KMAP
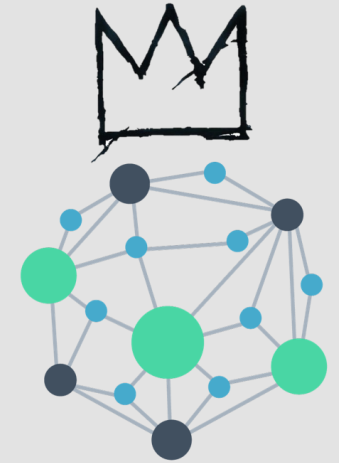# Technical overview

Johannes Jablonski
Tobias Kopyto
Igor Shmelev (ret.)
Cato Trütschel
Jule van de Logt
Thomas Wehr
Yannick Zuber

Knowledge is King

KMAP – Technical overview

- High-level component overview
- Database
- Backend
- Frontend
- Query archetypes

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
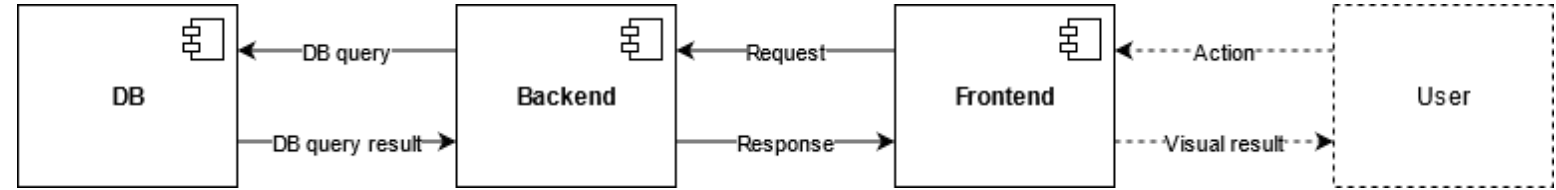
High-level component overview

High-level component overview

- Three-tire architecture
  - Database
  - Backend (incl. Data-access layer)
  - Frontend

- Database is not publicly accessible

- Database can stay (logically) external
  - Use existing databases
  - No public ally available HTTP end-point

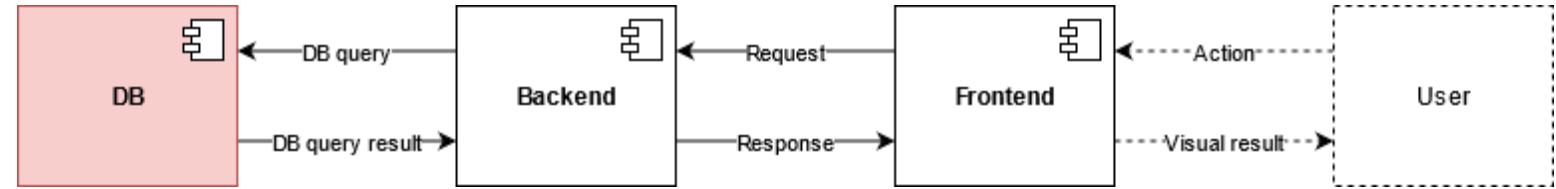- Backend can pre-process request and post-process responses

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Database

Database

- Requirements
  - Existing or new graph database
  - No write access required
  - Schema and data should not change currently due to caching (WIP)
  - DB system is pluggable by replacing DAL in backend

- Currently used DBMS is neo4j
  - Minimal configuration
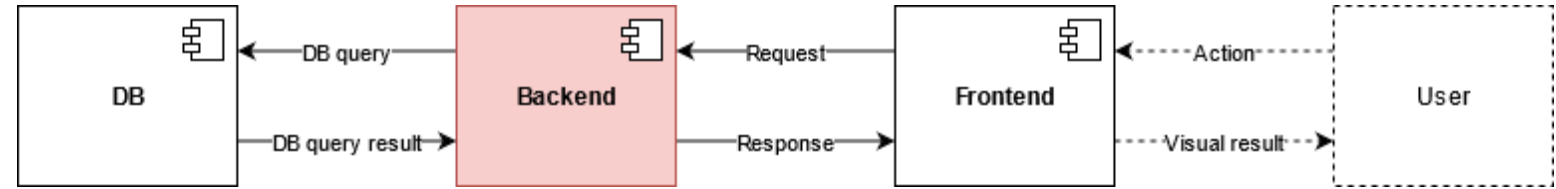 - APOC and GDS plugins needed (fallbacks are WIP)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend

Backend

- Provide domain specific data-service
  - HTTP API via a webserver for web-clients (frontend)
  - Process domain specific queries from frontend
  - Formulate DB-Queries, execute them via the DB instance
  - Post-processes query results

- Additional application-wide services
  - Data-indexing
  - Caching
  - Data-projection

- Technologies and frameworks
  - Node.js https://nodejs.org/en/
  - NestJS https://nestjs.com/

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Components
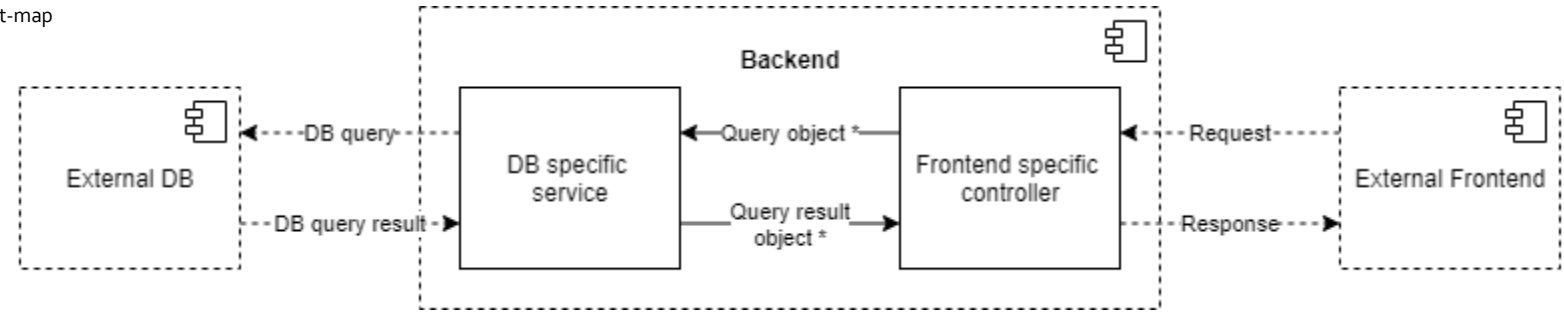
- Internal *Backend* components
  - DB specific services
  - Frontend specific controllers

- Reason for split
  - Adaption to two separate and distinct actors (SRP)
  - Support of testability via component tests

- Domain-specific *query* objects and *query-result* objects

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Controller
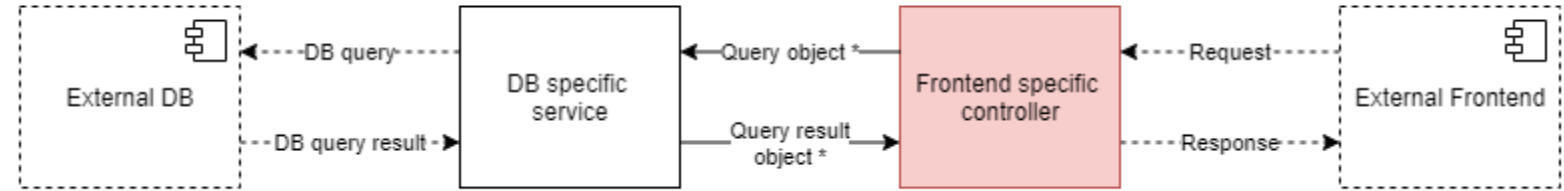
- Responsibility
  - Error-handling and validation of requests
  - Construction and augmentation of *request* objects
  - Ensure backwards-compatibility and support future API changes (via translation)

- NestJS controllers

- Makes available HTTP API to web-clients (frontend)

- API adapted to and specialized for frontend needs

Backend – Service (1)
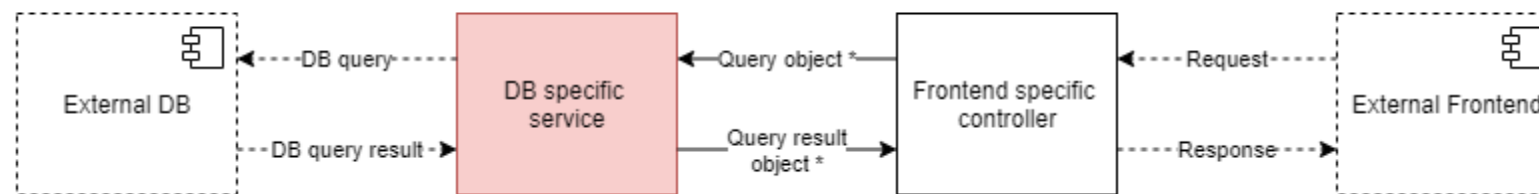
- Responsibility
  - Process domain specific request (*query* objects) and deliver self-contained response (*query-result* object)

