Lab5 Egress: Part 1 - REST vs GraphQL

Lorin Speybrouck

Remarks

When making the first task I found it frustrating that it was not obvious what a API route parameters where. For this purpose i implemented Type-safe routing with the ktor Resources plug in, this made it possible to define the name of parameters and query parameters and there type in a class. This is show in the following tasks.

Another tool I missed was Swagger to easily visualise the API routes and test them. For this i added Ktor OpenAPI Tools, this library can automatically generate OpenAPI documentation from code and also provides a route for the Swagger UI. This is also shown in the following task.

When developing the graphql endpoint I also missed a playground with auto generated documentation. For this i used graphiQL using the already used GraphQL Kotlin library. This is show in task 2.

Taks

Task 1: Implementing REST

`Results`

Sources

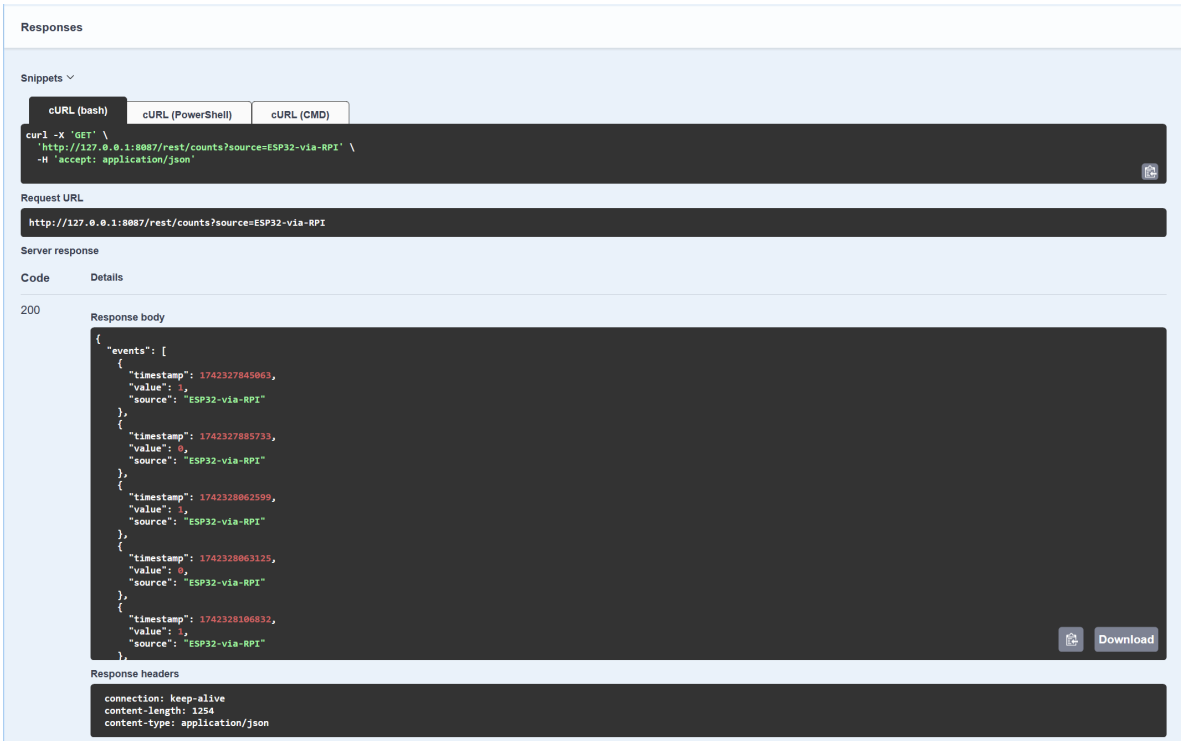
`Result`

`Code`

```kotlin
// Sources.kt
@Serializable
data class SourcesResponse(
    val sources: List<String>
)

@Resource("sources")
class Sources(
)
fun Route.sources() {
    get<Sources>({
        summary="Get all sources"
        description = "Returns an overview of all sources that ever
submitted data"
        response {
            HttpStatusCode.OK to { body<SourcesResponse> {} }
        }
    }) { _ ->
        val results = Influx.query(getSourcesFluxQuery())
        val foundSources = results.mapNotNull { record ->
            record.getValueByKey("source")?.toString()
        }

        call.respond(SourcesResponse(foundSources))
    }
}
```

