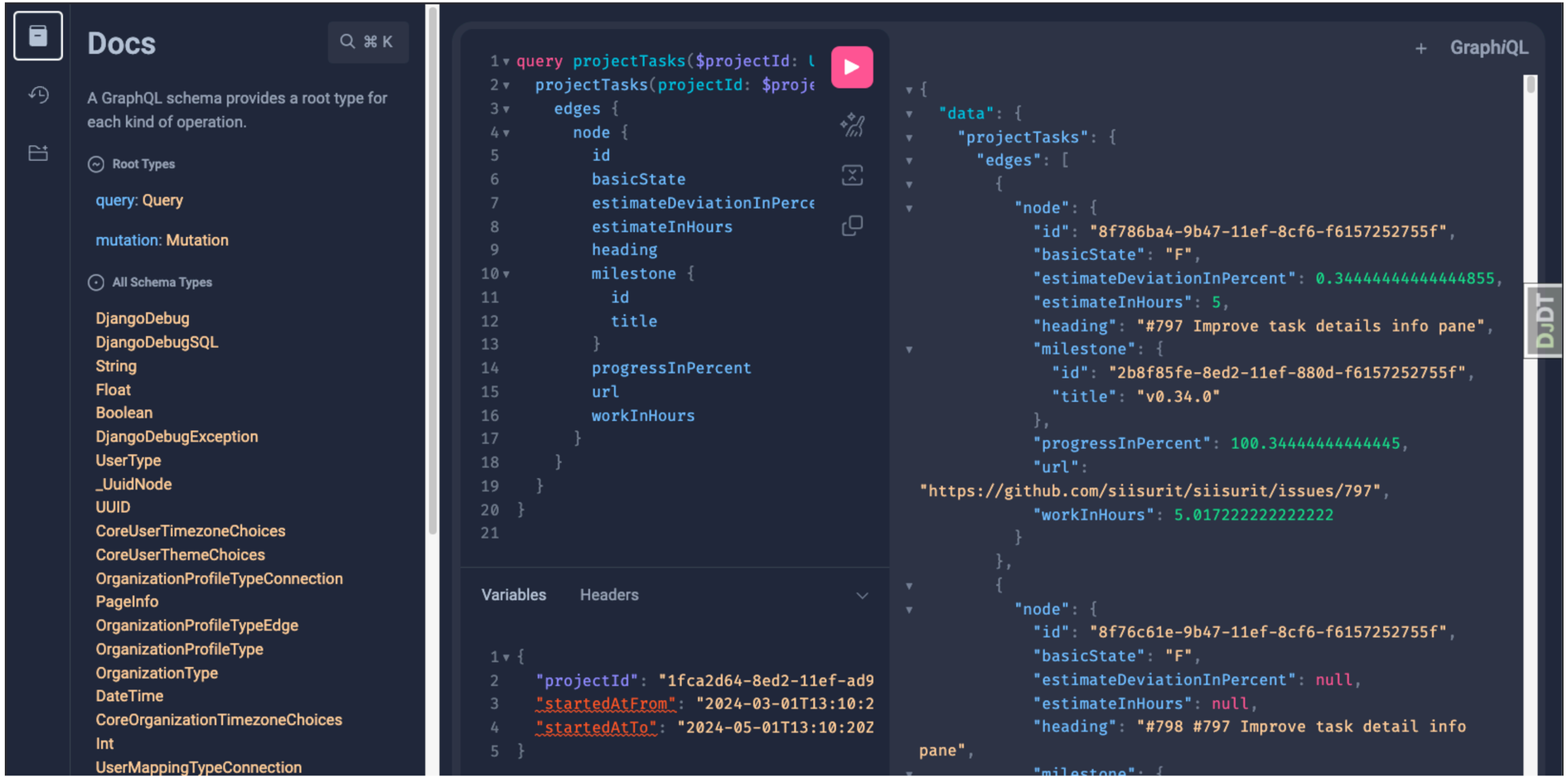
GraphQL schema explorers

Example: GitLab

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

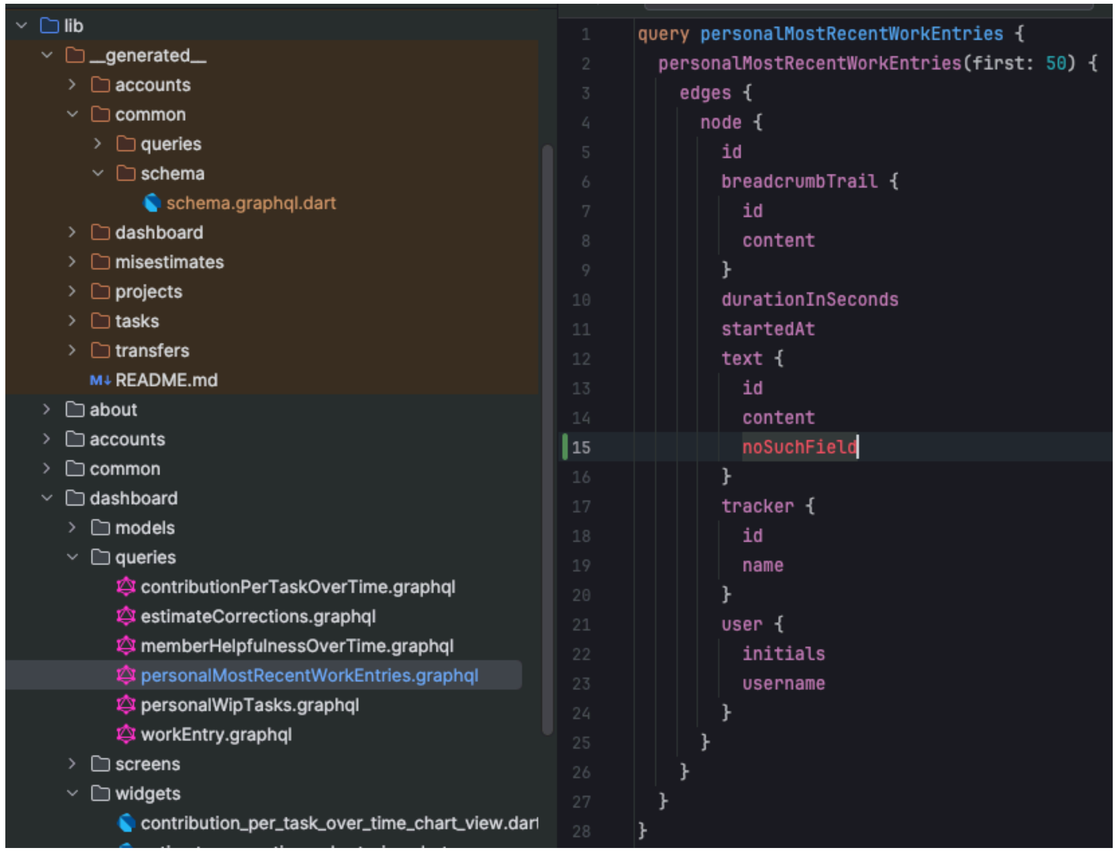
GraphQL schema explorers

Graphene: includes graphiql as Django view



GraphQL schema tool integration

- Schema can also be used to generate code to access data from 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: Flutter and GraphQL with Authentication

Why not just use HTTP bearer token?

- You can.

Why not just use HTTP bearer token?

- You can.

When is JWT useful?

- Pros
  - Attach additional information: For example, store entire user permissions from Active Directory in token → No need to have your own permission management and UI in the application
  - Micro services: Signed token can be trusted, no need have **every** service check with database upon **every** request.
  - Security: Intercepted tokens are only useful for a very short time.
- Cons
  - increased network traffic
  - permission changes only become active after the token expires

GraphQL IDs

- Every GraphQL node must have a unique ID
  - ...that is unique even compared to other types
- Standard solutions:
  - Combine type name and int ID from DB, then do base64
  - Reserve some digits for type ID: 23017 -> object = 23, type = 17, max. 999 types
- Result: Constant encode/decode to get ID actually needed in a specific context
- Possible alternative approach: Use UUID
  - Consider using UUIDv7 = time + random

Multi-tenant security

- Backend typically just maps GraphQL queries to database queries
- Database queries technically can access all data.
- Possibly not every user should see all data.
- You need to limit the database query for every GraphQL query yourself.
- Either by...
  - ...using the permission features in your backend (if there are any)
  - ...adding filter conditions to the database query
- Most of the responsibility to get this right is with you.
- Consider writing unit tests that try to subvert every query/mutation 🤭

GraphQL - The bad 😭

N+1 queries

- Problem: When querying nested graphs: some backends
  - send 1 query to get the first level of the graph
    Example: Task → title, status, due date
  - send N queries to get related information for each task
    Example: Task.assignee → username, email
- Some backend optimize this automatically, some don't.
  - Instead, you to manually implement "resolvers"

