AI support for project management

A story from the trenches

Thomas Aglassinger, 2025-09-24

© 2025 by Siisurit, licensed under CC BY-SA 4.0. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/

About me

Thomas Aglassinger

- Software developer with 20+ years industry experience
  - various sectors: banking, health, e-commerce, contract work
- Founder of Siisurit https://siisurit.com/
- Part-time at Providence Analytics https://providens.at/
- Common ground: process, query, visualize data
- Used many tools for that, AI and LLM in particular is one of them.

Project management

- Wikipedia: "Project management is the process of supervising the work of a team to achieve all project goals within the given constraints."
- Constraints: People, budget, time, quality
- Many companies track projects in
  - ticket trackers
  - time trackers

Project controlling

- "How close am I to where I want to be?"
- "What actions should I take to get back on track?"
- Can be challenging
- The data are there, but hard to relate to each other
- Often: Planned vs. actual budget at the end of **project** or milestone
- More useful: Planned vs actual budget of a **ticket**

Ticket controlling

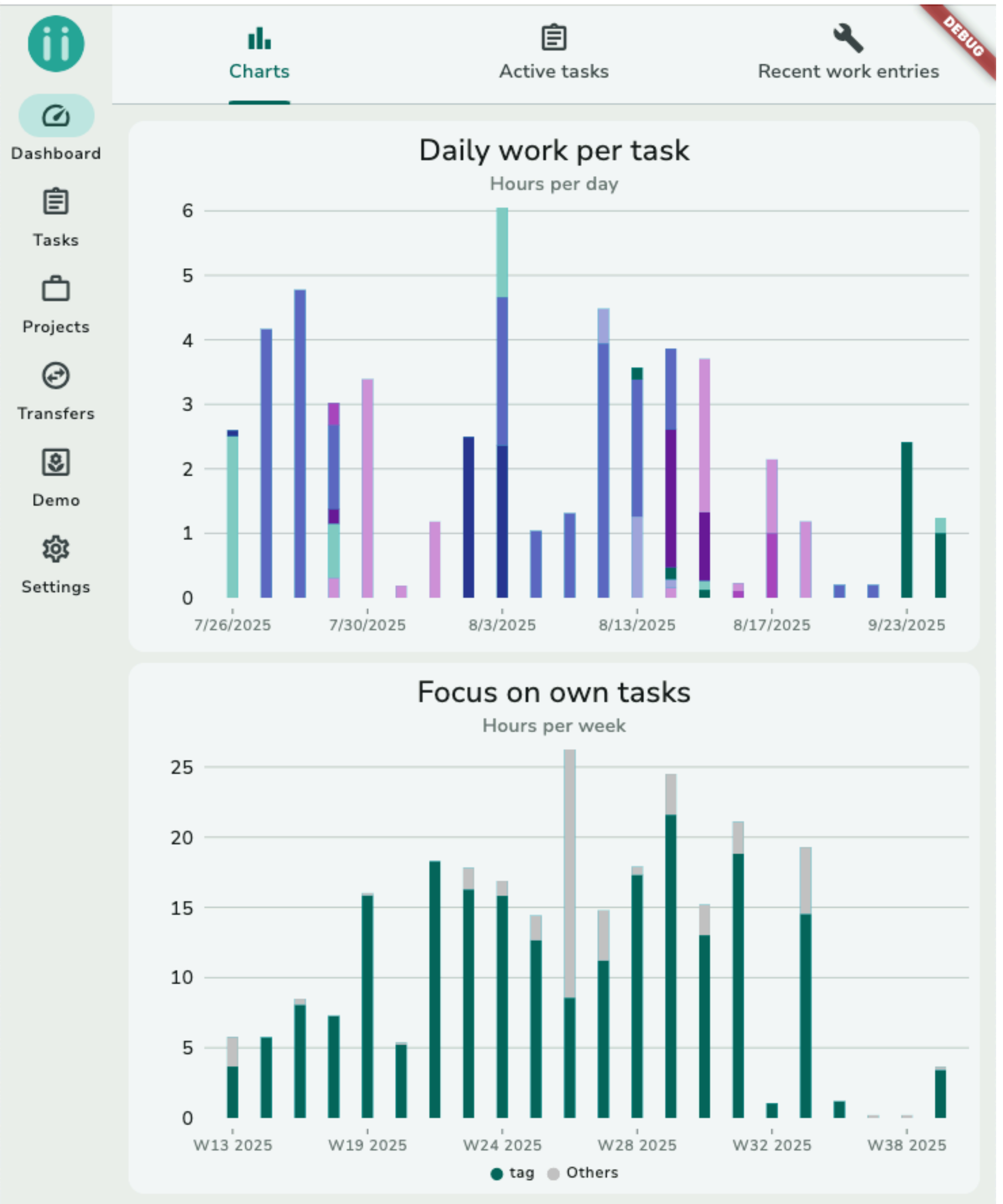
- Common approaches:
  - Integrated solution
  - Wild scripts and Excel macros
    (maintained by project manager, regularly break)
- Application that can connect tickets and work time using API of the trackers

Siisurit and its layers of information

- Layer 1: Dashboard and standard views in an application
- Layer 2: Structured data to connect through business intelligence (BI) tools for individual reports
- Layer 3: Artificial intelligence (AI)

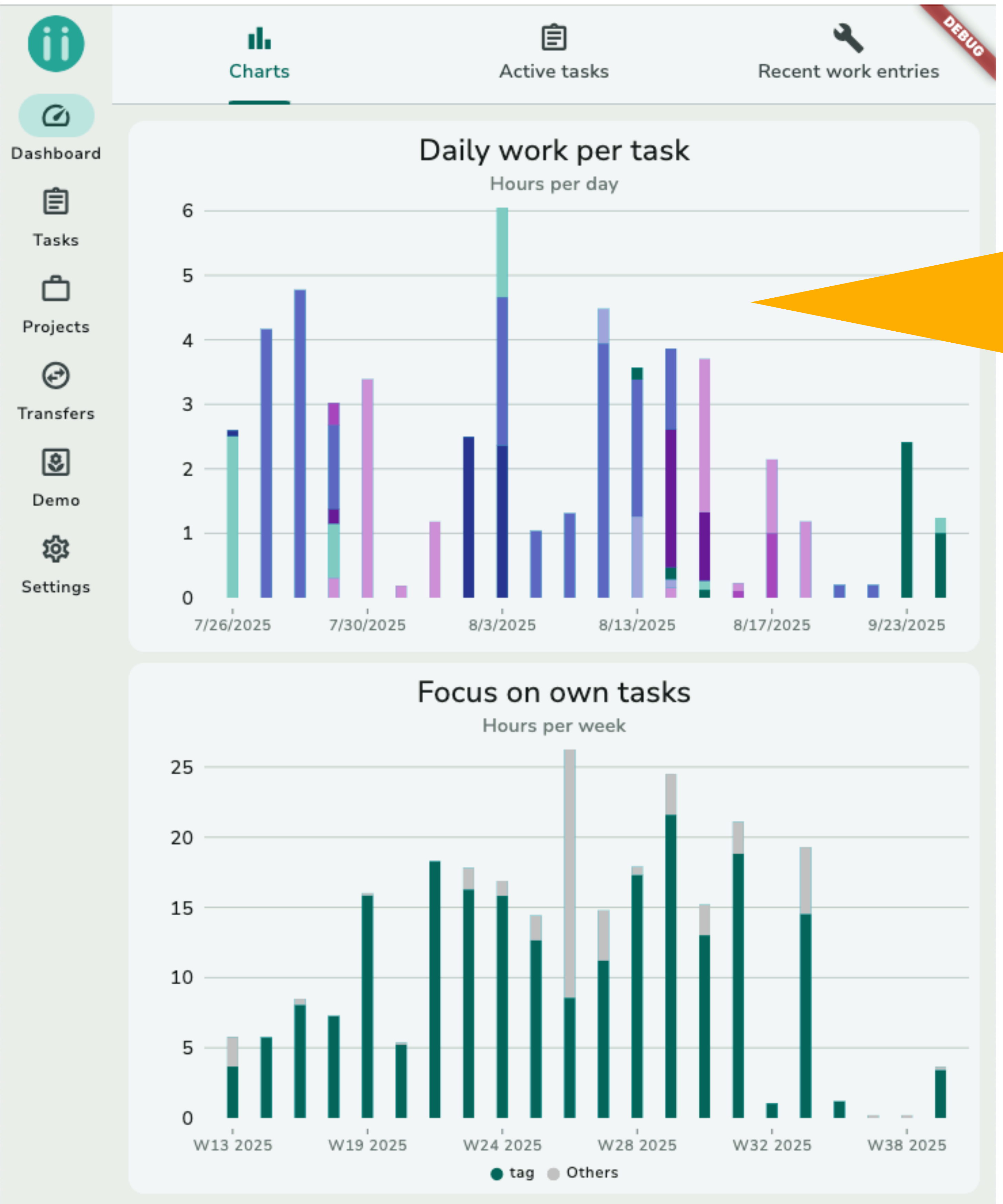
Dashboard and standard views in an application