Counts

Result



Code

```kotlin
// Counts.kt
@Serializable
data class EventData(
    val timestamp: Long,
    val value: Int,
    val source: String
)
@Serializable
data class CountsResponse(
    val events: List<EventData>
)

@Resource("counts")
class Counts(
    val start: Long = 946684800000, // 2000-01-01T00:00:00Z
    val stop: Long = -1, // -1 means now
    val source: String? = null
)
fun Route.counts() {
    get<Counts>({
        summary = "Get counts over time"
        description =
            "Returns an overview of the changing count values over time.
Filterable by start and stop timestamps (epochs in ms) and source tag."
        response {
            HttpStatusCode.OK to { body<CountsResponse> {} }
        }
    }) { counts ->
        val start = Instant.fromEpochMilliseconds(counts.start)
        val stop = if (counts.stop > -1)
Instant.fromEpochMilliseconds(counts.stop) else Clock.System.now()
        val source = counts.source

        val records = Influx.query( getCountFluxQuery(start, stop,
source))
        val events = records.map { record ->
            EventData(
                timestamp = record.time?.toEpochMilli() ?: 0,
                value = record.value.toString().toInt(),
                source = record.getValueByKey("source")?.toString() ?:
"unknown"
            )
        }

        call.respond(CountsResponse(events))
    }
}
```

IDs

Result

Responses

Snippets

cURL (bash) | cURL (PowerShell) | cURL (CMD)

```
curl -X 'GET' \
  'http://127.0.0.1:8087/rest/ids?stop=1743263951000&source=RPI' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8087/rest/ids?stop=1743263951000&source=RPI
```

Server response

Code	Details
200 *Undocumented*	Response body


```
    },
    {
      "timestamp": 1741814842101,
      "id": "10",
      "source": "RPI"
    },
    {
      "timestamp": 1741814843159,
      "id": "1",
      "source": "RPI"
    },
    {
      "timestamp": 1741814844217,
      "id": "9",
      "source": "RPI"
    },
    {
      "timestamp": 1741814845266,
      "id": "3",
      "source": "RPI"
    },
    {
      "timestamp": 1741814846313,
      "id": "0",
      "source": "RPI"
    },
    {
```


Download

Response headers

```
connection: keep-alive
content-length: 16728
content-type: application/json
```

Code

```kotlin
// IDs.kt
@Serializable
data class IDsEvent(
    val timestamp: Long,
    val id: String,
    val source: String
)

@Resource("ids")
class IDs(
    val start: Long = 946684800000, // 2000-01-01T00:00:00Z
    val stop: Long = -1, // -1 means now
    val source: String? = null
)
fun Route.ids() {
    get<IDs>({
        summary="Get scanned ids overview"
        description = "Returns an overview of scanned ids. Filterable by
start and stop timestamps (epochs in ms) and source tag."
        response {
            HttpStatusCode.OK to { body<IDsResponse> {} }
        }
    }) { ids ->
        val start = Instant.fromEpochMilliseconds(ids.start)
        val stop = if (ids.stop > -1)
Instant.fromEpochMilliseconds(ids.stop) else Clock.System.now()
        val source = ids.source

        val records = Influx.query(getIDsFluxQuery(start, stop, source))
        val events = records.map { record ->
            IDsEvent(
                timestamp = record.time?.toEpochMilli() ?: 0,
                id = record.getValueByKey("_value")?.toString() ?:
"unknown",
                source = record.getValueByKey("source")?.toString() ?:
"unknown"
            )
        }

        call.respond(IDsResponse(events))
    }
}
```

Attendance

Result

Responses

Snippets

cURL (bash) | cURL (PowerShell) | cURL (CMD)

```
curl -X 'GET' \
  'http://127.0.0.1:8087/rest/attendance?start=171172795100O' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8087/rest/attendance?start=171172795100O
```

Server response

Code	Details
200	Response body


```
{
  "events": [
    {
      "timestamp": 1741814689970,
      "id": "2",
      "arrival": true,
      "source": "RPI"
    },
    {
      "timestamp": 1741814691028,
      "id": "3",
      "arrival": true,
      "source": "RPI"
    },
    {
      "timestamp": 1741814692084,
      "id": "0",
      "arrival": false,
      "source": "RPI"
    },
    {
      "timestamp": 1741814693136,
      "id": "3",
      "arrival": false,
      "source": "RPI"
    },
    {
```