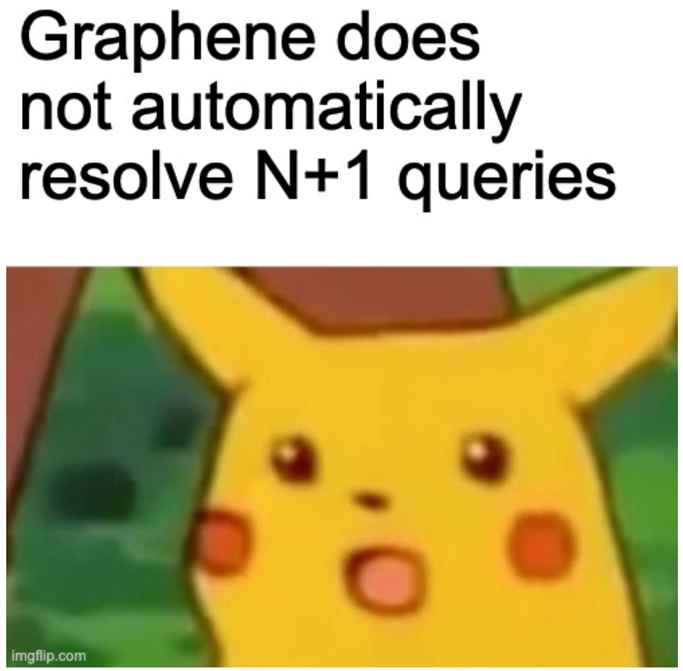
N+1 queries

- 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
    }
  }
}
```

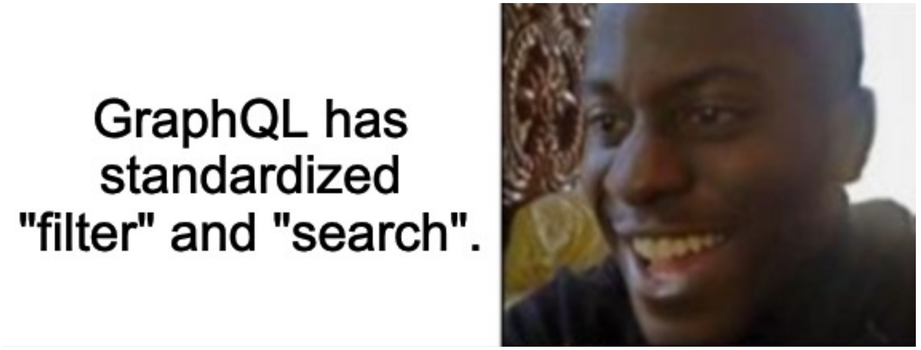
N+1 queries

- Problem: When querying nested graphs: some backends
- Some backend optimize this automatically, some don't.
- If not, you need to manually implement "resolvers"



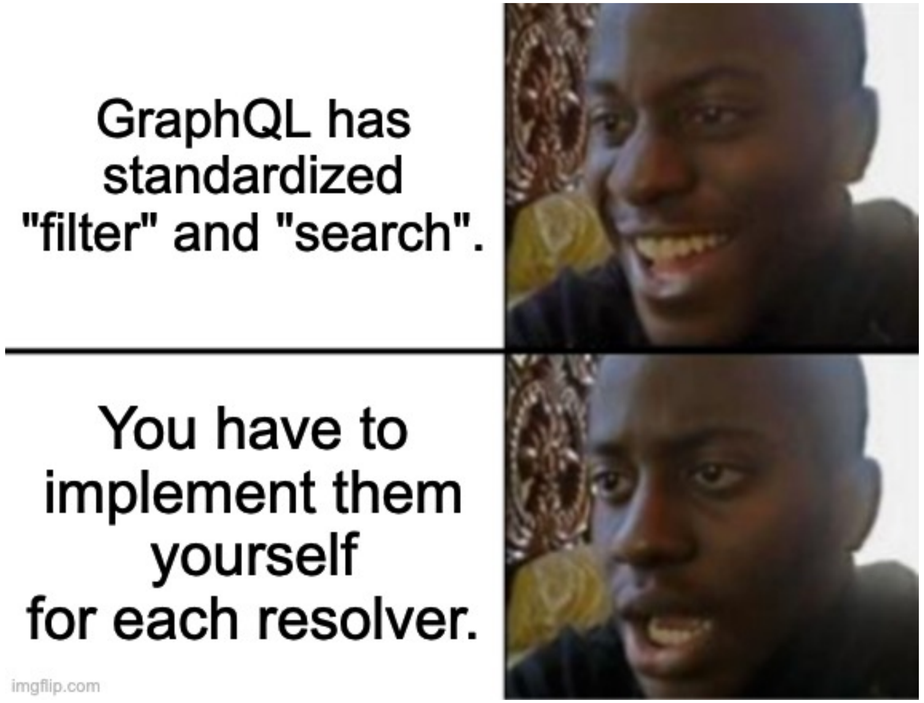
GraphQL filtering

Think SQL-Where



GraphQL filtering

Think SQL-Where



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

Where I use GraphQL