- Injected, composable services

- Adapted to and specialized for the used database

- Includes the data-access layer (DAL)

- Support for other DBMS then neo4j
  - DB dedicated implementation of service interface for target DBMS
  - Replacement of injected service in service composition

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
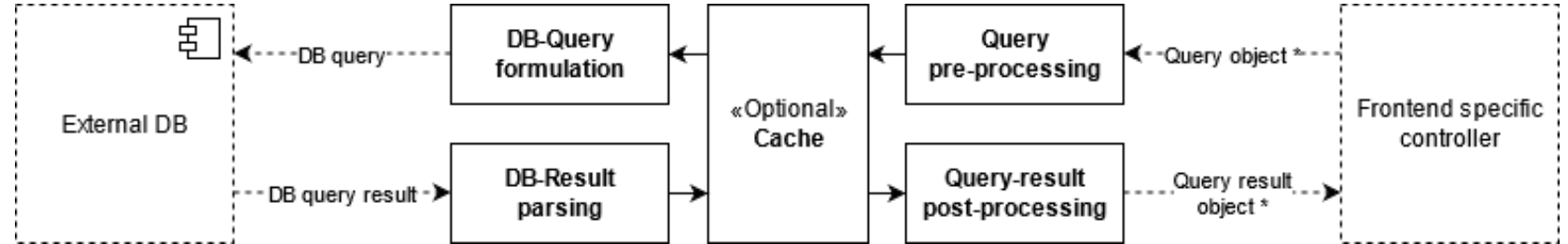
Backend – Service (2)

- Detailed structural design of a backend service
- Dataflow
  - Query object preprocessing
  - DB-Query formulation
  - DB-Query execution (external)
  - DB-Result parsing
  - Query-result post-processing
- Optional caching
- Other specialized services

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Query pre-processing
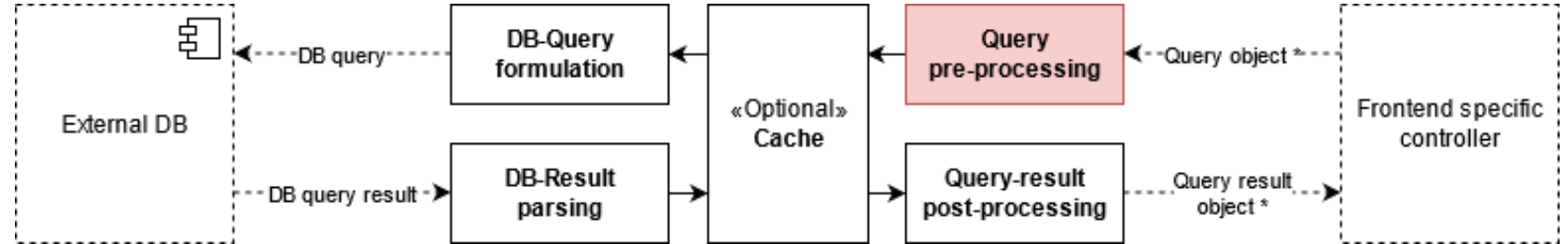
- Pre-processing of *query* objects
- Translation into internal (cacheable) format
- Preparation of query-formulation
- Example
  - Split into separate requests of nodes and edges
  - Split into two subqueries that are executed and the results combined

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – DB-Query formulation
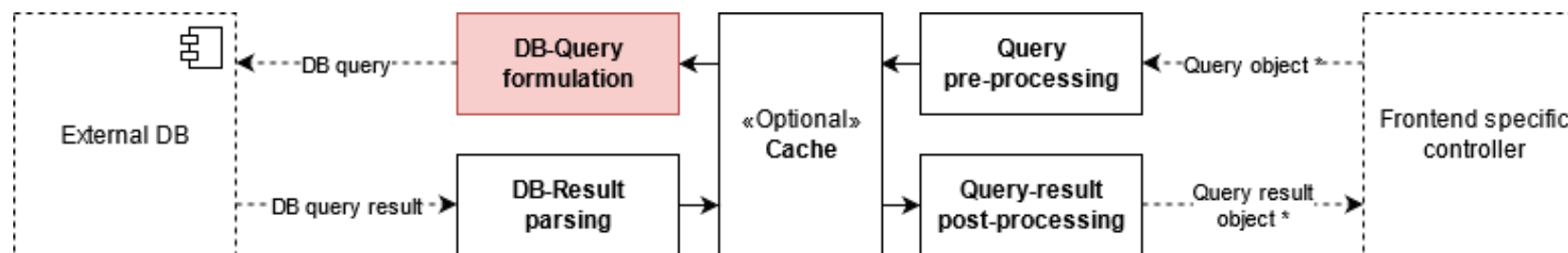
- Formulation of DB-query

- Using the DB specific query language (cypher for neo4j)

- Setting parameters from the *query* objects (or its internal representation)

- Construction of (rather) dynamic queries
  - Translation of domain specific query-language
  - Deep-first tree traversal
  - Used for filter-queries (see *query archetypes*)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – DB-Result parsing
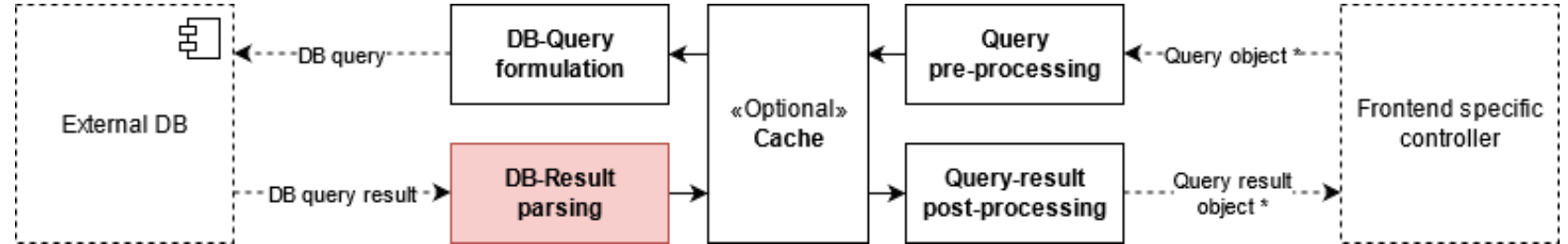
- Translation of DB query results to *query-result* objects (or its internal format)
- DB-Errors processing
  - Failure
  - No result

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Query-Result post-processing (1)
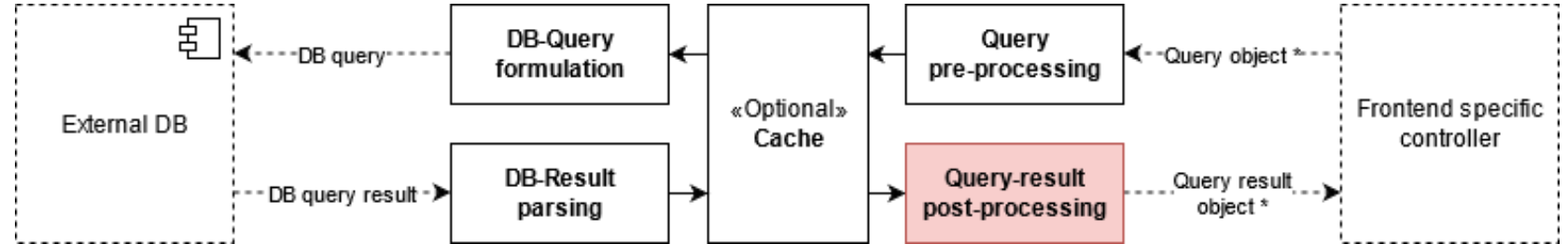
- Post-processing of *query-result* objects

- Translation from the internal format (if required)

- Combination of results
  - If additional services were used for execution
  - If query pre-processing split request into multiple parts

- Query-result object consolidation
  - Removing duplicate entries
  - Handle subsidiary nodes