Information layer 1



Dashboard and standard views in an application

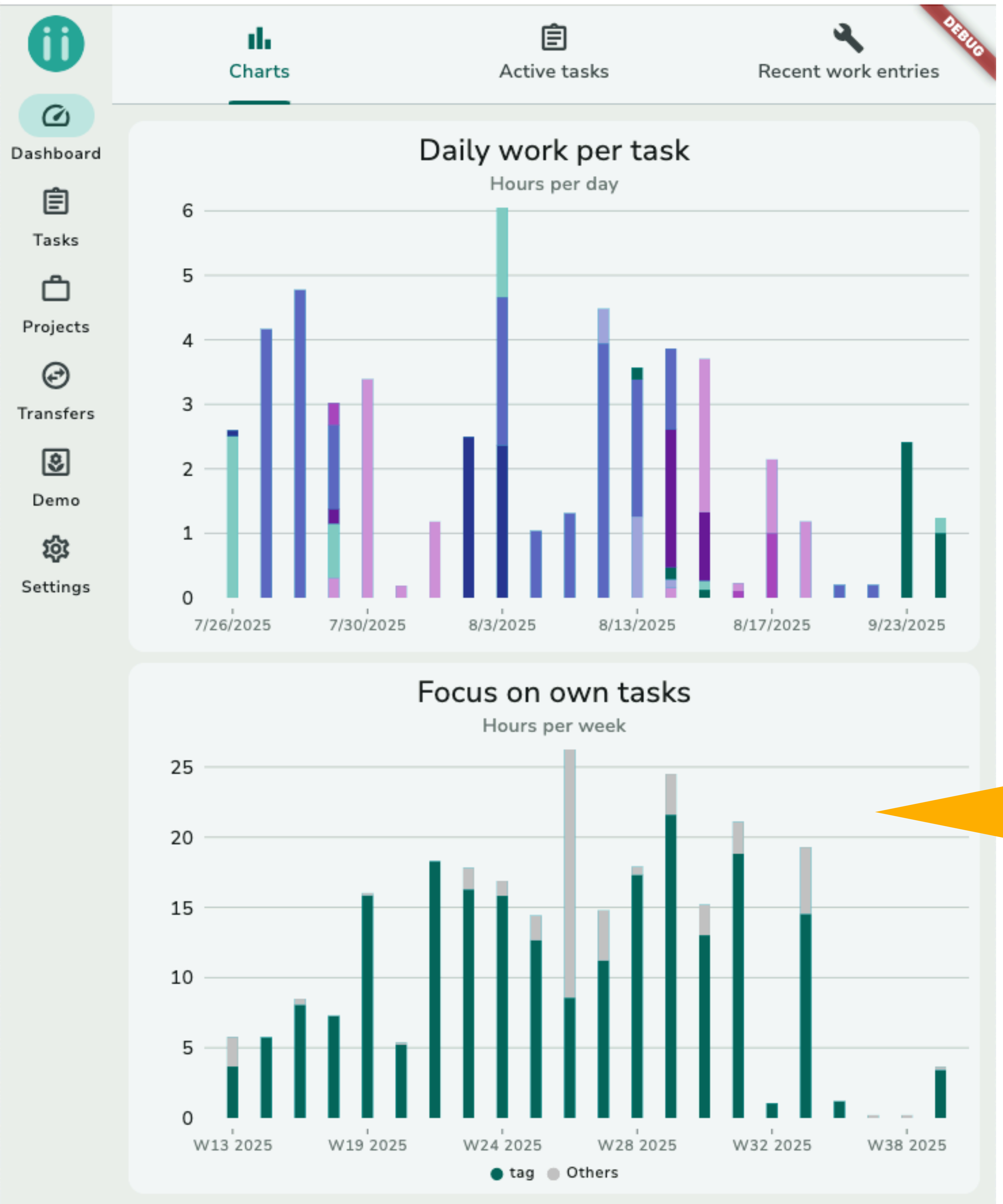
Information layer 1



How much of a mess is my life?

Dashboard and standard views in an application

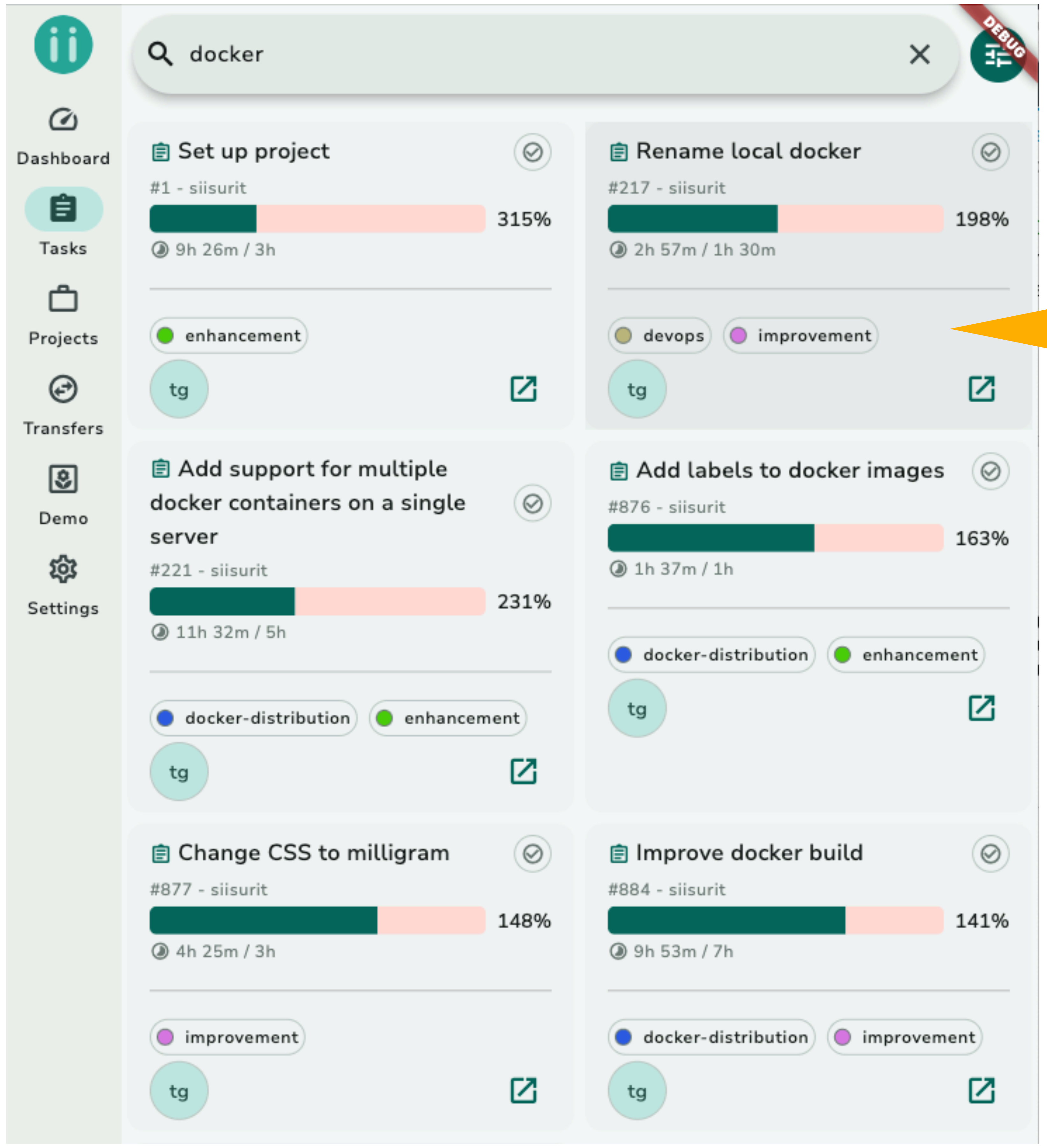
Information layer 1



How much do I work on things actually assigned to me?

Dashboard and standard views in an application

Information layer 1



docker

Set up project
#1 - siisurit
315%
9h 26m / 3h
enhancement
tg

Rename local docker
#217 - siisurit
198%
2h 57m / 1h 30m
devops
improvement
tg

Add support for multiple docker containers on a single server
#221 - siisurit
231%
11h 32m / 5h
docker-distribution
enhancement
tg

Add labels to docker images
#876 - siisurit
163%
1h 37m / 1h
docker-distribution
enhancement
tg

Change CSS to milligram
#877 - siisurit
148%
4h 25m / 3h
improvement
tg

Improve docker build
#884 - siisurit
141%
9h 53m / 7h
docker-distribution
improvement
tg

How bad are we at estimating docker related tickets, and do we at least improve?