Continuous task and time tracking

Use a task tracker to have unique key for each task

Add finished rod to project risk chart #780

Closed | 1 task done | Add project risk chart #778

roskakori opened this issue 5 days ago · 0 comments · Fixed by #786

roskakori commented 5 days ago • edited

Story

See [illegible]

Goals

- [x] The "finished" rod in the project risk chart shows the work invested over the period in tasks that got done during this period.

Estimate: 2h

Continuous task and time tracking

Use a task tracker to have unique key for each task

Add finished rod to project risk chart #780

Closed | 1 task done | Add project risk chart #778

roskakori opened this issue 5 days ago · 0 comments · Fixed by #786

roskakori commented 5 days ago • edited

Story

See [redacted]

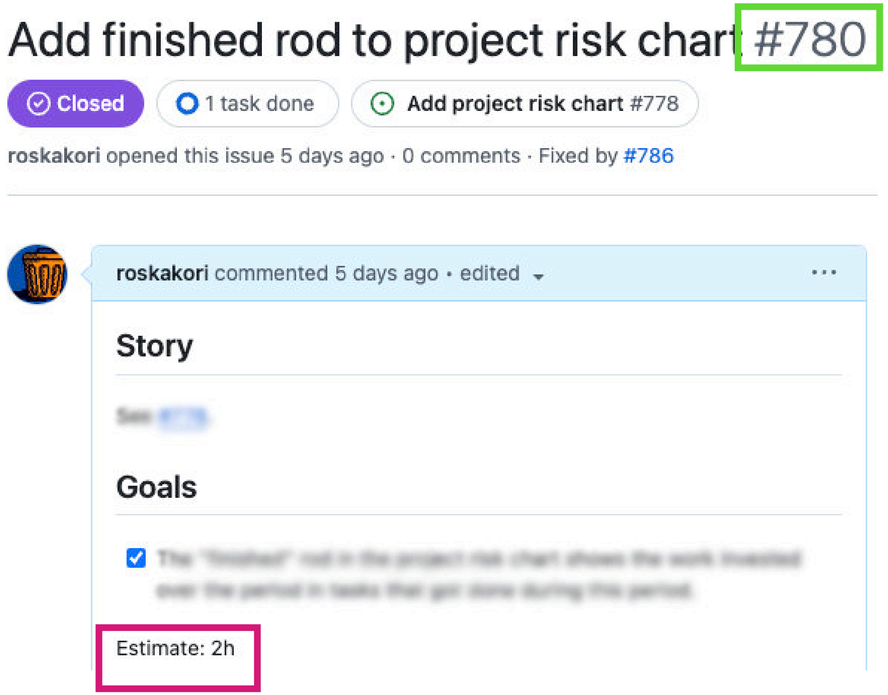
Goals

- [x] The [redacted]

Estimate: 2h

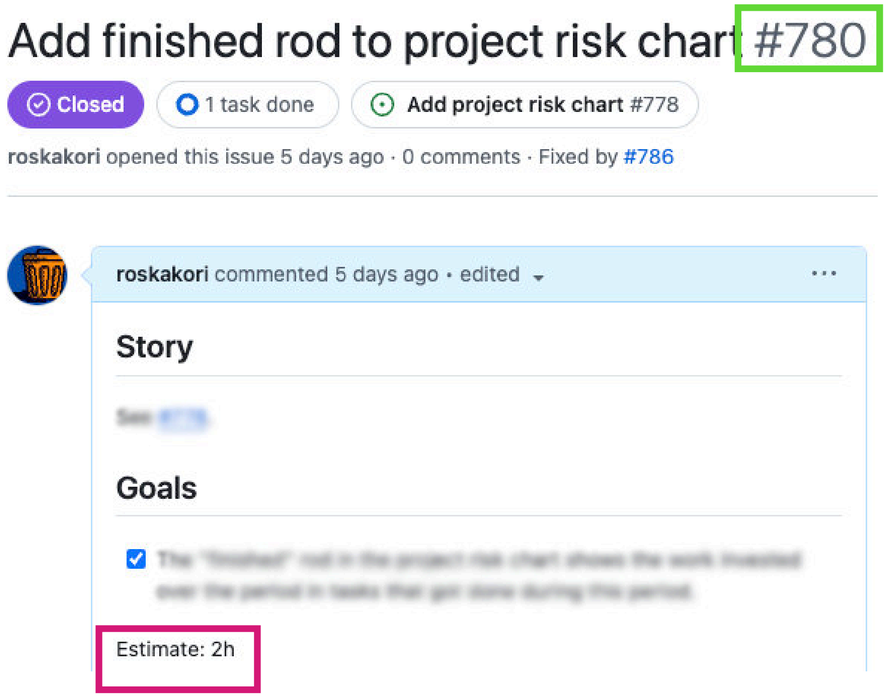
Continuous task and time tracking

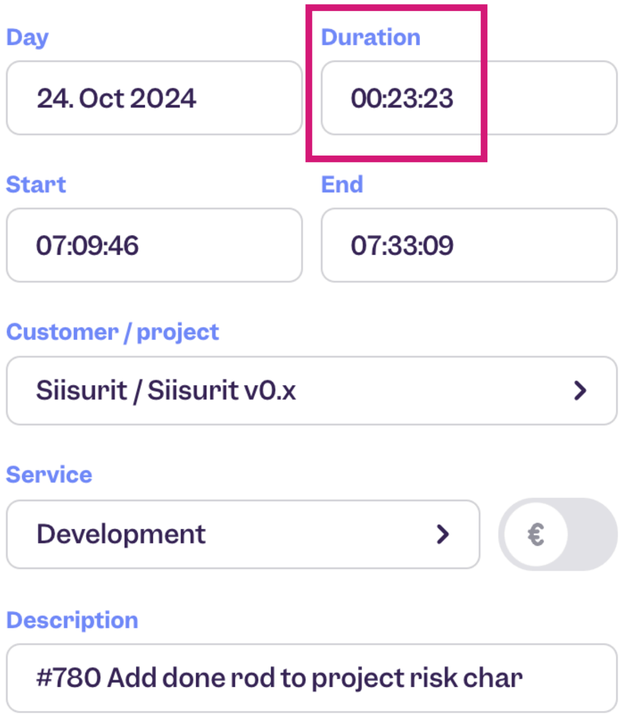
Add estimate of expected effort



Continuous task and time tracking

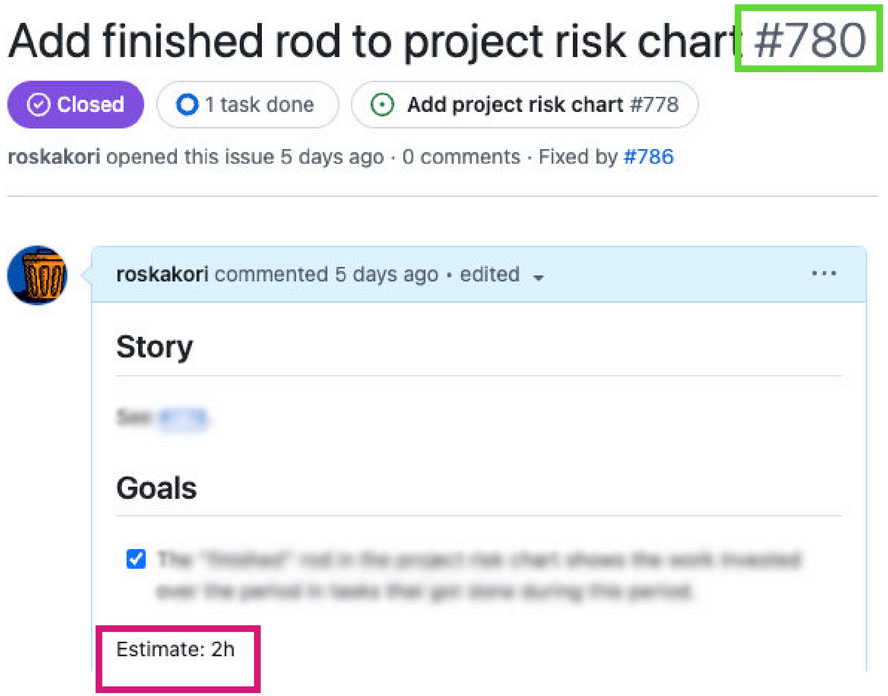
