Features (regarding queries from CorporateMemory):

1. Named Graph (by adapting queries):

   2 cases: a) default graph = named graph —> remove named graph

   ex:
   ```
   select *
   from <graph>
   from named <graph>
   where { ?s ?p ?o .}
   ```
   
   ```
   select *
   from <graph>
   where { ?s ?p ?o .}
   ```

   b) only named graph —> query with filter

   ex:
   ```
   select *
   from named <graph>
   where { graph ?g
      { ?s ?p ?o .}
   }
   ```
   
   ```
   select *
   from named <graph>
   where { graph ?g
      { ?s ?p ?o .}
   filter(?g in(named_list)) }
   ```

2. (Auto union all graphs when) graph is not defined in queries:

   ```
   select * where { ?s ?p ?o .}
   ```

I've tested some endpoints like Dbpedia, stardog, in this case, all triples from all graphs should be delivered. I did it also by adapting queries automatically to

```
select * where { graph ?g { ?s ?p ?o . } }
```

3. PUT as you see.
4. Large file upload: transfer parsed graph directly instead of query (see helpers.py)
5. DROP graph
6. CREATE graph (create graph is possible, but won't be shown by „selecting all graphs“ —> same case in CLEAR)
7. CLEAR graph
8. Support for Content-Type: „application/rdf+xml“
9. Commit message —> only for demonstration

10. Fix Construct-query

Problems:

1. Query Problems (the following queries don't work, I'm still not sure where the problems are):

A. select count distinct

ex:

```
SELECT  (COUNT(DISTINCT ?numSchemes) AS ?schemes) (COUNT(DISTINCT ?numConcept)
AS ?concept) (COUNT(DISTINCT ?numNarrowerConcept) AS ?narrowerConcepts)
(COUNT(DISTINCT ?numRelation) AS ?relations)
WHERE
  { GRAPH <https://ns.eccenca.com/9d09daf3-16bf-49fa-9289-e81478a20b32/>
      {   { ?numSchemes  a <http://www.w3.org/2004/02/skos/core#ConceptScheme> }
        UNION
          { ?numConcept  a  <http://www.w3.org/2004/02/skos/core#Concept> }
        UNION
          { ?numNarrowerConcept
            <http://www.w3.org/2004/02/skos/core#narrower>  ?connectedConcepts
          }
        UNION
       { ?numRelation  <http://www.w3.org/2004/02/skos/core#related>  ?related }
      }
  }
```

B. Queries with having bound

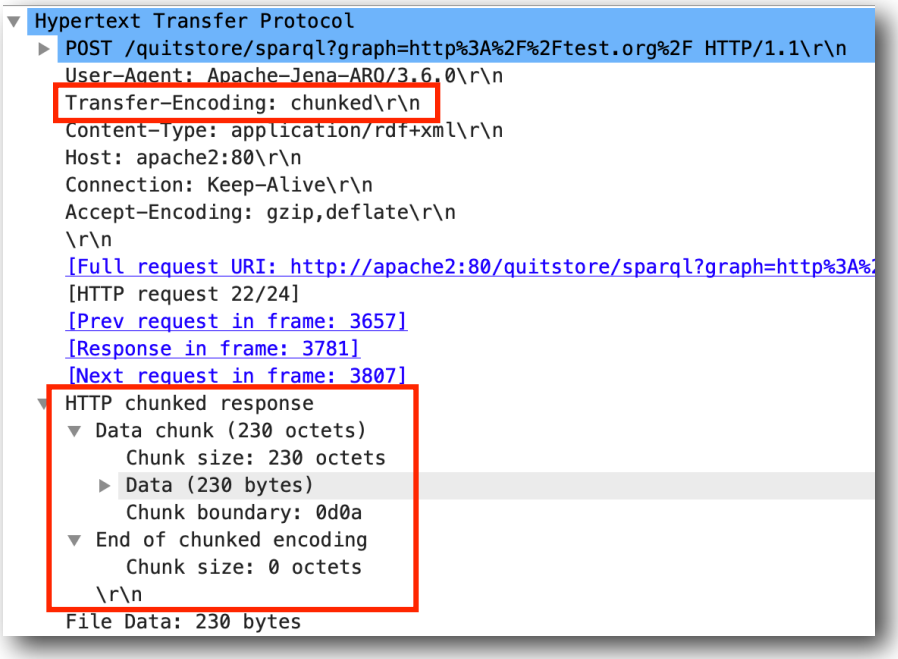
ex:

```
SSELECT DISTINCT  ?dataset ?description ?domain ?status ?created ?modified
FROM <https://ns.eccenca.com/example/data/dataset/>
WHERE
  { {SELECT  ?dataset ?description ?domain ?status ?created (SAMPLE(?mod) AS ?
modified)
      WHERE
        { ?dataset  a  <https://vocab.eccenca.com/dsm/Dataset>
          OPTIONAL
            { ?dataset  <http://purl.org/dc/terms/created>  ?created }
          OPTIONAL
            { ?dataset  <http://purl.org/dc/terms/modified>  ?mod }
          OPTIONAL
            { ?dataset  <https://vocab.eccenca.com/dsm/hasStatus>  ?status }
          OPTIONAL
            { ?dataset  <http://purl.org/dc/terms/description>  ?description }
          OPTIONAL
            { ?dataset  <http://www.w3.org/ns/dcat#theme>  ?domain }
        }
      GROUP BY ?dataset ?description ?domain ?status ?created ?modified
      HAVING bound(?dataset)
    }
```

```
    OPTIONAL
      { ?dataset  <http://www.w3.org/2000/01/rdf-schema#label>  ?datasetLabel1 }
    OPTIONAL
      { ?dataset  <http://www.w3.org/2004/02/skos/core#prefLabel>  ?
datasetLabel2 }
    OPTIONAL
      { ?dataset  <http://www.w3.org/2004/02/skos/core#definition>  ?
datasetLabel3 }
    OPTIONAL
      { ?dataset  <http://purl.org/dc/terms/description>  ?datasetLabel4 }
    OPTIONAL
      { ?dataset  <http://www.w3.org/ns/shacl#name>  ?datasetLabel5 }
    FILTER ( ( ( ( ( ( isLiteral(?dataset) && contains(lcase(str(?dataset)),
lcase("test")) ) || ( isLiteral(?datasetLabel1) && contains(lcase(str(?
datasetLabel1)), lcase("test")) ) ) || ( isLiteral(?datasetLabel2) &&
contains(lcase(str(?datasetLabel2)), lcase("test")) ) ) || ( isLiteral(?
datasetLabel3) && contains(lcase(str(?datasetLabel3)), lcase("test")) ) ) ||
( isLiteral(?datasetLabel4) && contains(lcase(str(?datasetLabel4)),
lcase("test")) ) ) || ( isLiteral(?datasetLabel5) && contains(lcase(str(?
datasetLabel5)), lcase("test")) ) )
  }
OFFSET  0
LIMIT   10
```

2. HTTP Problem

Requests with chunked data (Transfer-Encoding: chunked) cannot be correctly processed by QuitStore(maybe by uwsgi framework) like following:



Other information in request could arrive at QuitStore correctly, only the chunked data was ignored. Although I used Apache-Proxy to dechunk the data and it worked, it would still be better when QuitStore could process this data correctly.