Structured data to connect through BI tools for individual reports

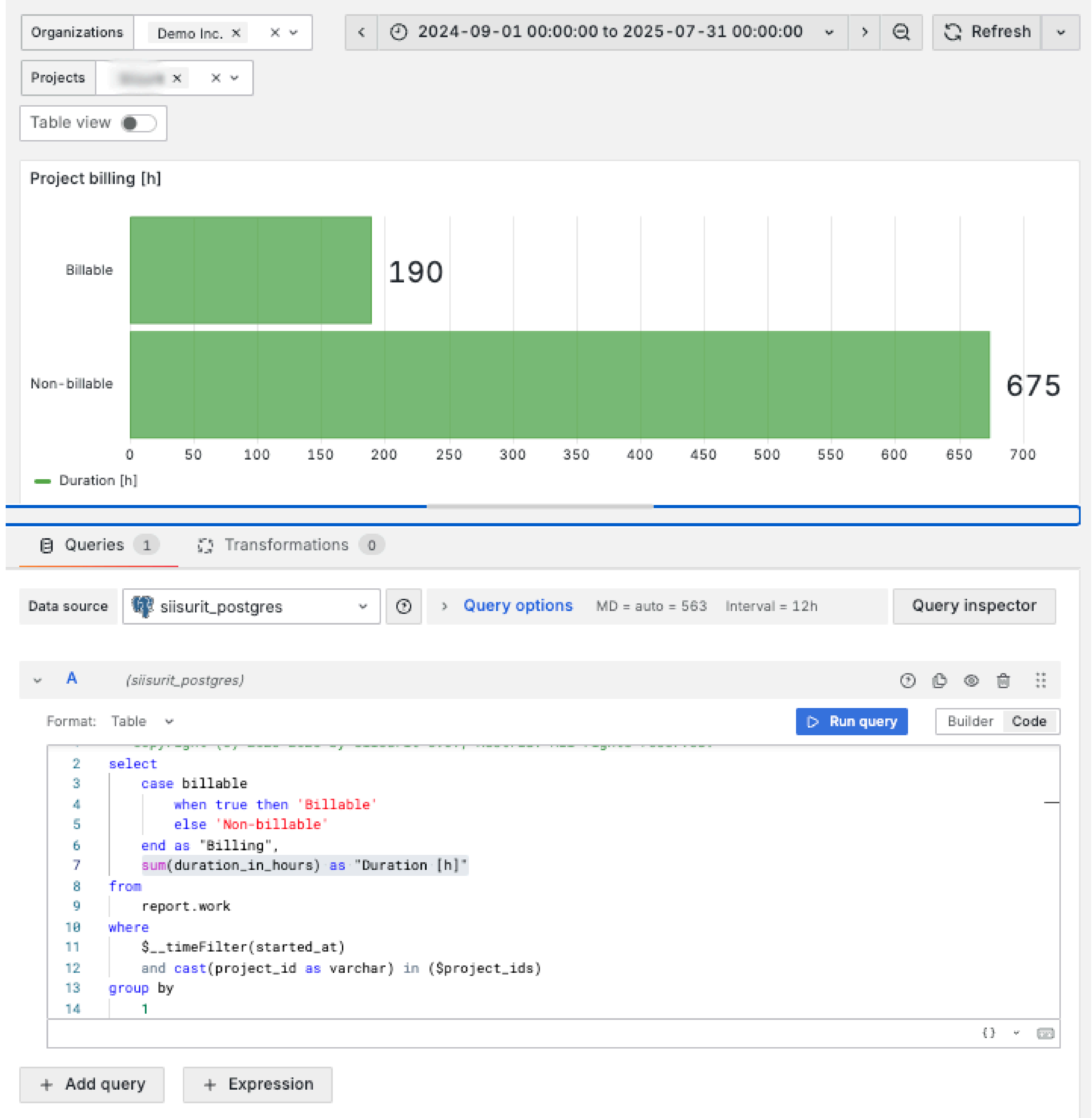
Information layer 2

- Provide structured data for reporting
- Siisurit: Flattened SQL views (PostgreSQL)
  - Read-only
  - No or view joins
  - If necessary: use materialized SQL views for performance
  - Many BI tools can connect to it

Structured data to connect through BI tools for individual reports

Information layer 2

- Example: Excel
  - ubiquitous
- Example: Grafana
  - Basic edition is open source
  - Can turn SQL statement into Chart



Organizations: Demo Inc.

Projects: [redacted]

2024-09-01 00:00:00 to 2025-07-31 00:00:00

Data source: siisurit_postgres

Query options MD = auto = 563 Interval = 12h

A (siisurit_postgres)

Format: Table

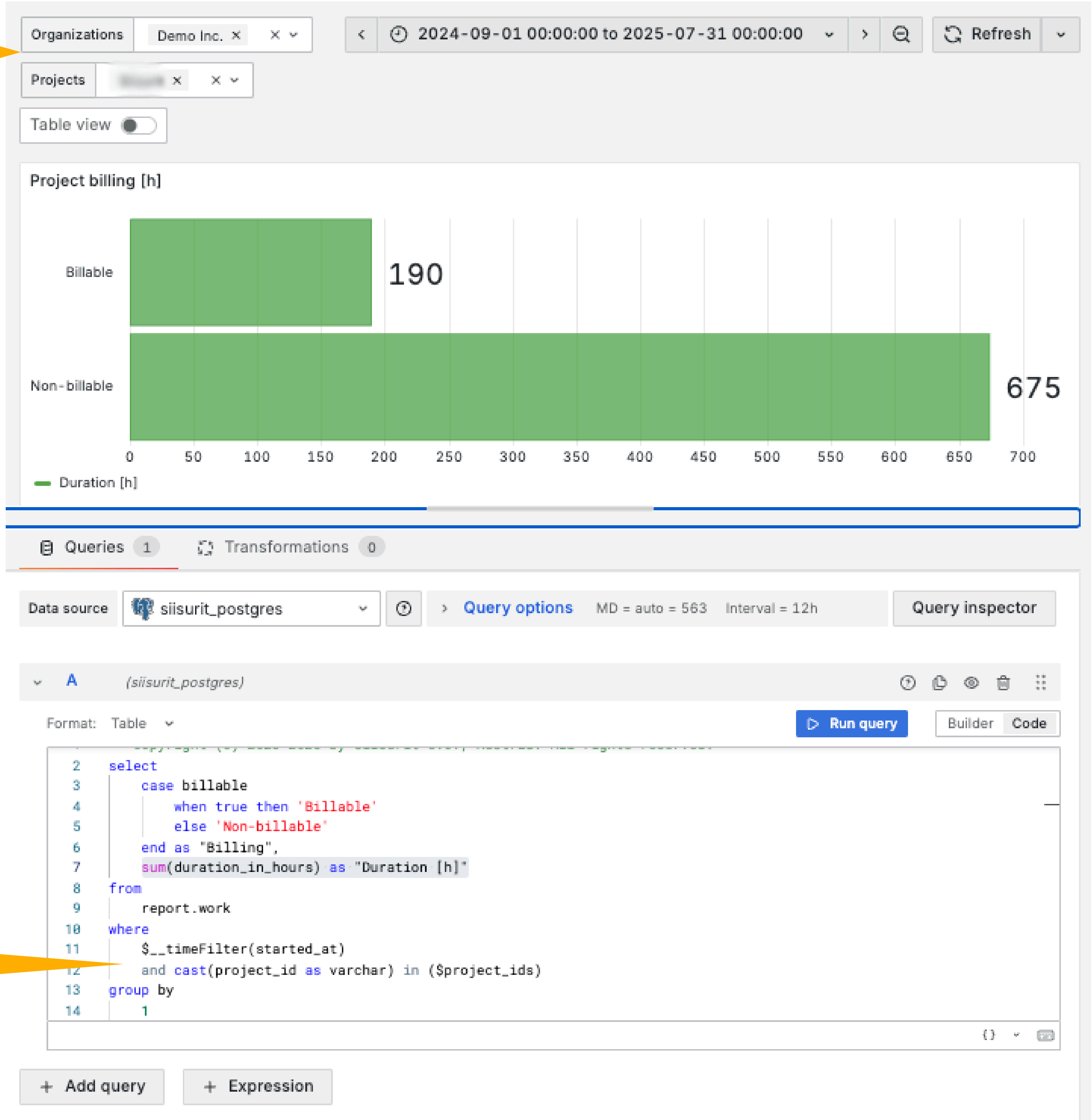
```sql
select
    case billable
        when true then 'Billable'
        else 'Non-billable'
    end as "Billing",
    sum(duration_in_hours) as "Duration [h]"
from
    report.work
where
    $__timeFilter(started_at)
    and cast(project_id as varchar) in ($project_ids)
group by
    1
```

Structured data to connect through BI tools for individual reports

Information layer 2

Interactive Filter

- Example: Excel
  - ubiquitous
- Example: Grafana
  - Basic edition is open source
  - Can turn SQL statement into Chart



```
select
    case billable
        when true then 'Billable'
        else 'Non-billable'
    end as "Billing",
    sum(duration_in_hours) as "Duration [h]"
from
    report.work
where
    $__timeFilter(started_at)
    and cast(project_id as varchar) in ($project_ids)
group by
    1
```

Filter variables in

Intermission: Currently supported trackers

For now

- GitHub
- Jira
- Trello
- GitLab
- Gitea / Forgejo
- YouTrack
- Kimai
- Clockodo
- Toggl track
- Troi timr

Intermission: Currently supported trackers

For now

- GitHub
- Jira
- Trello
- GitLab
- Gitea / Forgejo
- YouTrack
- Kimai
- Clockodo
- Toggl track
- Troi timr

Carefully handmade solution to MxN problem

💪 🤪 🦕

Layer 3: AI & LLM

Layer 3: AI & LLM

(Finally! About time!)