Use time tracker to keep track of actual effort

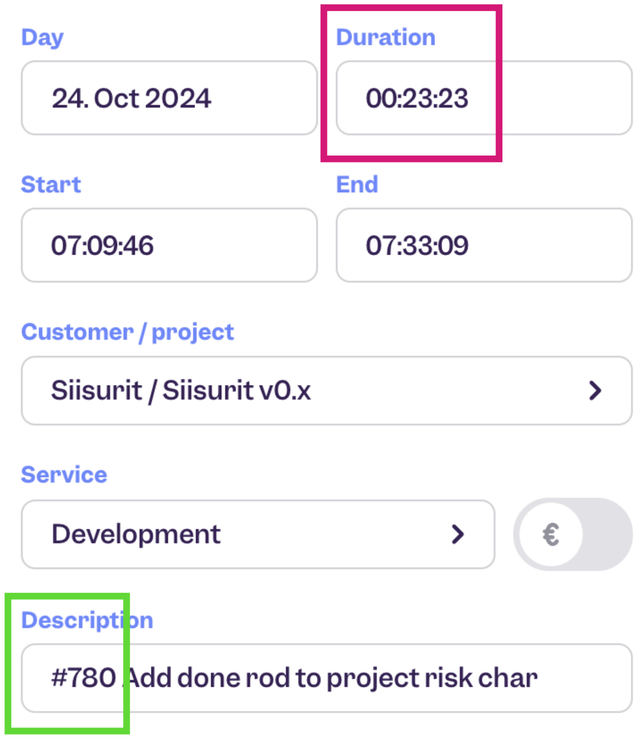




Continuous task and time tracking

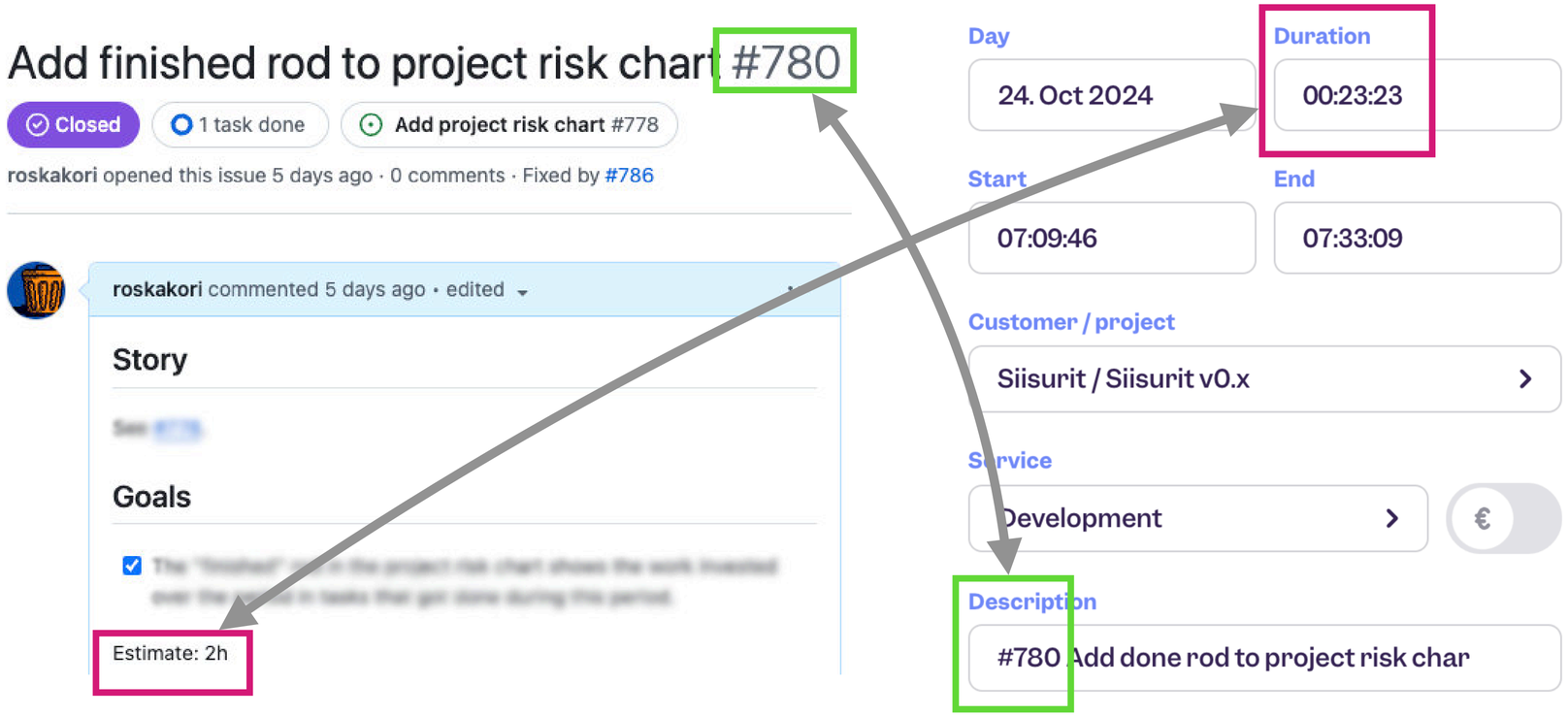
Assign each time entry to a task





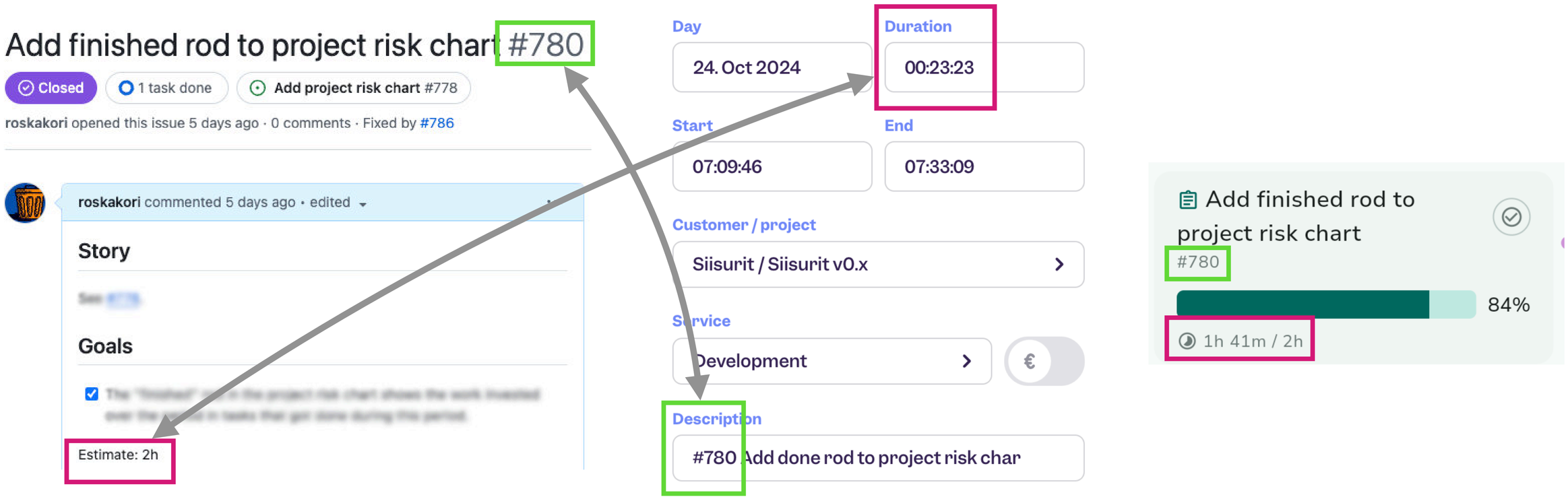
Continuous task and time tracking

These data can be matched...



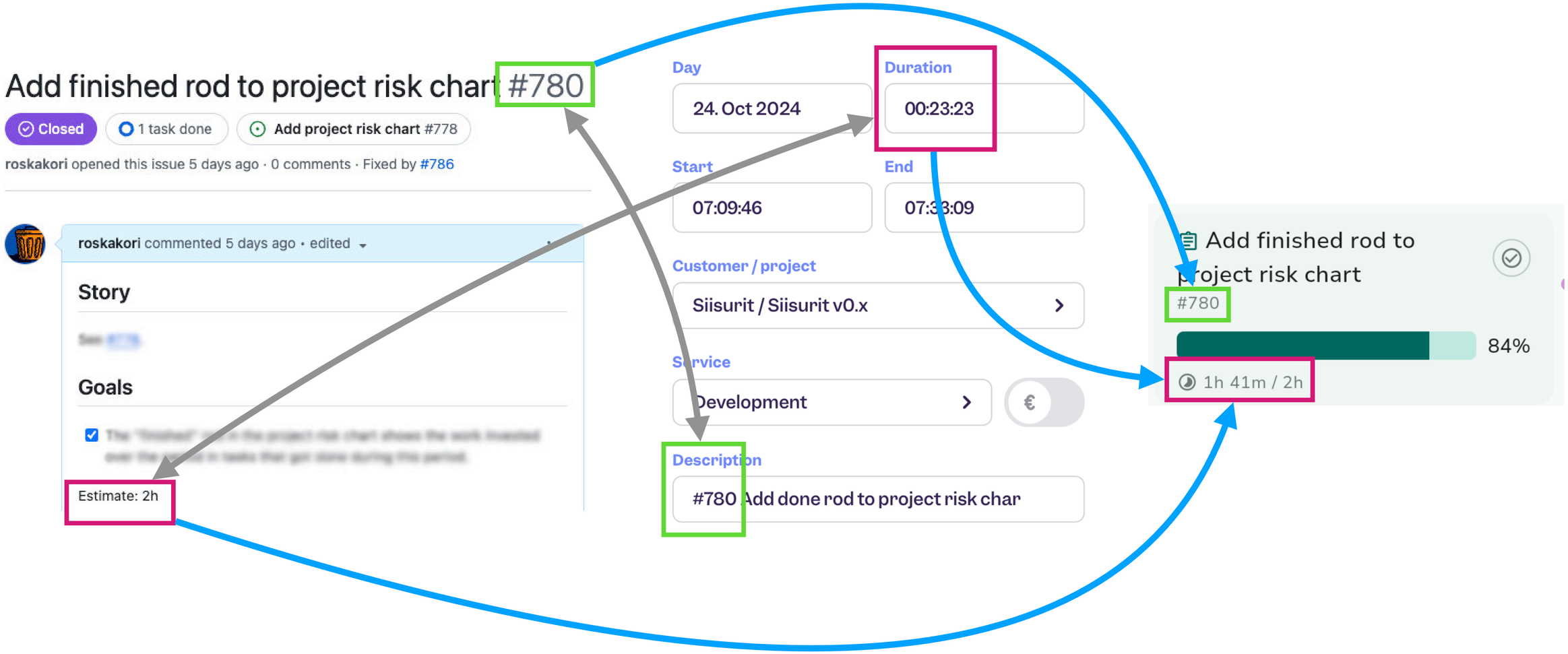
Continuous task and time tracking

...and then be used to compute progress

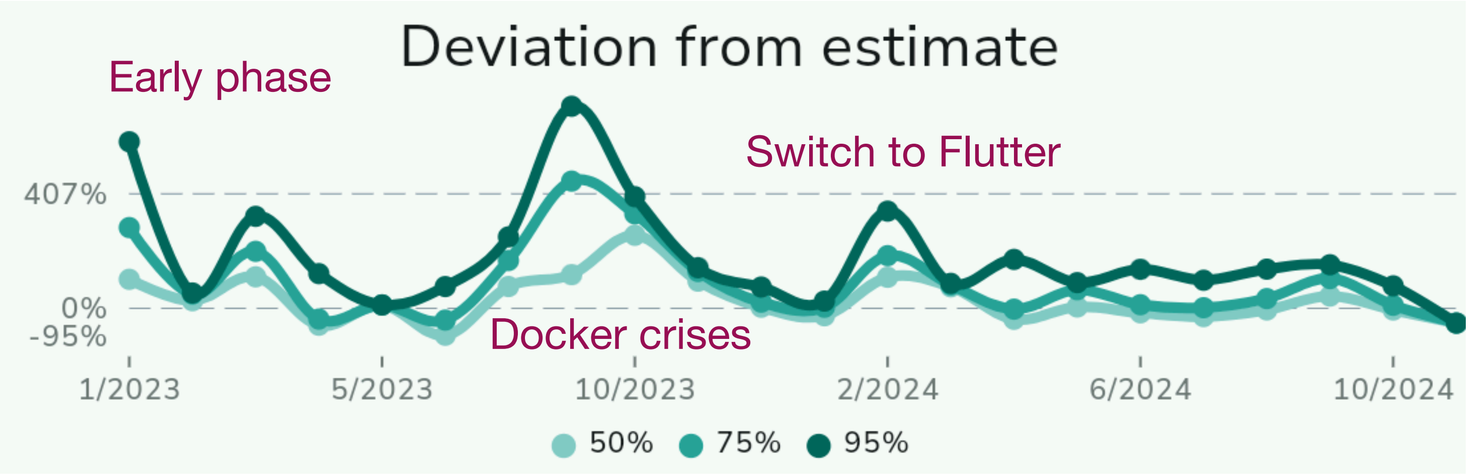


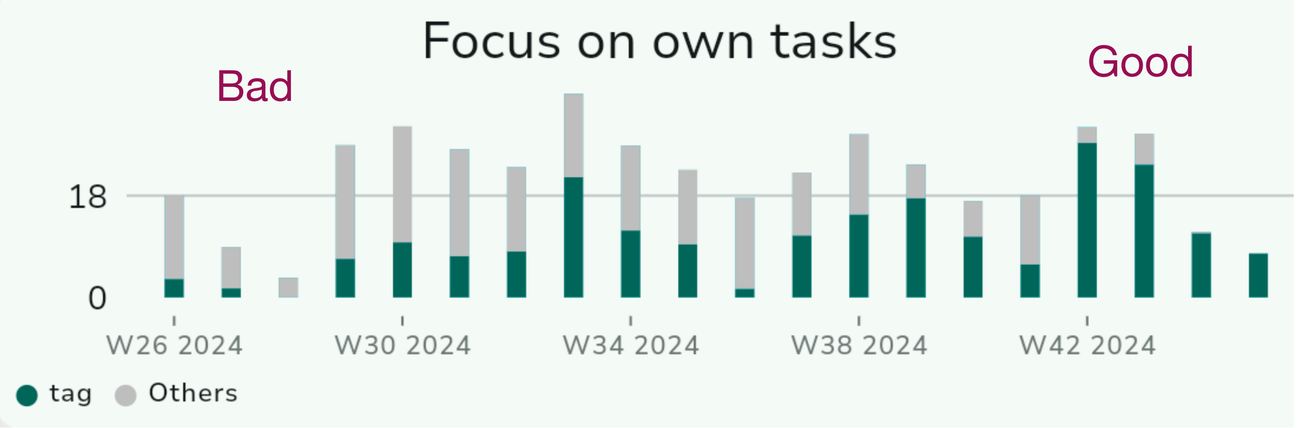
Continuous task and time tracking

...and then be used to compute progress



Example Flutter charts built with GraphQL





Flutter and GraphQL in action

General situation

- Multiple packages can help
- They might add a lot of voodoo and magic in the background
- Error analysis can be frustrating
  - Regular experience in the beginning: Set breakpoints in the external packages to understand what went wrong
- In my cases:
  - limited experience with GraphQL and Flutter
  - but know how to work with HTTP and JSON

graphql_flutter

https://pub.dev/packages/graphql_flutter

1. Create a GraphQL client