Backend – Query-Result post-processing (2)
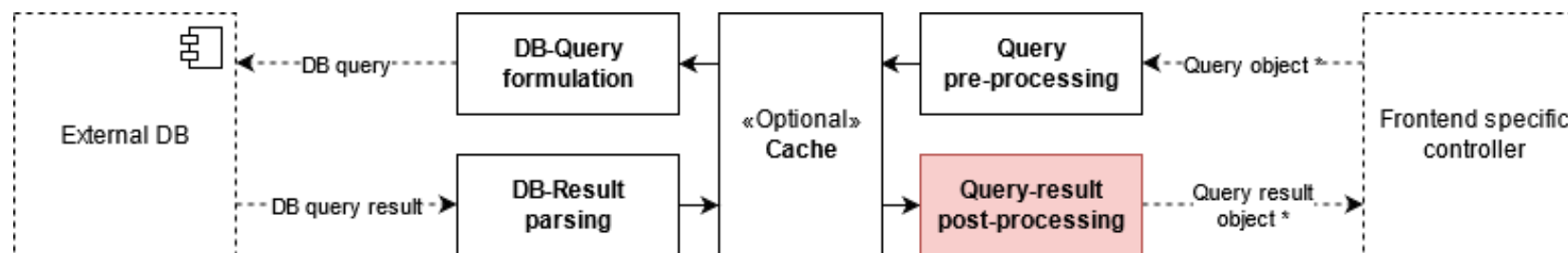
- Adding values from cache if present

- Example
  - Combine edge and node query result to a self-container query-result
  - Combine filter-result as received from external service with special query (See shortest-path query)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Optional Caching
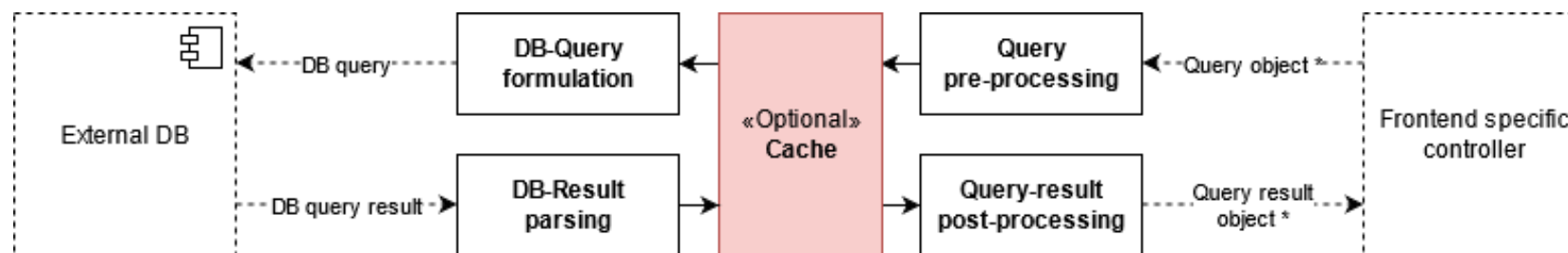
- Optional query-result caching
- Cache complete query-result for given key
- Cache partial query-results
  - Query pre-processing removes all query arguments that cache contains result for
  - Query-result post-processing combines cached results with results from database
- Cache is currently not updates and assumes a read-only database (WIP)
- Example
  - Caching of data schema
  - Search-index

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Specialized services
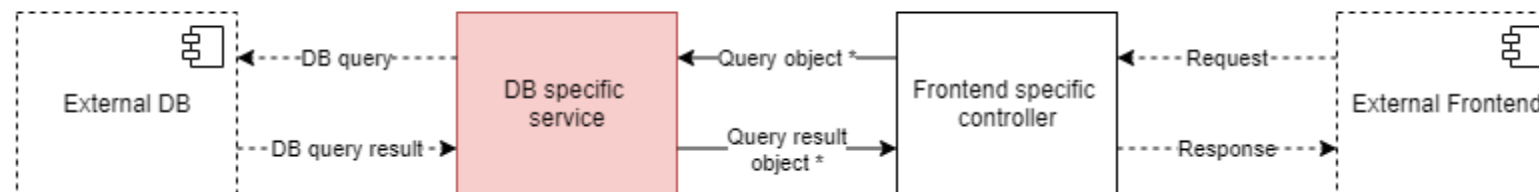
- Specialized domain-specific services
- Cannot directly be represented with the service-pipeline
- Services are application global
- Examples
  - Projection of data-set into domain specific format for further processing
  - Indexing the dataset for fast full-text lookup

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Backend – Composability
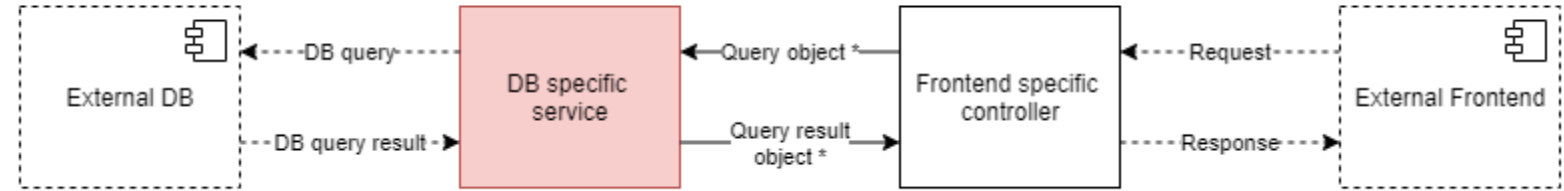
- Backend services are composable

- Services are injected via DI system

- Can built up on each others functionality

- Example:
  - Shortest-path service uses filter-service and combines results
  - Search-service uses general query-service to build search-index

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend

Frontend
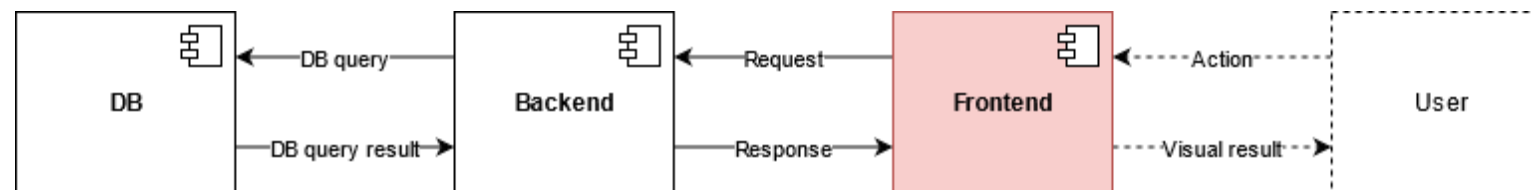
- Single page application (SPA) in the user's browser

- Displays graph and exploration results to user

- Processes action (request, queries, explorations) via HID

- Technologies and frameworks
  - Modern browser (Firefox, Chrome, Edge, Safari, etc.)
  - React https://reactjs.org/
  - Inversify https://inversify.io/
  - Material UI https://material-ui.com/
  - RxJS https://rxjs.dev/