Common concerns

- "Unrestricted access for everyone"
- "Sensitive data end up in remote IT centers and can be accessed by random people we have no control over."
- "AI confabulates"

Common concerns

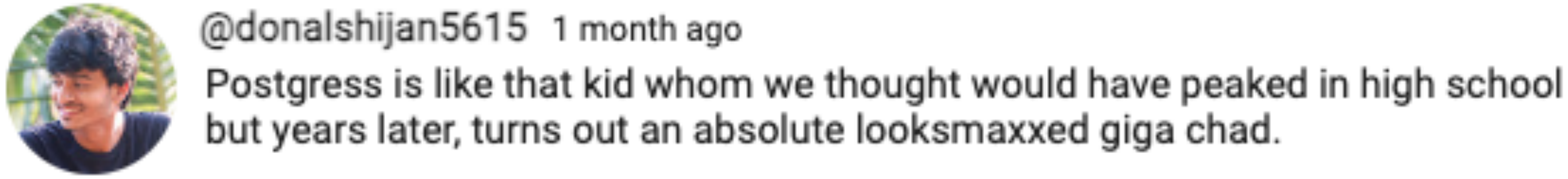
Permission system even for AI related queries

- "Unrestricted access for everyone"
- "Sensitive data end up in remote IT centers and can be accessed by random people we have no control over."
- "AI confabulates"

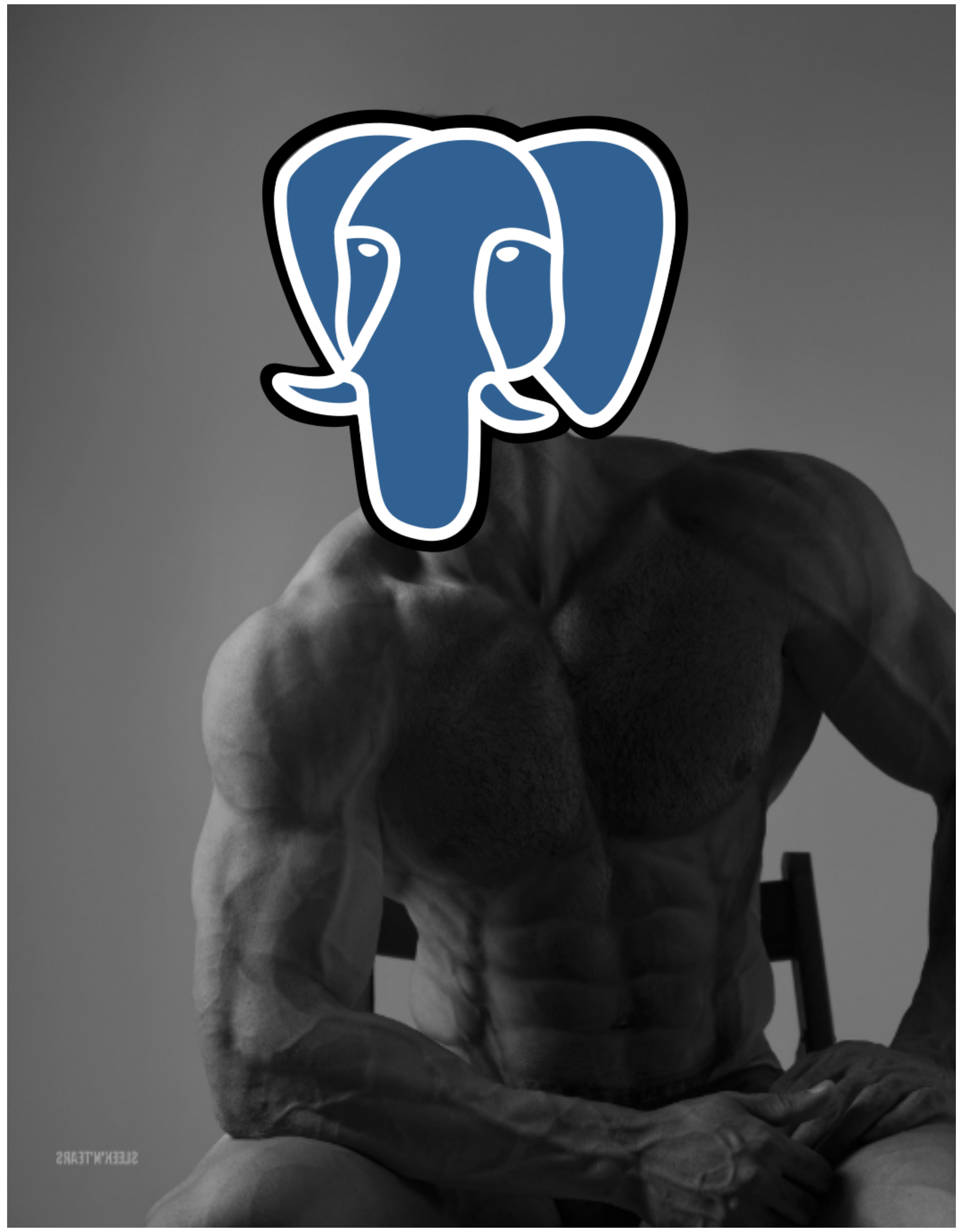
Permission system even for AI related queries

- Use a database that can do both...
    - relational data (for permission tables) and...
    - and vectorized data (for AI queries).
- Join and filter depending on user that queries the data.

PostgreSQL


@donalshijan5615 1 month ago
Postgress is like that kid whom we thought would have peaked in high school but years later, turns out an absolute looksmaxxed giga chad.

Video: https://youtu.be/3JW732GrMdg



Common concerns

- "Unrestricted access for everyone"
- "Sensitive data end up in remote IT centers and can be accessed by random people we have no control over."
- "AI confabulates"

Provide an on-premise solution without inherent need for cloud

How to do RAG and MCP without cloud?

- Embed with Ollama: https://ollama.com/
- Store vectors in PostgreSQL (with pgvector extension)
- Use retrieval-augmented generation (RAG) for semantic search and simple chat
- Add model context protocol (MCP)
  - For Python-Django backend: https://github.com/omarbenhamid/django-mcp-server
  - MCP command line client: https://github.com/jonigl/mcp-client-for-ollama

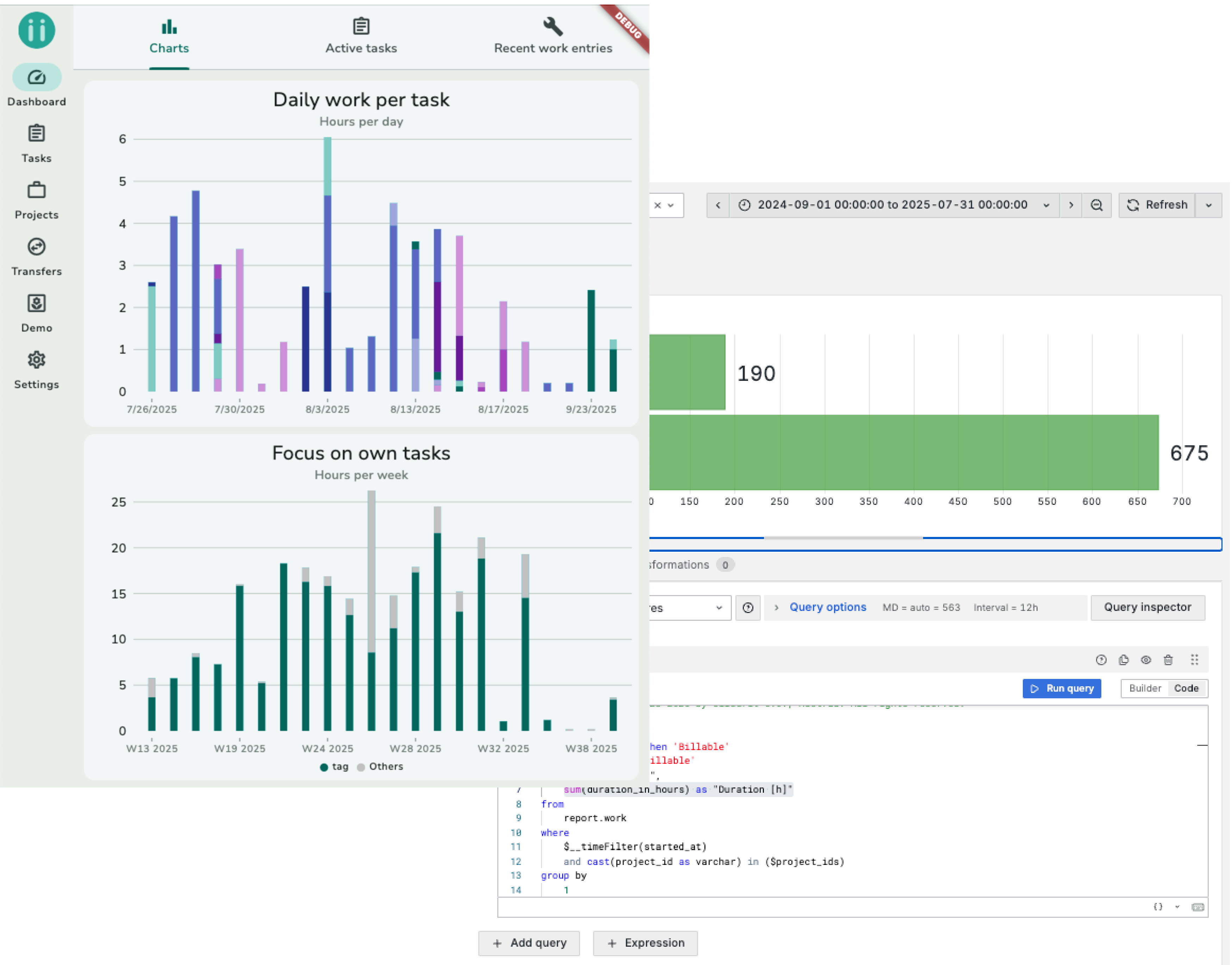
Common concerns

- "Unrestricted access for everyone"
- "Sensitive data end up in remote IT centers and can be accessed by random people we have no control over."
- "AI confabulates"

Make it easy to validate the claims of the AI

Make it easy to validate the claims of the AI

- Layer 1: Dashboards and standard reports
- Layer 2: BI tools



```
hen 'Billable'
illable'
",
    sum(duration_in_hours) as "Duration [h]"
from
    report.work
where
    $__timeFilter(started_at)
    and cast(project_id as varchar) in ($project_ids)
group by
    1
```

How does it work?