Download

Response headers

```
connection: keep-alive
content-length: 53180
content-type: application/json
```

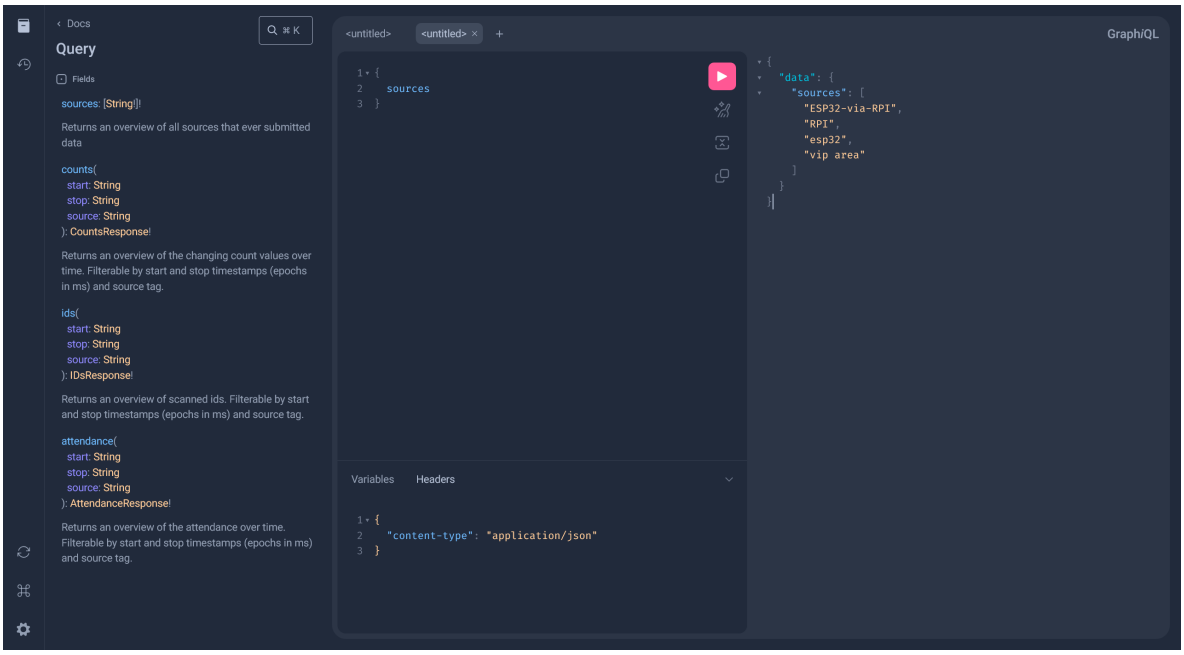
Code

```kotlin
// Attendance.kt
@Serializable
data class AttendanceResponse(
    val events: List<AttendanceEvent>
)
@Serializable
data class AttendanceEvent(
    val timestamp: Long,
    val id: String,
    val arrival: Boolean,
    val source: String
)

@Resource("attendance")
class Attendance(
    val start: Long = 946684800000, // 2000-01-01T00:00:00Z
    val stop: Long = -1, // -1 means now
    val source: String? = null
)
fun Route.attendance() {
    get<Attendance>({
        summary="Get attendance over time"
        description="Returns an overview of the attendance over time.
Filterable by start and stop timestamps (epochs in ms) and source tag."
        response {
            HttpStatusCode.OK to { body<AttendanceResponse> {} }
        }
    }) { attendance ->
        val start = Instant.fromEpochMilliseconds(attendance.start)
        val stop = if (attendance.stop > -1)
Instant.fromEpochMilliseconds(attendance.stop) else Clock.System.now()
        val source = attendance.source

        val records = Influx.query(getAttendanceFluxQuery(start, stop,
source))
        val events = records.map { record ->
            AttendanceEvent(
                timestamp = record.time?.toEpochMilli() ?: 0,
                id = record.getValueByKey("_value")?.toString() ?:
"unknown",
                arrival =
record.getValueByKey("arrival")?.toString()?.toBoolean() ?: false,
                source = record.getValueByKey("source")?.toString() ?:
"unknown"
            )
        }

        call.respond(AttendanceResponse(events))
    }
}
```