```
void main() async {

  final HttpLink httpLink = HttpLink(
    'https://api.github.com/graphql',
  );

  final AuthLink authLink = AuthLink(
    getToken: () async => 'Bearer <YOUR_PERSONAL_ACCESS_TOKEN>',
    // OR
    // getToken: () => 'Bearer <YOUR_PERSONAL_ACCESS_TOKEN>',
  );

  final Link link = authLink.concat(httpLink);

  ValueNotifier<GraphQLClient> client = ValueNotifier(
    GraphQLClient(link: link),
  );

  ...
}
```

graphql_flutter

https://pub.dev/packages/graphql_flutter

1. Create a GraphQL client
2. Wrap GraphQLProvider around app

```
return GraphQLProvider(
  client: client,
  child: MaterialApp(
    title: 'Flutter Demo',
    ...
  ),
);
```

graphql_flutter

https://pub.dev/packages/graphql_flutter

1. Create a GraphQL client
2. Wrap GraphQLProvider around app
3. Prepare query

```
String readRepositories = """
  query ReadRepositories(\$nRepositories: Int!) {
    viewer {
      repositories(last: \$nRepositories) {
        nodes {
          id
          name
          viewerHasStarred
        }
      }
    }
  }
""";
```

graphql_flutter

https://pub.dev/packages/graphql_flutter

1. Create a GraphQL client
2. Wrap GraphQLProvider around app
3. Prepare query
4. Send query

```
Query(
  options: QueryOptions(
    document: gql(readRepositories), // this is the query string you just created
    variables: {
      'nRepositories': 50,
    },
    pollInterval: const Duration(seconds: 10),
  ),
  builder: (QueryResult result, { VoidCallback? refetch, FetchMore? fetchMore }) {
    if (result.hasException) {
        return Text(result.exception.toString());
    }

    if (result.isLoading) {
      return const Text('Loading');
    }

    List? repositories = result.data?['viewer']?['repositories']?['nodes'];

    if (repositories == null) {
      return const Text('No repositories');
    }

    return ListView.builder(
      itemCount: repositories.length,
      itemBuilder: (context, index) {
        final repository = repositories[index];

        return Text(repository['name'] ?? '');
    });
  },
);
```

graphql_flutter

https://pub.dev/packages/graphql_flutter

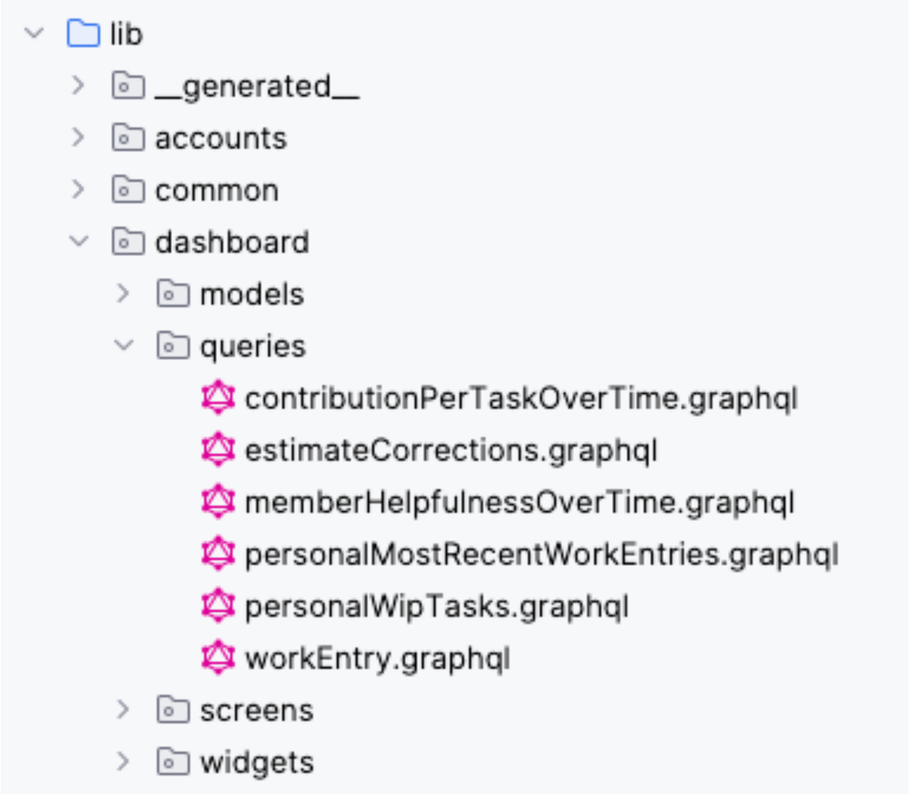
1. Create a GraphQL client
2. Wrap GraphQLProvider around app
3. Prepare query
4. Send query

```
Query(
  options: QueryOptions(
    document: gql(readRepositories), // this is the query string you just created
    variables: {
      'nRepositories': 50,
    },
    pollInterval: const Duration(seconds: 10),
  ),
  builder: (QueryResult result, { VoidCallback? refetch, FetchMore? fetchMore }) {
    if (result.hasException) {
        return Text(result.exception.toString());
    }

    if (result.isLoading) {
      return const Text('Loading');
    }

    List? repositories = result.data?['viewer']?['repositories']?['nodes'];

    if (repositories == null) {
      return const Text('No repositories');
    }

    return ListView.builder(
      itemCount: repositories.length,
      itemBuilder: (context, index) {
        final repository = repositories[index];

        return Text(repository['name'] ?? '');
    });
  },
);
```

Managing GraphQL source code

Good practice independent of how you send queries

- Store code in pubspec assets
- Load before sending query
- Pros:
  - Syntax highlighting
  - Schema validation while typing



dio

Authentication

- Open a HTTP session
- Authenticate
- Wrap it in a riverpod StateNotifier to sign in and out
- After sign in with new user, widgets rebuild automatically