RAG

- RAG: See (German) talk at Grazer Linuxtage 2024 about "Wolkenlose KI für eigene Dokumente zu Hause": https://pretalx.linuxtage.at/glt25/talk/FJKZKN/

Django MCP server

The good

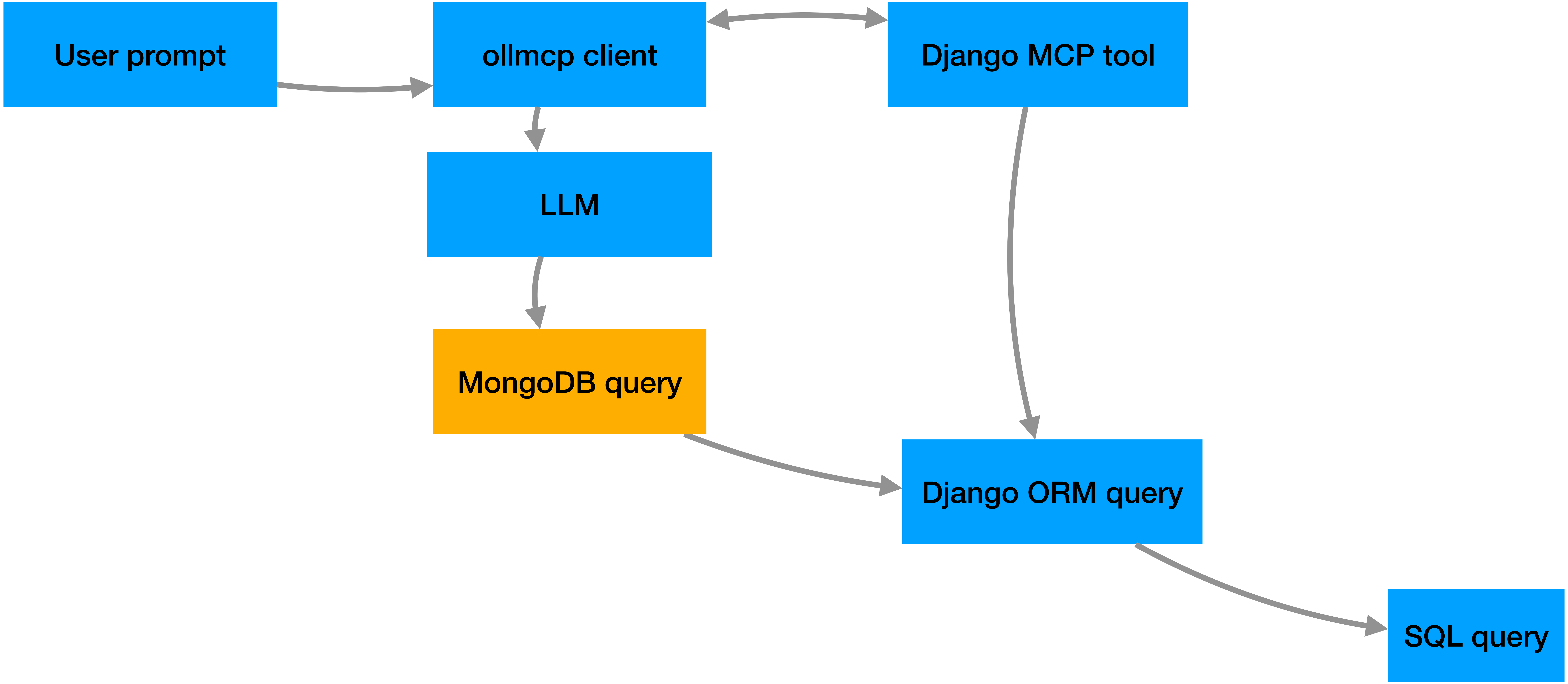
- Django ORM models can be exposed as MCP tools.
- Can utilize Django REST framework
  - permission checks
  - filters
- Very little new backend code needed

Django MCP server

The bad

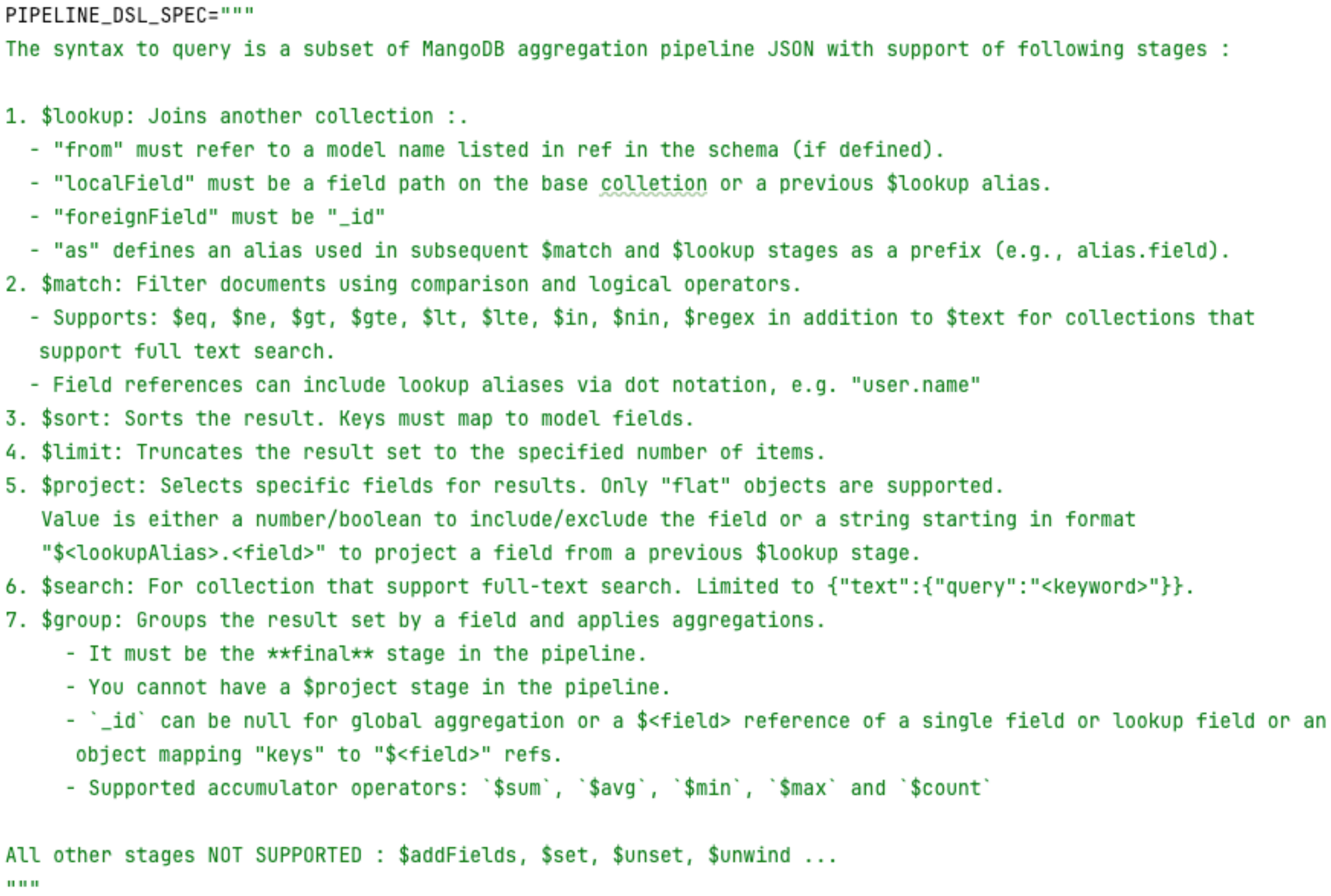
Django MCP server

The ugly: MongoDB queries as intermediate layer



Django MCP server

From https://github.com/omarbenhamid/django-mcp-server/blob/cc1e6ef3cc405ef27d854e501280bc808dbb56c7/mcp_server/query_tool.py#L125



```
PIPELINE_DSL_SPEC="""
The syntax to query is a subset of MangoDB aggregation pipeline JSON with support of following stages :

1. $lookup: Joins another collection :.
  - "from" must refer to a model name listed in ref in the schema (if defined).
  - "localField" must be a field path on the base colletion or a previous $lookup alias.
  - "foreignField" must be "_id"
  - "as" defines an alias used in subsequent $match and $lookup stages as a prefix (e.g., alias.field).
2. $match: Filter documents using comparison and logical operators.
  - Supports: $eq, $ne, $gt, $gte, $lt, $lte, $in, $nin, $regex in addition to $text for collections that
   support full text search.
  - Field references can include lookup aliases via dot notation, e.g. "user.name"
3. $sort: Sorts the result. Keys must map to model fields.
4. $limit: Truncates the result set to the specified number of items.
5. $project: Selects specific fields for results. Only "flat" objects are supported.
   Value is either a number/boolean to include/exclude the field or a string starting in format
   "$<lookupAlias>.<field>" to project a field from a previous $lookup stage.
6. $search: For collection that support full-text search. Limited to {"text":{"query":"<keyword>"}}.
7. $group: Groups the result set by a field and applies aggregations.
     - It must be the **final** stage in the pipeline.
     - You cannot have a $project stage in the pipeline.
     - `_id` can be null for global aggregation or a $<field> reference of a single field or lookup field or an
      object mapping "keys" to "$<field>" refs.
     - Supported accumulator operators: `$sum`, `$avg`, `$min`, `$max` and `$count`

All other stages NOT SUPPORTED : $addFields, $set, $unset, $unwind ...
"""
```