Frontend – Components

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – Request service (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
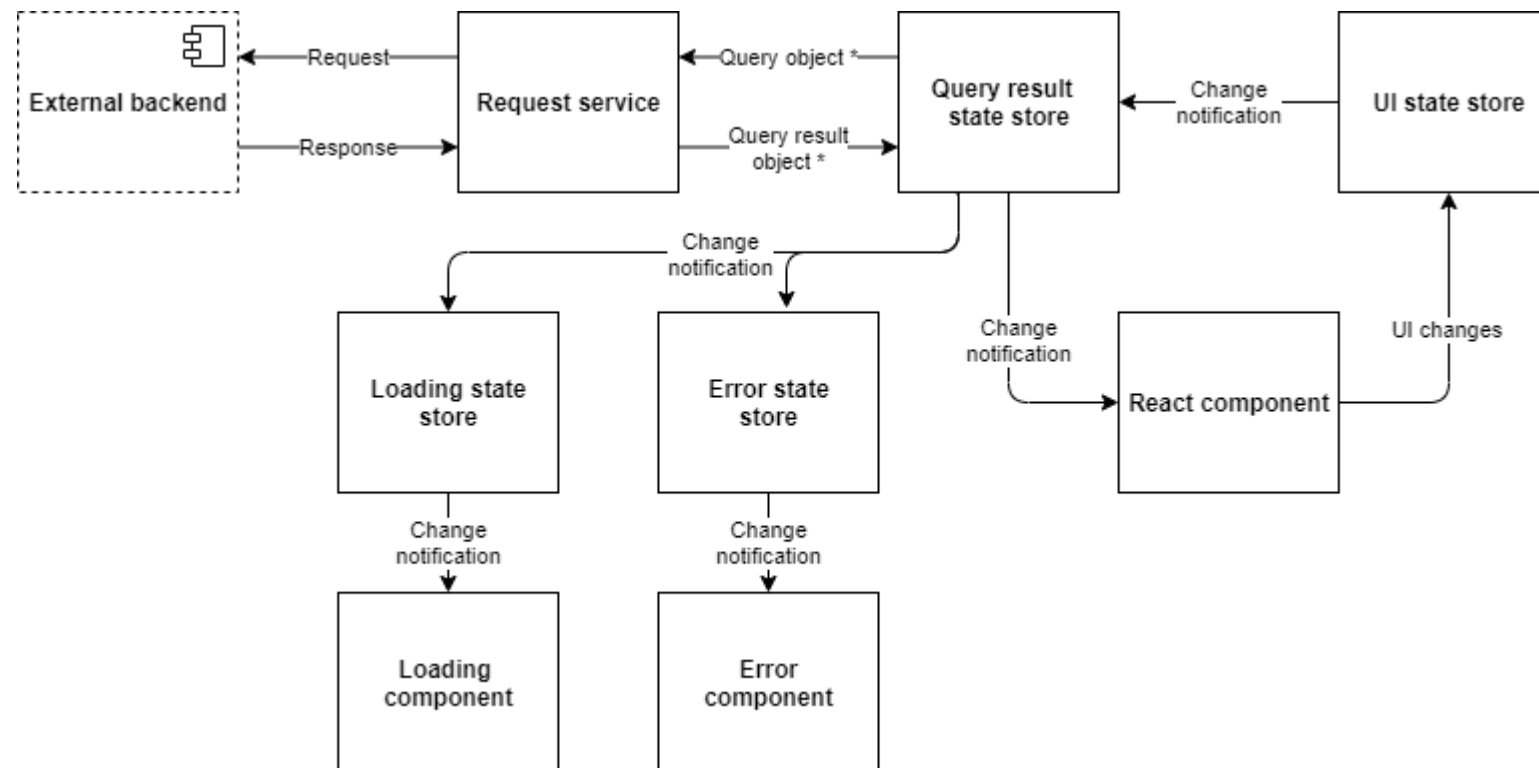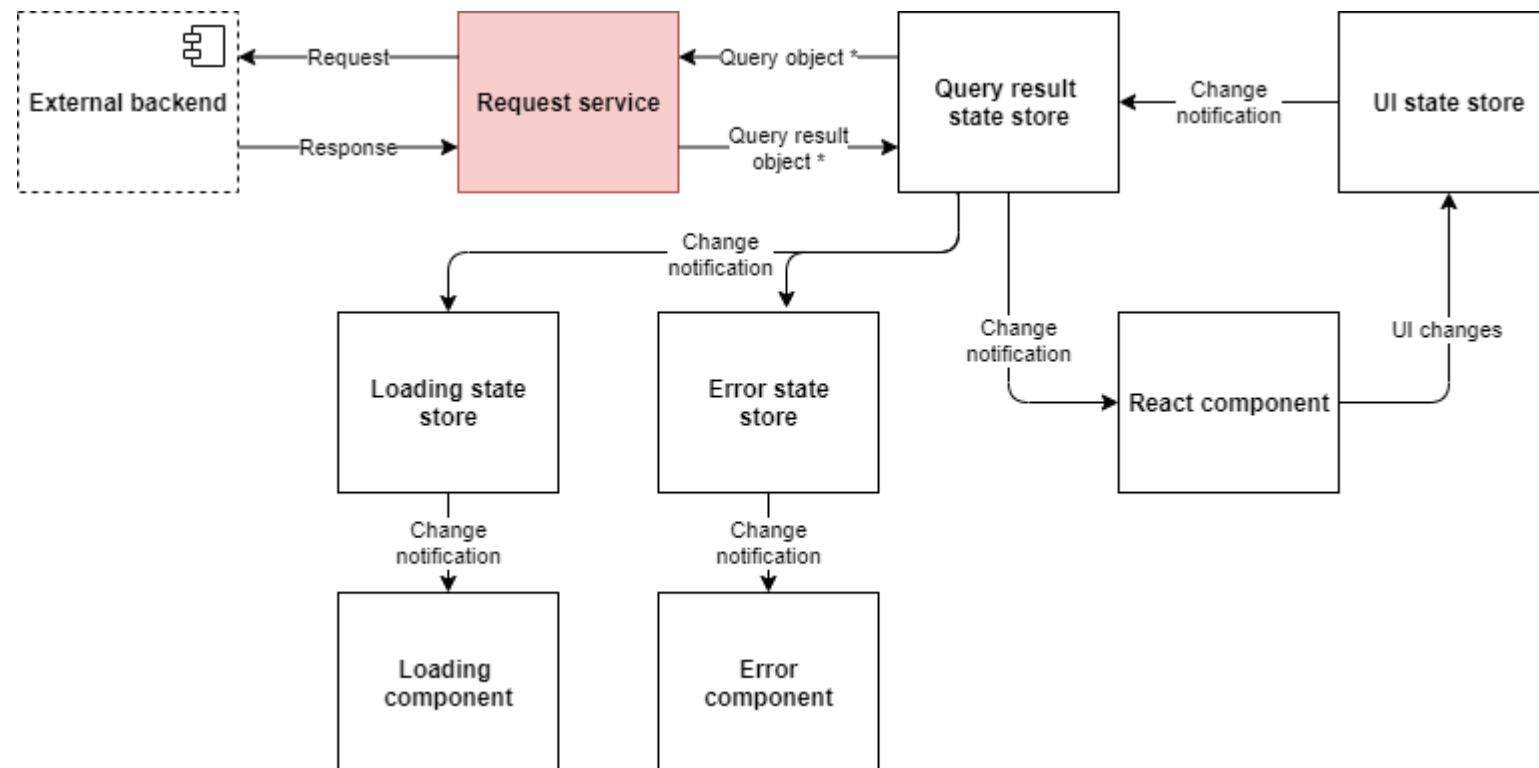
Frontend – Request service (2)

- Responsible for communication with backend

- Translates *query* objects from the app into HTTP calls

- Translates the HTTP Response from the backend into *query-result* objects

- Uses JSON serialization

- Error handling
  - Network errors
  - Bad requests
  - Internal server errors

- Uses asynchronous IO via Promises (no blocking IO)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – Request service (3)
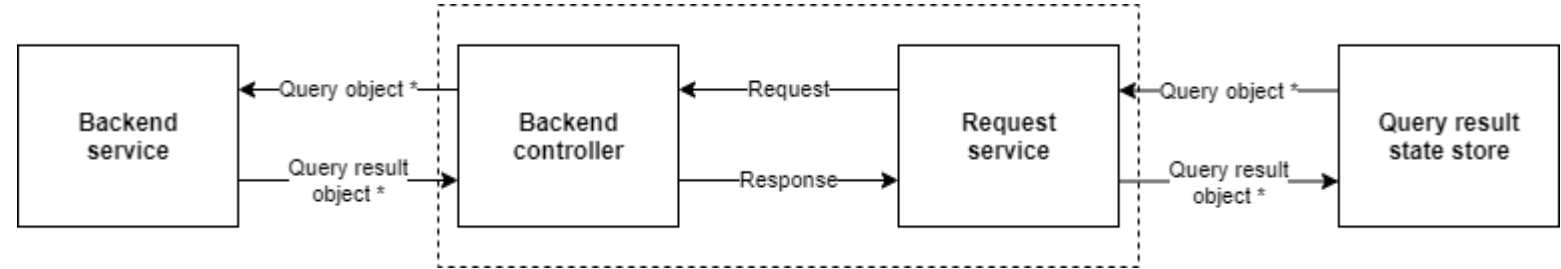
- Together with the respective backend controller
  - Domain specific asynchronous RPC infrastructure
  - *Query* objects are passed through to the backend
  - *Query-result* objects are passed back to the frontend