Task 2: Implementing GraphQL

SourcesQueryService

Result



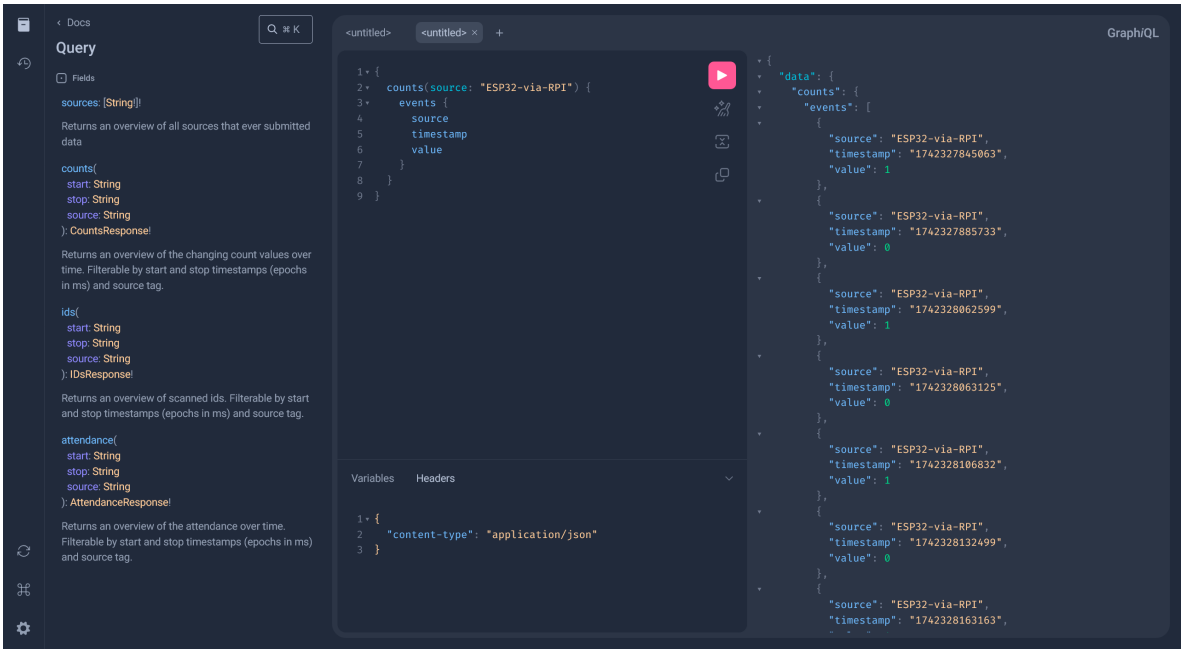

```
{
    sources
}
```

Code

```
// SourcesQueryService.kt
class SourcesQueryService: Query {
    @GraphQLDescription("Returns an overview of all sources that ever
submitted data")
    suspend fun sources() :List<String> {
        val records = Influx.query(getSourcesFluxQuery())
        val foundSources = records.mapNotNull { record ->
            record.getValueByKey("source")?.toString()
        }

        return foundSources
    }
}
```

CountsQueryService

Result



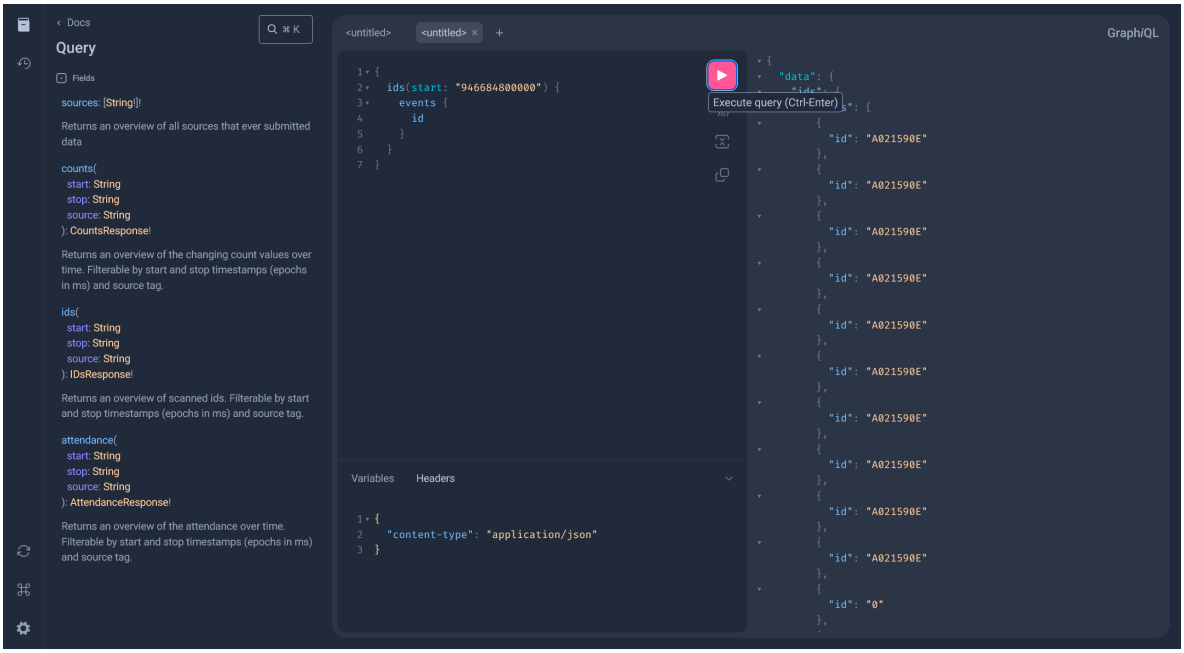

```
{
  counts(source: "ESP32-via-RPI") {
    events {
      source
      timestamp
      value
    }
  }
}
```