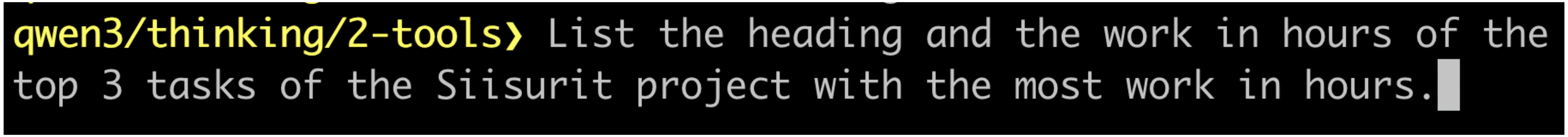
Does is actually work?

Does it actually work?

- 🤩 PostgreSQL with pgvector
- 😀 Ollama
- 🧐 Ollama models for mcp: qwen3:8b and llama3.1 work, most don't
- 🤨 ollmcp client - Works fine when it works, but hard to trace errors
- 🤪 django-mcp-server - Sometimes works, sometimes doesn't

Ideally...

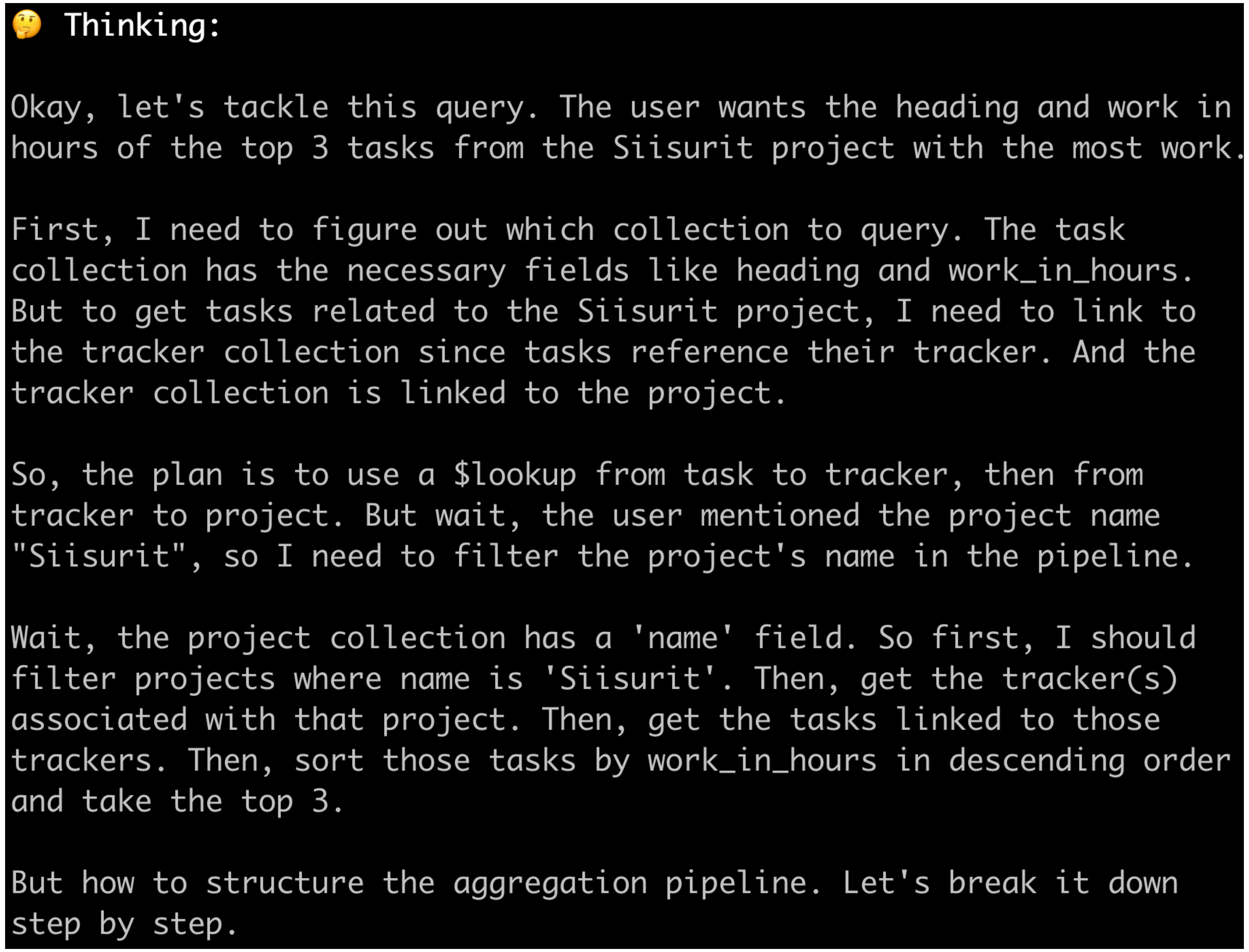
Enter project related query



```
qwen3/thinking/2-tools❯ List the heading and the work in hours of the
top 3 tasks of the Siisurit project with the most work in hours.
```

Ideally...

MCP returns MongoDB query



🤔 Thinking:

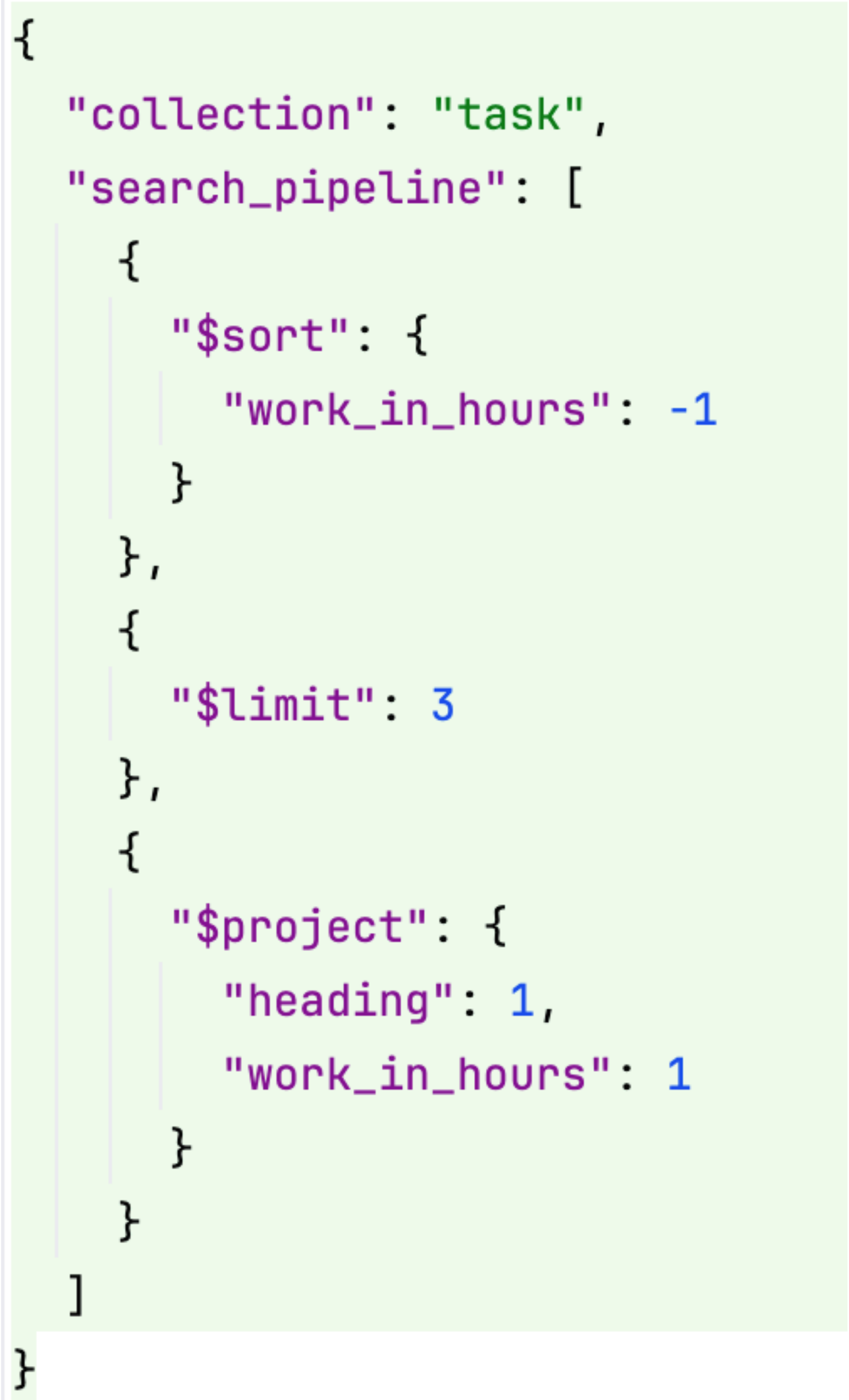
Okay, let's tackle this query. The user wants the heading and work in hours of the top 3 tasks from the Siisurit project with the most work.