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – UI state store (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
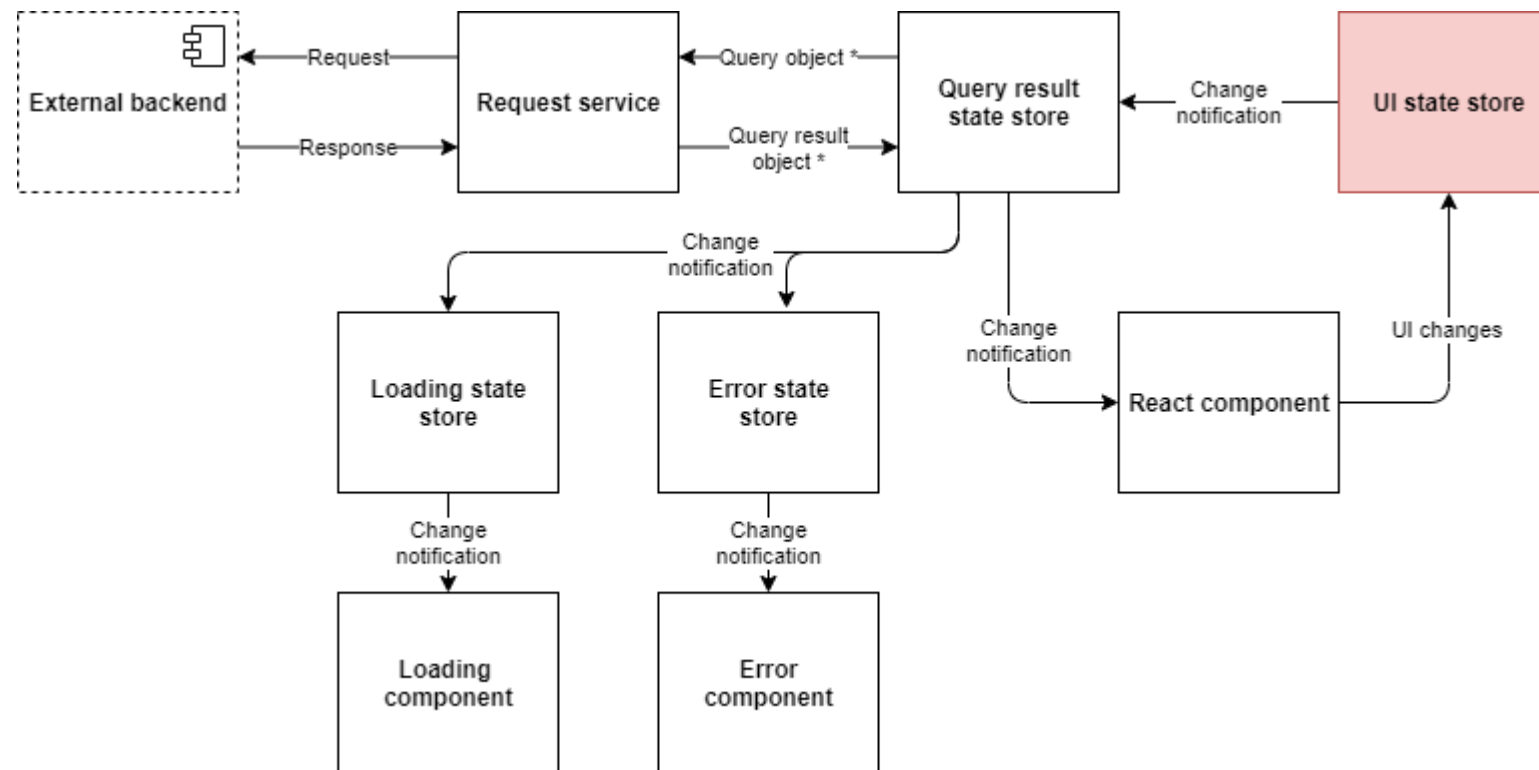
Frontend – UI state store (2)

- Stores the current state of the user-interface (UI)

- Decouples state from the presentation of the state

- Modularization of application

- Notifies other components (that are subscribed) about UI state changes

- UI state is preserved when displayed page is switched

- Possible future use
  - Store state in local-storage for persistence
  - Undo/Redo

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – Query result state store (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
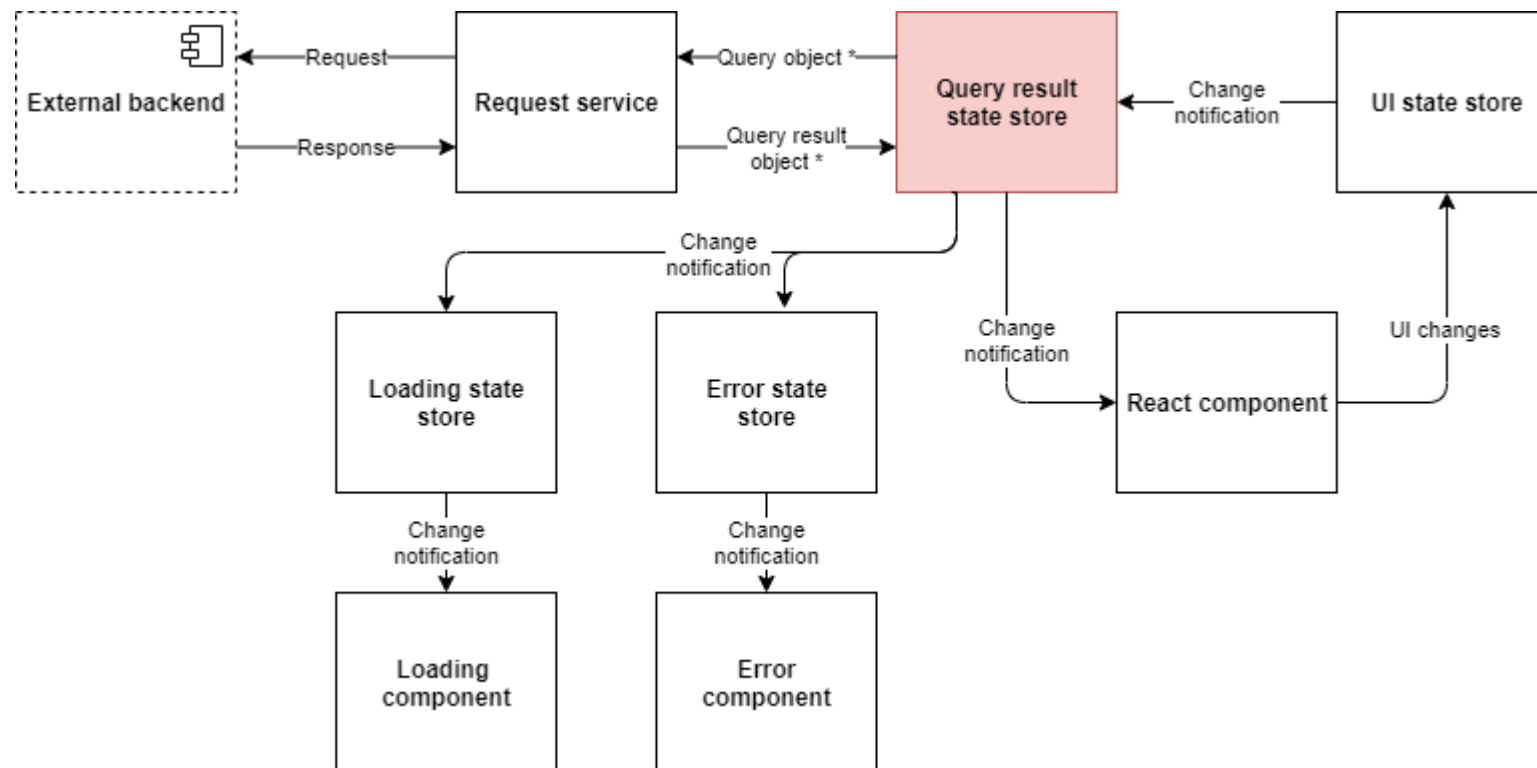
Frontend – Query result state store (2)