```
class SignInInfo {
  SignInInfo({required this.dio, required this.userInfo});

  Dio dio;
  ... // Application specific attributes
}

class SignInInfoNotifier extends StateNotifier<SignInInfo?> {
  SignInInfoNotifier() : super(null);

  Future<void> signIn(
    final String username,
    final String password,
    final String serverUrl,
  ) async {
    final dio = await _signedInDio(username, password, serverUrl);
    ... // Application specific logic
    state = SignInInfo(dio: dio, userInfo: userInfo);
  }
  ...
}
```

dio

Send queries and mutations

- Load query from bundle
- Send POST queries with JSON payload
- Parse JSON result: {"data": ..., "errors": ...}
  - if has "errors" is not null, throw Exception
  - otherwise, extract "data"

```
Future<T> _queryResult<T>(
  Dio dio,
  String logical,
  String queryName,
  Map<String, dynamic>? graphqlVariables,
) async {
  final query = await rootBundle.loadString("lib/$logical/queries/$queryName.graphql");
  final Response response = await dio.post<Map<String, dynamic>>(
    "/api/graphql/",
    data: {
      "query": query,
      "variables": graphqlVariables,
    },
  );
  final Map<String, dynamic> jsonMap = response.data;
  final List? possibleErrorMaps = jsonMap["errors"];
  if (possibleErrorMaps != null) {
    final List<Map<String, dynamic>> errorMaps = possibleErrorMaps.cast<Map<String, dynamic>>();
    if (errorMaps.isEmpty) {
      throw const _GraphQlException("ErrorMaps must contain data.");
    }
    final String message = errorMaps[0]["message"] ?? "Unknown error";
    throw _GraphQlException(message, errorMaps);
  }
  final Map<String, dynamic>? jsonData = jsonMap["data"];
  if (jsonData == null) {
    throw _GraphQlException("GraphQL JSON response must contain data.", jsonMap);
  }
  final Map<String, T?> queryData = Map<String, T>.from(jsonData);
  final T? queryResult = queryData[queryName];
  if (queryResult == null) {
    throw _GraphQlException("GraphQL query must contain $queryName", queryResult);
  }
  return queryResult;
}
```

graphql_codegen

https://pub.dev/packages/graphql_codegen

- Parses schema and your queries / mutations
- Generates Dart code with JSON de-/serializers from it
- Configure with build.yaml
- Run via build_runner or shell script

```yaml
targets:
  $default:
    builders:
      graphql_codegen:
        options:
          assetsPath: "lib/**.graphql"
          generatedFileHeader: "// ignore_for_file: type=lint\n"
          outputDirectory: "/lib/__generated__"
          scalars:
            # HACK: Declare DateTime explicitly because for
            # some reason it is not a built-in scalar.
            DateTime:
              type: DateTime
            UUID:
              type: String
```

graphql_codegen

Example generated code for user model

Interactive demo because the generated code is quite large and complex

Reuse code generated from GraphQL

- Every *.graphql results their own class
- Even when the result type is the same GraphQL type
  - active tasks in dashboard
    → Query$personalWipTasks$personalWipTasks.fromJson
  - tasks in project details screen
    → Variables$Query$projectTasks.fromJson
- Every class has a "fromJson" but no common base class
- Cannot easily use these classes as generic parameter to helper function

Reuse code generated from GraphQL

Solution: Pass both type and fromJson function as parameter

- Generic type to function defines result type if GraphQL helper function
- fromJson parameter defines how to convert a GraphQL JSON map to a Dart object
- caveat: No static check that type and function match

```
Future<List<TaskInfo>> futureTaskInfosById(
  SignInInfo signInInfo,
  List<String> taskIds,
}) async {
  final List<TaskInfo> result = [];
  final taskNodes = await graphQlItemList<Query$tasksById$tasksById>(
    signInInfo: signInInfo,
    fromJson: Query$tasksById$tasksById.fromJson,
    logical: "common",
    queryName: "tasksById",
    graphqlVariables: {"ids": taskIds},
  );
  for (final taskNode in taskNodes) {
    final assigneeInfos = hasAssignees ? _assigneeInfos(taskNode) : null;
    final labelInfos = hasLabels ? _labelInfos(taskNode) : null;
    final taskInfo = TaskInfo(
      assigneeInfos: assigneeInfos,
      id: taskNode.id,
      labelInfos: labelInfos,
      title: taskNode.title,
      url: hasUrl ? taskNode.url : null,
      ...
    );
    result.add(taskInfo);
  }
  return result;
}
```

Reuse with code generated from GraphQL

Naming conventions

- taskMap = raw JSON Map()
- taskNode = class generated from GraphQL schema
- taskInfo = data object with a type that the application uses to represent a task
  - Note: named "info" instead "data" to have proper plural "infos"

```
class Query$tasksById$tasksById {
  Query$tasksById$tasksById({
    required this.id,
    required this.assignees,
    required this.labels,
    required this.title,
    ...
  });

  factory Query$tasksById$tasksById.fromJson(Map<String, dynamic> json) {
    ...
  }
```

```
class TaskInfo extends Equatable with Compare<ContributorInfo> {
  TaskInfo({
    this.assigneeInfos,
    this.id,
    this.labelInfos,
    this.title,
    ...
  }) {...}
```

Query and caching

- Simple cached queries to obtain all fields of basic building blocks like tasks.
- Complex queries that return only the ID of tasks, e.g. my current active tasks or all tasks belonging to a project.

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 other package's code
- Check if GraphQL actually solves more problems than it causes.
- Check if JWT actually solves more problems than it causes. If not, consider using bearer tokens.

Summary

GraphQL and Flutter

- Dart/Flutter packages exist for both high- and low-level handling of GraphQL
- Code generation from GraphQL schema works well but makes reuse difficult
- Personal design decision: value a transparent approach over more integrated packages
  - dio and helper functions
  - manual caching
- YMMV

Ask for early access: hello@siisurit.com

https://siisurit.com/