First, I need to figure out which collection to query. The task collection has the necessary fields like heading and work_in_hours. But to get tasks related to the Siisurit project, I need to link to the tracker collection since tasks reference their tracker. And the tracker collection is linked to the project.

So, the plan is to use a $lookup from task to tracker, then from tracker to project. But wait, the user mentioned the project name "Siisurit", so I need to filter the project's name in the pipeline.

Wait, the project collection has a 'name' field. So first, I should filter projects where name is 'Siisurit'. Then, get the tracker(s) associated with that project. Then, get the tasks linked to those trackers. Then, sort those tasks by work_in_hours in descending order and take the top 3.

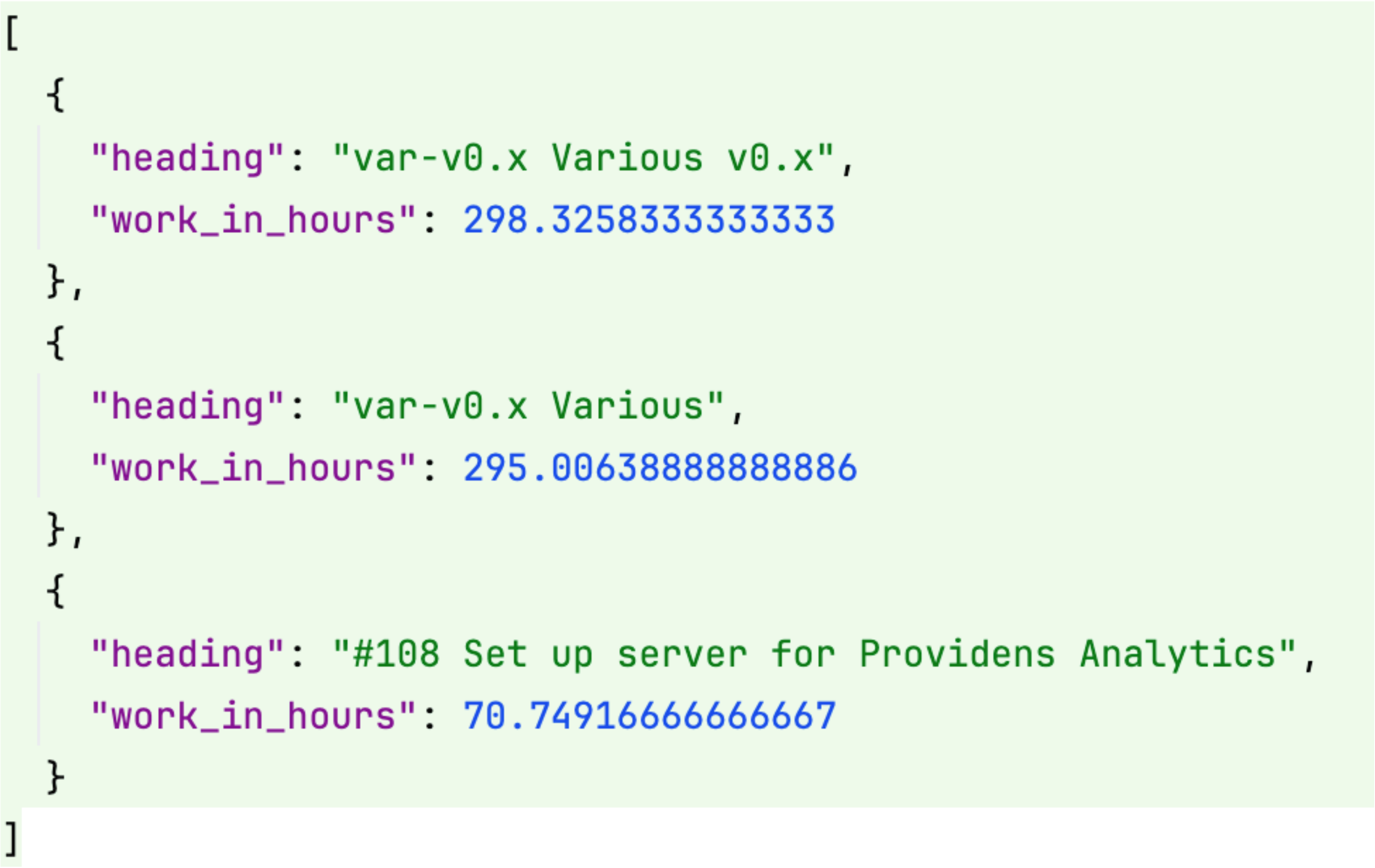
But how to structure the aggregation pipeline. Let's break it down step by step.



```
{
  "collection": "task",
  "search_pipeline": [
    {
      "$sort": {
        "work_in_hours": -1
      }
    },
    {
      "$limit": 3
    },
    {
      "$project": {
        "heading": 1,
        "work_in_hours": 1
      }
    }
  ]
}
```

Ideally...

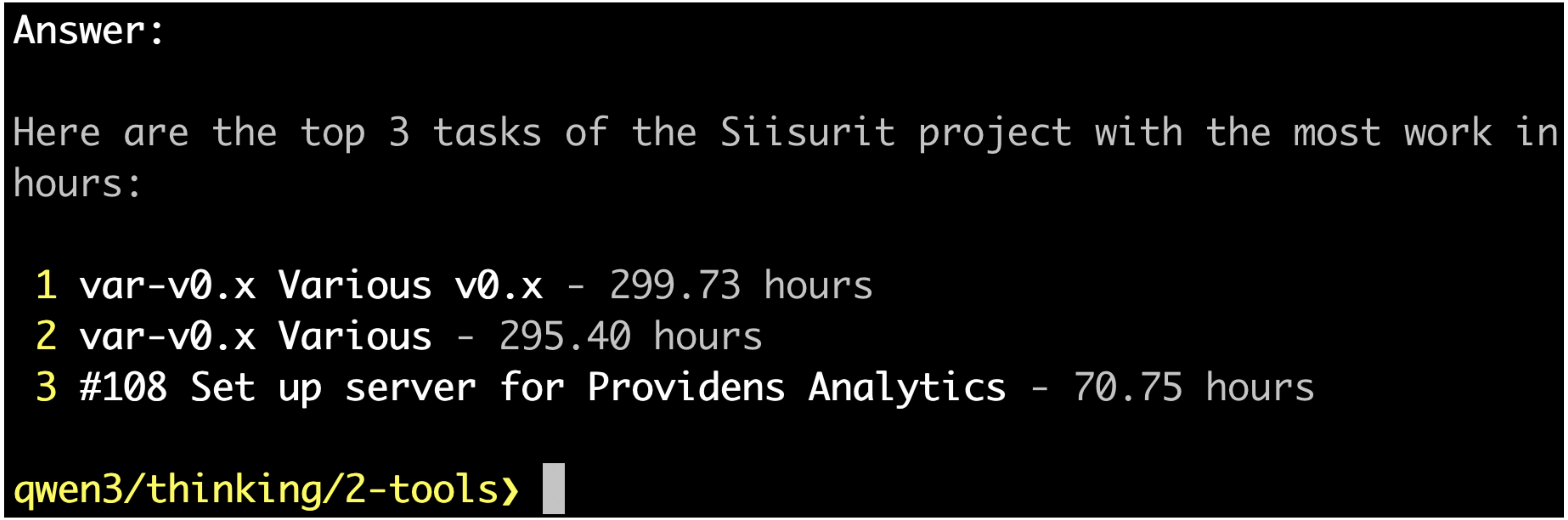
Django MCP converts MongDB into ORM query and returns result



```
[
  {
    "heading": "var-v0.x Various v0.x",
    "work_in_hours": 298.3258333333333
  },
  {
    "heading": "var-v0.x Various",
    "work_in_hours": 295.00638888888886
  },
  {
    "heading": "#108 Set up server for Providens Analytics",
    "work_in_hours": 70.74916666666667
  }
]
```

Ideally...