- Stores the result of the latest query-operation

- Updates the state when the *UI state-store* notifies about changes

- Builds query from the UI state

- Executes query via the *request-service*

- Notifies UI components about changes of state

- Handles also
  - Cancellation
  - Throttling
  - Loading
  - Errors

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – UI Component (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
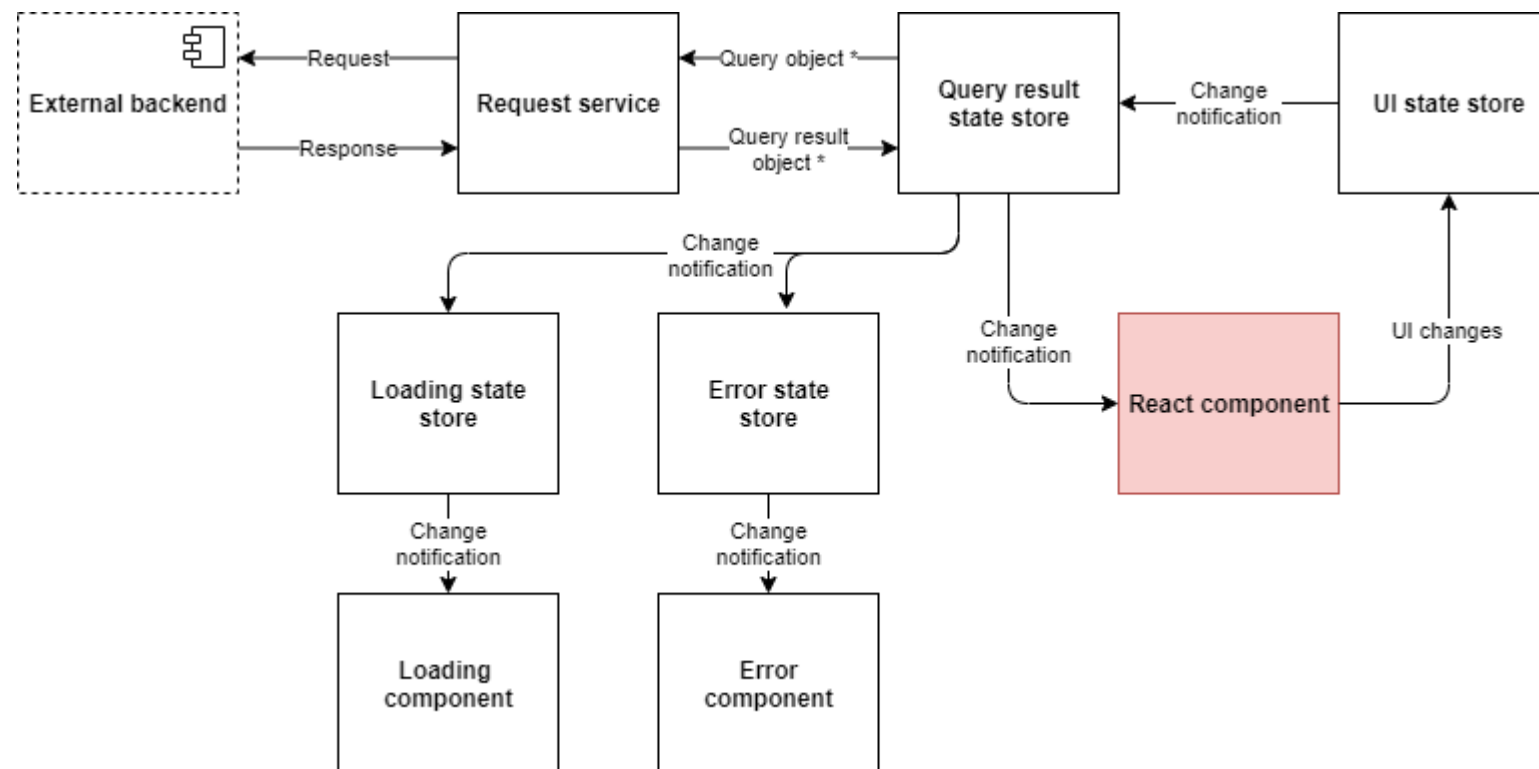
Frontend – UI Component (2)

- Display the state, as stored in the subscribed *UI state-store*

- Can also display state in secondary stores (like the *query-result state store*)

- When not a read-only view, allow state changes via the UI

- Update the *UI state-store* about updates in the UI

- *UI components* are implemented via React components

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – Loading (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
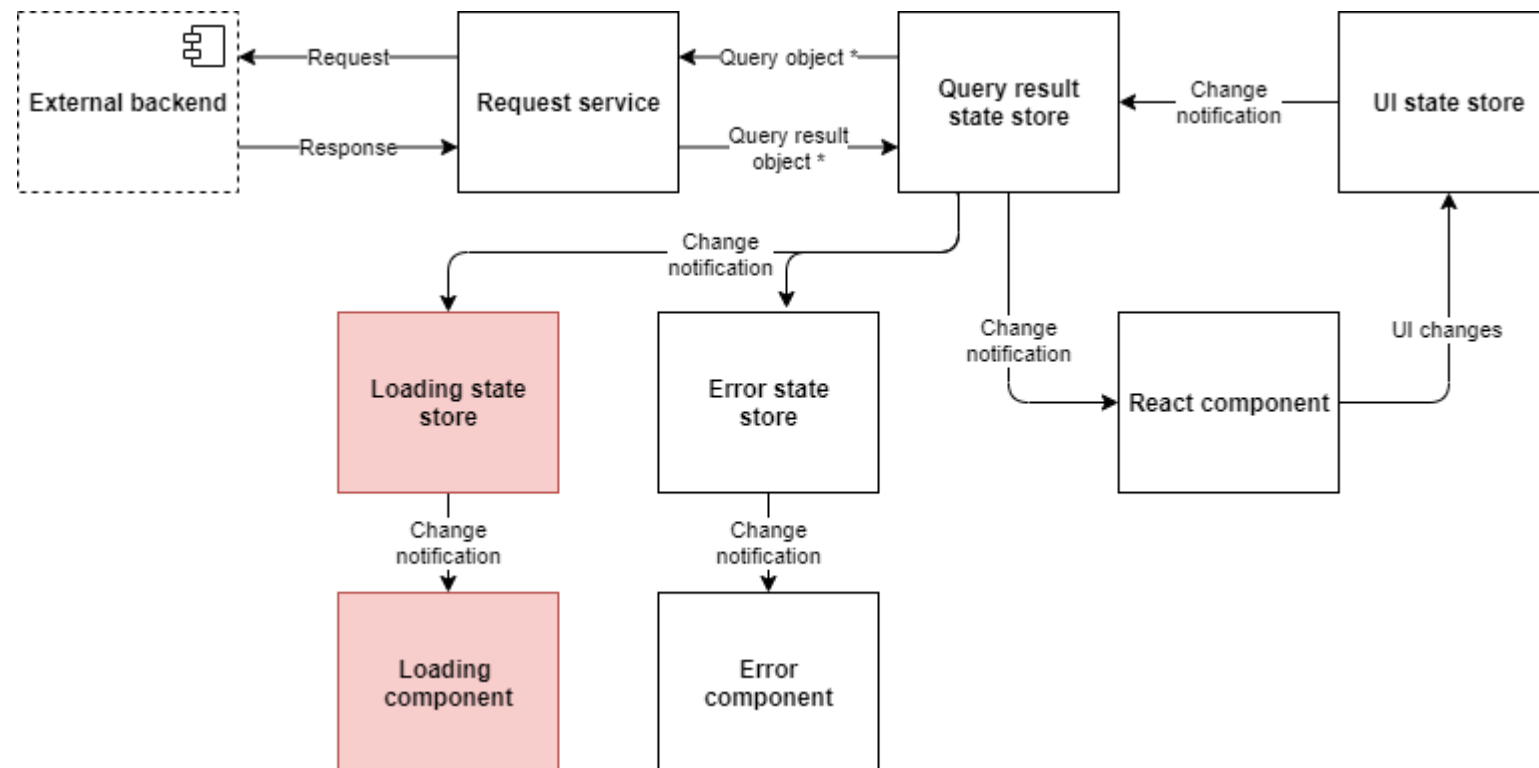
Frontend – Loading (2)

- A dedicated state store

- A dedicated *UI component*

- Store and display the loading operations in the current system

- Are notified of loading operations by other stores and services

- Display an option for cancellation that a loading operation can be canceled with

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Frontend – Error (1)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
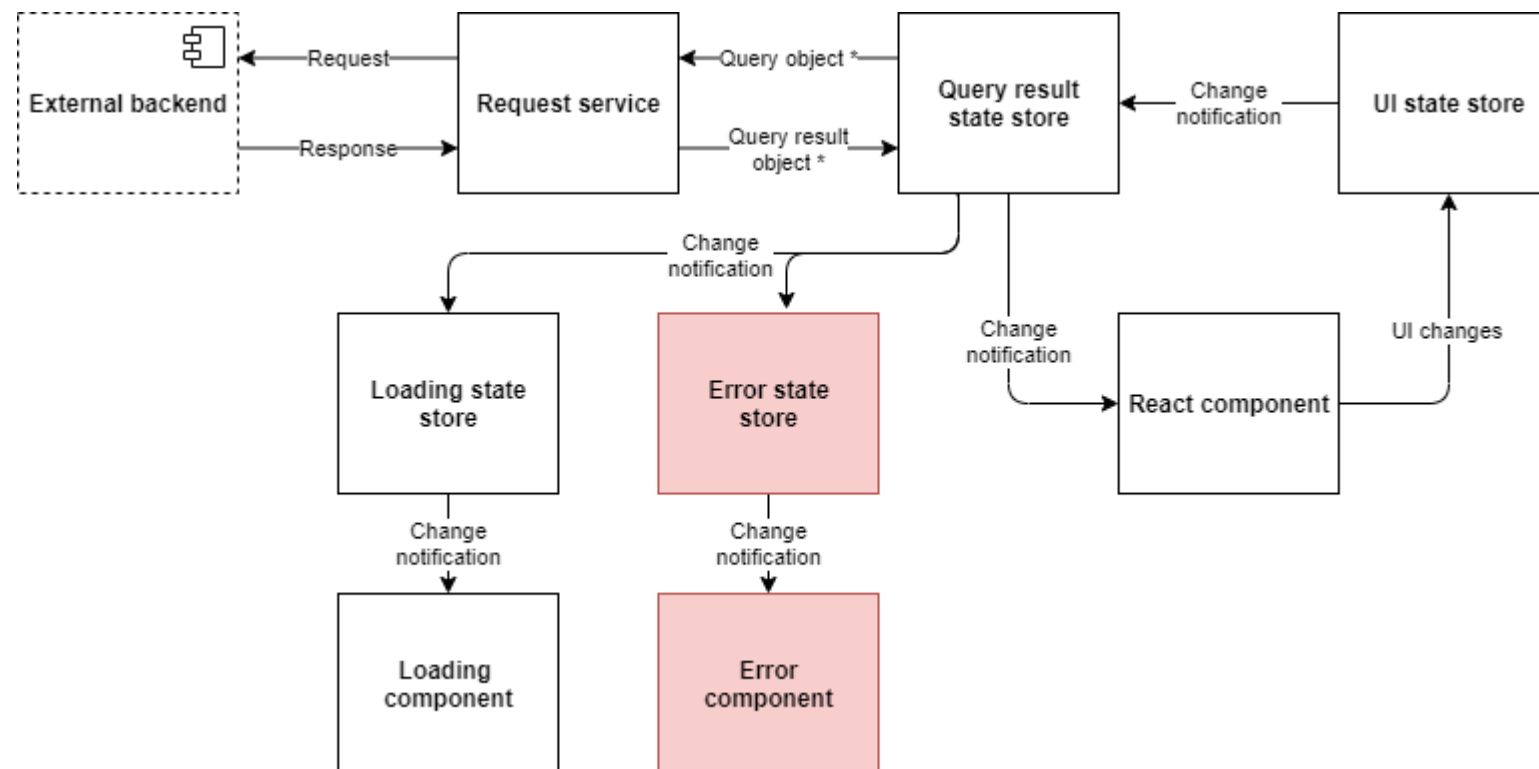
Frontend – Error (2)