Code

```kotlin
// CountsQueryService.kt
@Serializable
data class EventData(
    val timestamp: String,
    val value: Int,
    val source: String
)
@Serializable
data class CountsResponse(
    val events: List<EventData>
)

class CountsQueryService: Query {
    @GraphQLDescription("Returns an overview of the changing count
values over time. Filterable by start and stop timestamps (epochs in ms)
and source tag.")
    suspend fun counts(
        start: String? ="946684800000", // 2000-01-01T00:00:00Z
        stop: String? = "-1", // -1 means now
        source: String? = null
    ) : CountsResponse {
        val startInstant = Instant.fromEpochMilliseconds((start ?:
"946684800000").toLong() )
        val stopInstant = if (stop != null && stop.toLong() > -1)
Instant.fromEpochMilliseconds(stop.toLong()) else Clock.System.now()

        val records = Influx.query( getCountFluxQuery(startInstant,
stopInstant, source))
        val events = records.map { record ->
            EventData(
                timestamp = (record.time?.toEpochMilli() ?:
0).toString(),
                value = record.value.toString().toInt(),
                source = record.getValueByKey("source")?.toString() ?:
"unknown"
            )
        }

        return CountsResponse(events)
    }
}
```

IDQueryService

Result



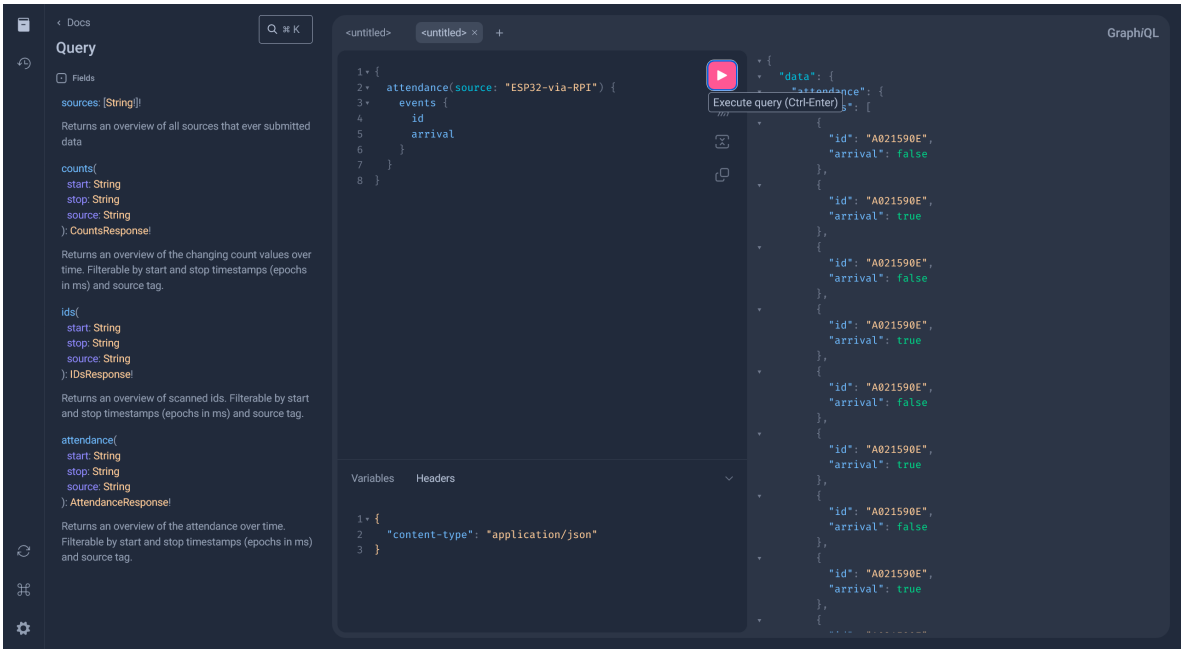

```
{
  ids(start: "946684800000") {
    events {
      id
    }
  }
}
```