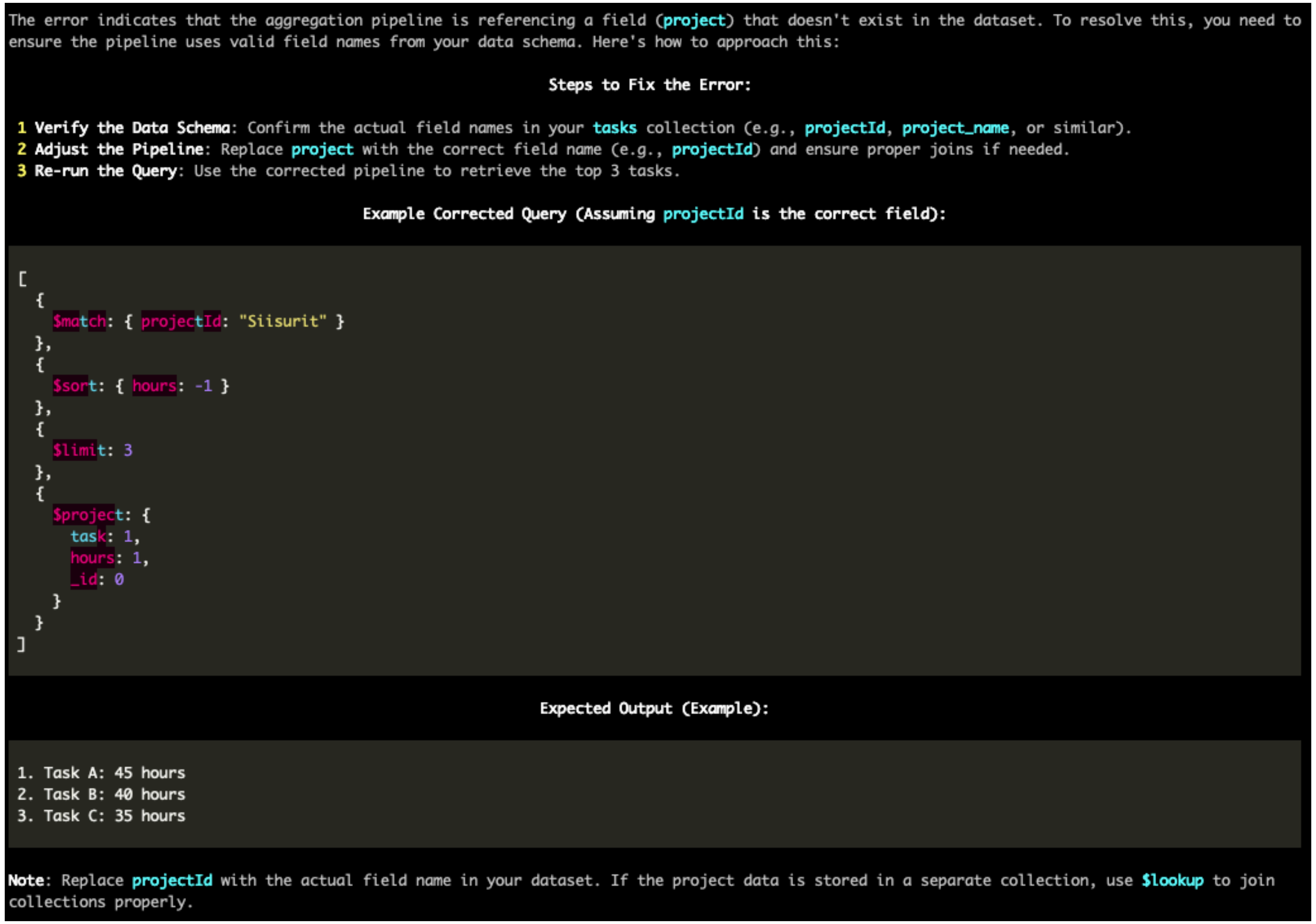
LLM translates JSON result into English sentence



```
Answer:

Here are the top 3 tasks of the Siisurit project with the most work in
hours:

 1 var-v0.x Various v0.x - 299.73 hours
 2 var-v0.x Various - 295.40 hours
 3 #108 Set up server for Providens Analytics - 70.75 hours

qwen3/thinking/2-tools❯
```

Sometimes...

LLM cannot resolve indirect relations etc etc etc



The error indicates that the aggregation pipeline is referencing a field (**project**) that doesn't exist in the dataset. To resolve this, you need to ensure the pipeline uses valid field names from your data schema. Here's how to approach this:

Steps to Fix the Error:

1. **Verify the Data Schema**: Confirm the actual field names in your **tasks** collection (e.g., **projectId**, **project_name**, or similar).
2. **Adjust the Pipeline**: Replace **project** with the correct field name (e.g., **projectId**) and ensure proper joins if needed.
3. **Re-run the Query**: Use the corrected pipeline to retrieve the top 3 tasks.

Example Corrected Query (Assuming projectId is the correct field):

```
[
  {
    $match: { projectId: "Siisurit" }
  },
  {
    $sort: { hours: -1 }
  },
  {
    $limit: 3
  },
  {
    $project: {
      task: 1,
      hours: 1,
      _id: 0
    }
  }
]
```

Expected Output (Example):

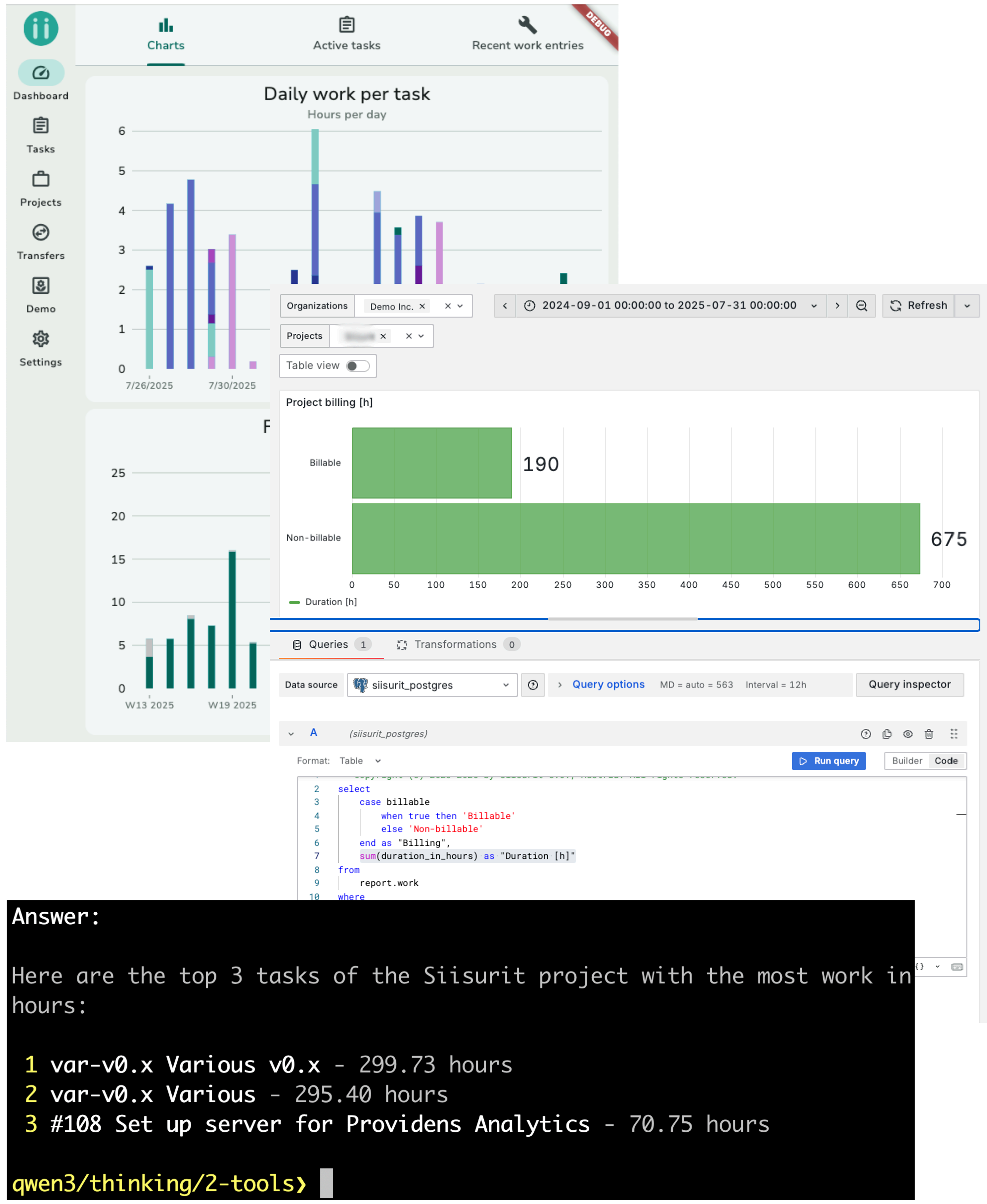
```
1. Task A: 45 hours
2. Task B: 40 hours
3. Task C: 35 hours
```

Note: Replace **projectId** with the actual field name in your dataset. If the project data is stored in a separate collection, use **$lookup** to join collections properly.

Conclusion

Conclusion

- Project management data can be processed using LLM
  - taking permissions into account
  - remaining inside your own IT infrastructure (on-premise)
- LLM can access the same data as dashboards and BI, possible confabulations can be checked comparably quickly
- MCP still is a moving target → still researching and prototyping stable solutions
- It was fun to prototype and learn new things.



```sql
select
    case billable
        when true then 'Billable'
        else 'Non-billable'
    end as "Billing",
    sum(duration_in_hours) as "Duration [h]"
from
    report.work
where
```

```
Answer:

Here are the top 3 tasks of the Siisurit project with the most work in
hours:

 1 var-v0.x Various v0.x - 299.73 hours
 2 var-v0.x Various - 295.40 hours
 3 #108 Set up server for Providens Analytics - 70.75 hours

qwen3/thinking/2-tools❯
```

siisurit.com

providens.at