- Dedicated state store

- Dedicated UI component

- Store and display the latest application error

- Error types
  - Cancelation errors
  - Network errors
  - Server errors
  - Page not found (HTTP 404) errors
  - etc.

- Notified by other services, state stores and the page router about errors

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Query archetypes

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Query archetypes

- Graph query
  - Filter
  - Shortest-path
  - etc.
- Schema query (without deep insight)
- Search query (without deep insight)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

KMAP https://github.com/amosproj/amos-ss2021-project2-context-map
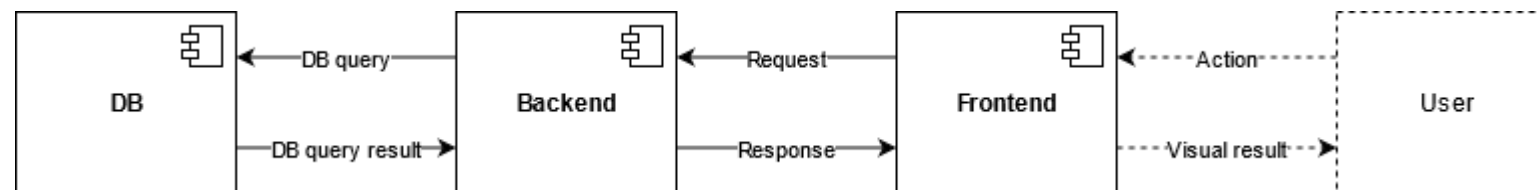
Graph query

- Basic structure of all graph-queries
- User specifies meta-parameter via action
  - Filter settings
  - Start and end-node of shortest-path search
  - Etc.
- Domain specific *query* object is constructed in the frontend
- Backend processes *query* object and sends back *query-result* object
- *Query-result* object is displayed in the frontend

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query – Query result object

- Common *QueryResult* type
- Visualizations do not need to be adapted to query-archetype (Mix & Match)
- Entity properties
  - Subsidiary
  - Virtual
  - IsPath
- Query result is guaranteed to be consistent (See backend query-result post-processing)

EdgeDescriptor
«interface»
+ id: number
+ type: string
+ from: number
+ to: number

NodeDescriptor
«interface»
+ id: number
+ types: string[]

QueryEdgeResult
«interface»
+ subsidiary: boolean?
+ virtual: number?
+ isPath: boolean?

QueryNodeResult
«interface»
+ subsidiary: boolean?
+ isPath: boolean?
+ virtual: number?

edges — nodes

QueryResult
«interface»

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

```
 ┌─────────────────────────┐                    ┌─────────────────────────┐
 │      QueryBase          │                    │      QueryLimits        │
 │     «interface»         │──limits?──────────▶│     «interface»         │
 ├─────────────────────────┤                    ├─────────────────────────┤
 │ + includeSubsidiary:    │                    │ + nodes: number?        │
 │   boolean?              │                    │                         │
 │                         │                    │ + edges: number?        │
 └─────────────────────────┘                    └─────────────────────────┘
```

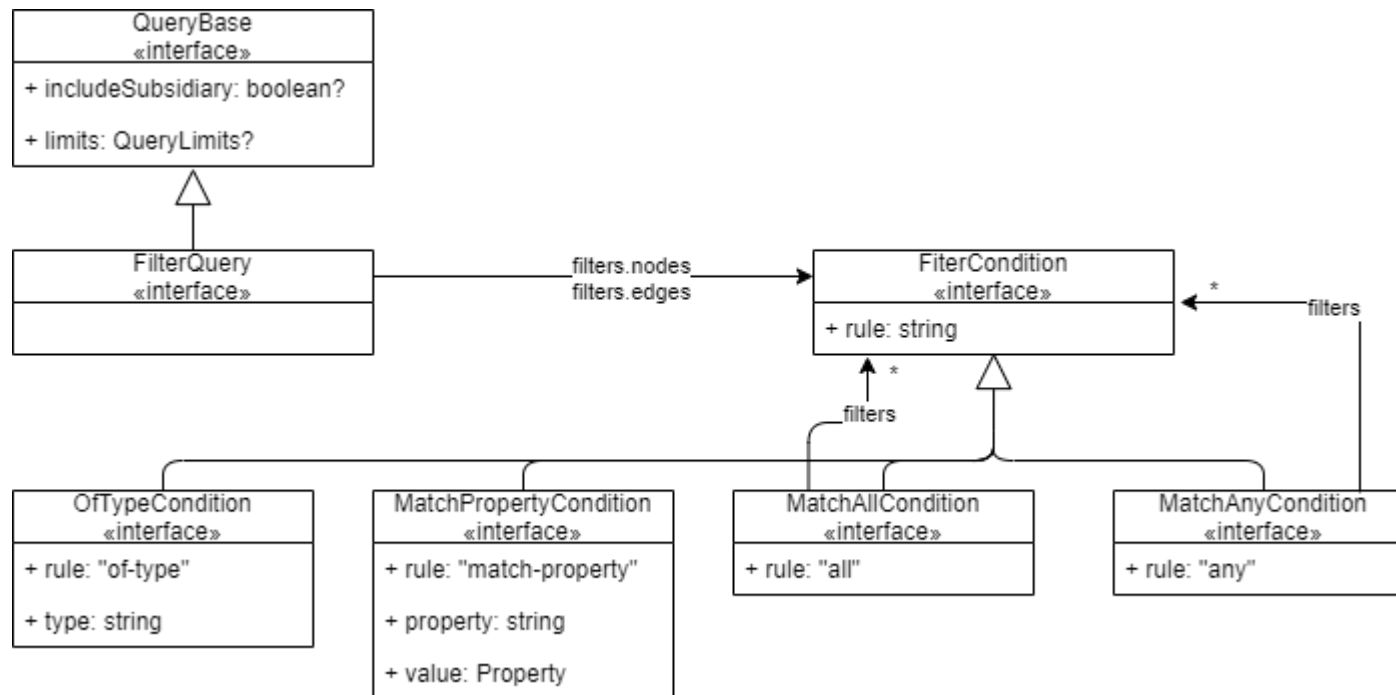# Graph query – Query object
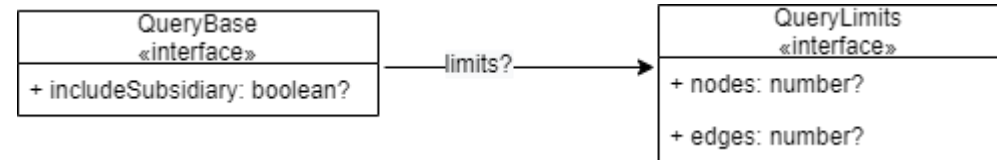
- QueryBase interface as a common subtype for all queries

- Subsidiary filter option

- Node and edge limit

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query – Filter query (1)

QueryBase
«interface»

+ includeSubsidiary: boolean?