Code

```kotlin
// IDQueryService.kt
@Serializable
data class IDsResponse(
    val events: List<IDsEvent>
)
@Serializable
data class IDsEvent(
    val timestamp: String,
    val id: String,
    val source: String
)

class IDQueryService: Query {
    @GraphQLDescription("Returns an overview of scanned ids. Filterable
by start and stop timestamps (epochs in ms) and source tag.")
    suspend fun ids(
        start: String? = null,
        stop: String? = null,
        source: String? = null
    ) : IDsResponse {
        val startInstant = Instant.fromEpochMilliseconds((start ?:
"946684800000").toLong() )
        val stopInstant = if (stop != null && stop.toLong() > -1)
Instant.fromEpochMilliseconds(stop.toLong()) else Clock.System.now()

        val records = Influx.query(getIDsFluxQuery(startInstant,
stopInstant, source))
        val events = records.map { record ->
            IDsEvent(
                timestamp = (record.time?.toEpochMilli() ?:
0).toString(),
                id = record.getValueByKey("_value")?.toString() ?:
"unknown",
                source = record.getValueByKey("source")?.toString() ?:
"unknown"
            )
        }

        return IDsResponse(events)
    }
}
```

AttendanceQueryService

Result




```
{
    attendance(source: "ESP32-via-RPI")
{
    events {
      id
      arrival
    }
  }
}
```

Code

```kotlin
// AttendanceQueryService.kt
@Serializable
data class AttendanceResponse(
    val events: List<AttendanceEvent>
)
@Serializable
data class AttendanceEvent(
    val timestamp: String, // Long is not supported in GraphQL
    val id: String,
    val arrival: Boolean,
    val source: String
)

class AttendanceQueryService: Query {
    @GraphQLDescription("Returns an overview of the attendance over time. Filterable by start and stop timestamps (epochs in ms) and source tag.")
    suspend fun attendance(
        start: String? = "946684800000", // 2000-01-01T00:00:00Z
        stop: String? = "-1", // -1 means now
        source: String? = null
    ): AttendanceResponse  {
        val startInstant = Instant.fromEpochMilliseconds((start ?: "946684800000").toLong() )
        val stopInstant = if (stop != null && stop.toLong() > -1) Instant.fromEpochMilliseconds(stop.toLong()) else Clock.System.now()

        val records = Influx.query(getAttendanceFluxQuery(startInstant, stopInstant, source))
        val events = records.map { record ->
            AttendanceEvent(
                timestamp = (record.time?.toEpochMilli() ?: 0).toString(),
                id = record.getValueByKey("_value")?.toString() ?: "unknown",
                arrival = record.getValueByKey("arrival")?.toString()?.toBoolean() ?: false,
                source = record.getValueByKey("source")?.toString() ?: "unknown"
            )
        }

        return AttendanceResponse(events)
    }
}
```

Task 3: Add Egress API to Your Microservices Architecture

Optional Task 4: Differences between GraphQL and REST

Task 5: Obtaining forecasts

Task 6: Scheduling and caching forecasts

Task 7: GETting forecasts

Questions

Question 1

> Why is it useful to expose the same content (same fields) from both REST and GraphQL queries?

Question 2

> Define the four GraphQL services using the GraphQL Schema Definition Language (SDL). If you need help doing this, here is a good explanation and examples of the SDL.

Question 3

> The forecasting job system helps with improving the egress' response time when querying forecasting results. Give two more reasons why scheduling and/or caching these requests/results is beneficial.