+ limits: QueryLimits?

FilterQuery
«interface»

FiterCondition
«interface»

+ rule: string

filters.nodes
filters.edges

* filters

* filters

OfTypeCondition
«interface»

+ rule: "of-type"

+ type: string

MatchPropertyCondition
«interface»

+ rule: "match-property"

+ property: string

+ value: Property

MatchAllCondition
«interface»

+ rule: "all"

MatchAnyCondition
«interface»

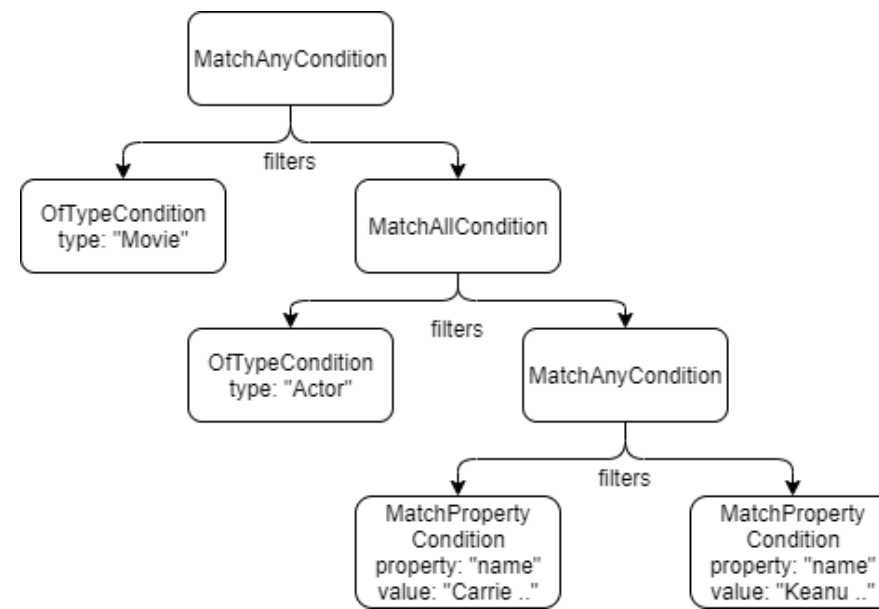+ rule: "any"

- Formulation of arbitrary filters (predicates)
- Extensible to additional predicates
- Translated to DB specific query via deep search tree-traversal

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query – Filter query (2)

- Edges are unfiltered

- Nodes are filtered

- Condition
  - All nodes of type "Movie"
  - All nodes of type "Actor" where the name matches any of
    - Keanu Reeves
    - Carrie-Ann Moss

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
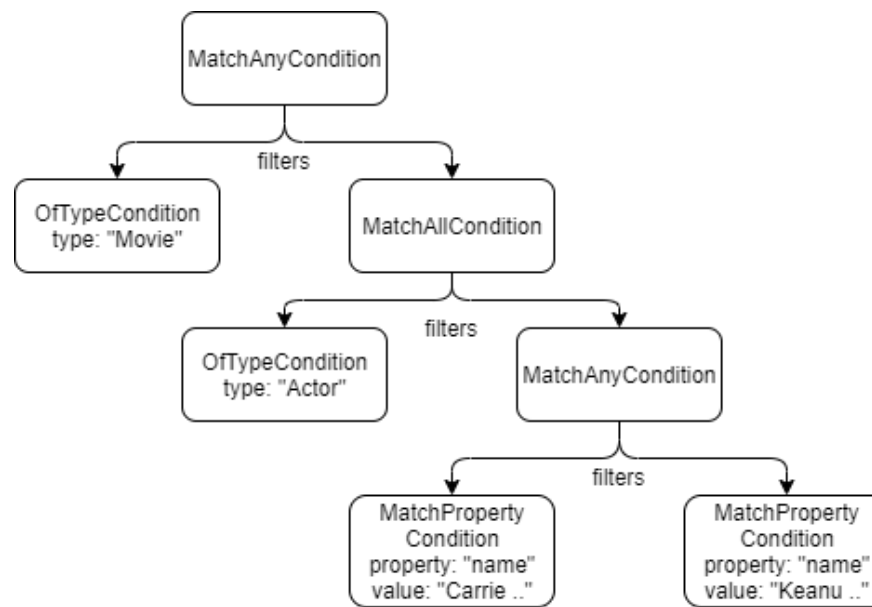
Graph query – Filter query (3)

```
1  {
2    "filters" : {
3      "nodes" : {
4        "rule" : "any",
5        "filters" : [{
6          "rule" : "of-type",
7          "type" : "Movie"
8        }, {
9          "rule" : "all",
10         "filters" : [{
11           "rule" : "of-type",
12           "type" : "Actor"
13         }, {
14           "rule" : "any",
15           "filters" : [{
16             "rule" : "match-property",
17             "property" : "name",
18             "value" : "Keanu Reeves"
19           }, {
20             "rule" : "match-property",
21             "property" : "name",
22             "value" : "Carrie-Anne Moss"
23           }
24         ]
25         }
26       ]
27       }
28     ]
29     }
30   }
31 }
```
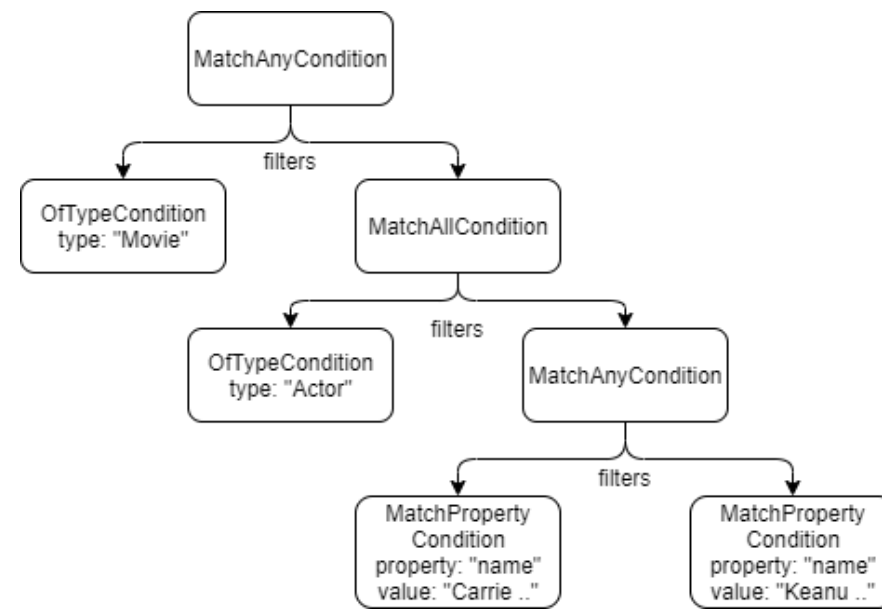
Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query – Filter query (4)

- Transformed to native cypher query via *ConditionVisitor* (performs deep-first tree traversal)


```
1    (
2      "Movie" in labels(n)
3      OR
4      (
5        "Actor" in labels(n)
6        AND
7        (
8          n["name"] = "Keanu Reeves"
9          OR
10         n["name"] = "Carrie-Anne Moss"
11       )
12     )
13   )
```

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Graph query – Shortest path query

- Extended *FilterQuery*

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber
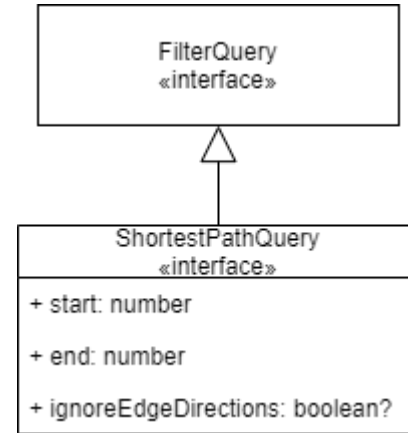
Graph query – Entity details

- *Query result* objects only contains most relevant information about edges

- How to access entity details (i.e. property values)

- Solution: Specialized query-service

- Cache entity details in the frontend (by their id)

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Schema query

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Schema query

- Allows to query the schema

- Node types present in the dataset and their properties

- Edge types present in the dataset and their properties

- Which edge-type connects which node types (WIP)

- Massive caching in the frontend

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Search query

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber

Search query

- The *query* object is a single string

- Possibility to enhance the query object in the future for further options
  - Search only in specific fields
  - Case-sensitivity
  - etc.

- Search result
  - Nodes
  - Edges
  - Node types
  - Edge types

Johannes Jablonski, Tobias Kopyto, Igor Shmelev (ret.), Cato Trütschel, Jule van de Logt, Thomas Wehr, Yannick Zuber