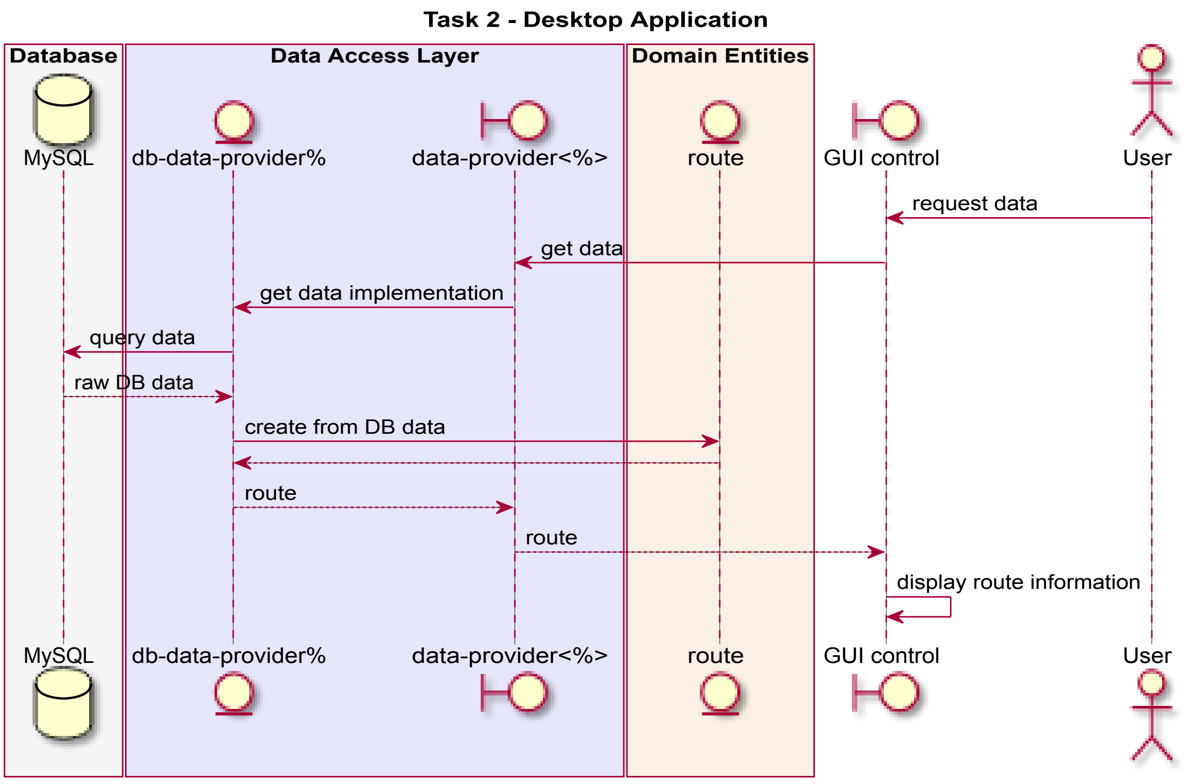
1 Task 2 - Desktop Application

To recap, here is a typical data retrieval and display cycle as implemented in the desktop application:

- user performs action on a Racket GUI control
- interaction of GUI layer with data access layer via interface `data-provider<%>`
- database-backed implementation `db-data-provider%` interacts with database via queries
- domain entities (e.g., `route` struct) are created or manipulated
- domain entity data is processed and displayed on the GUI

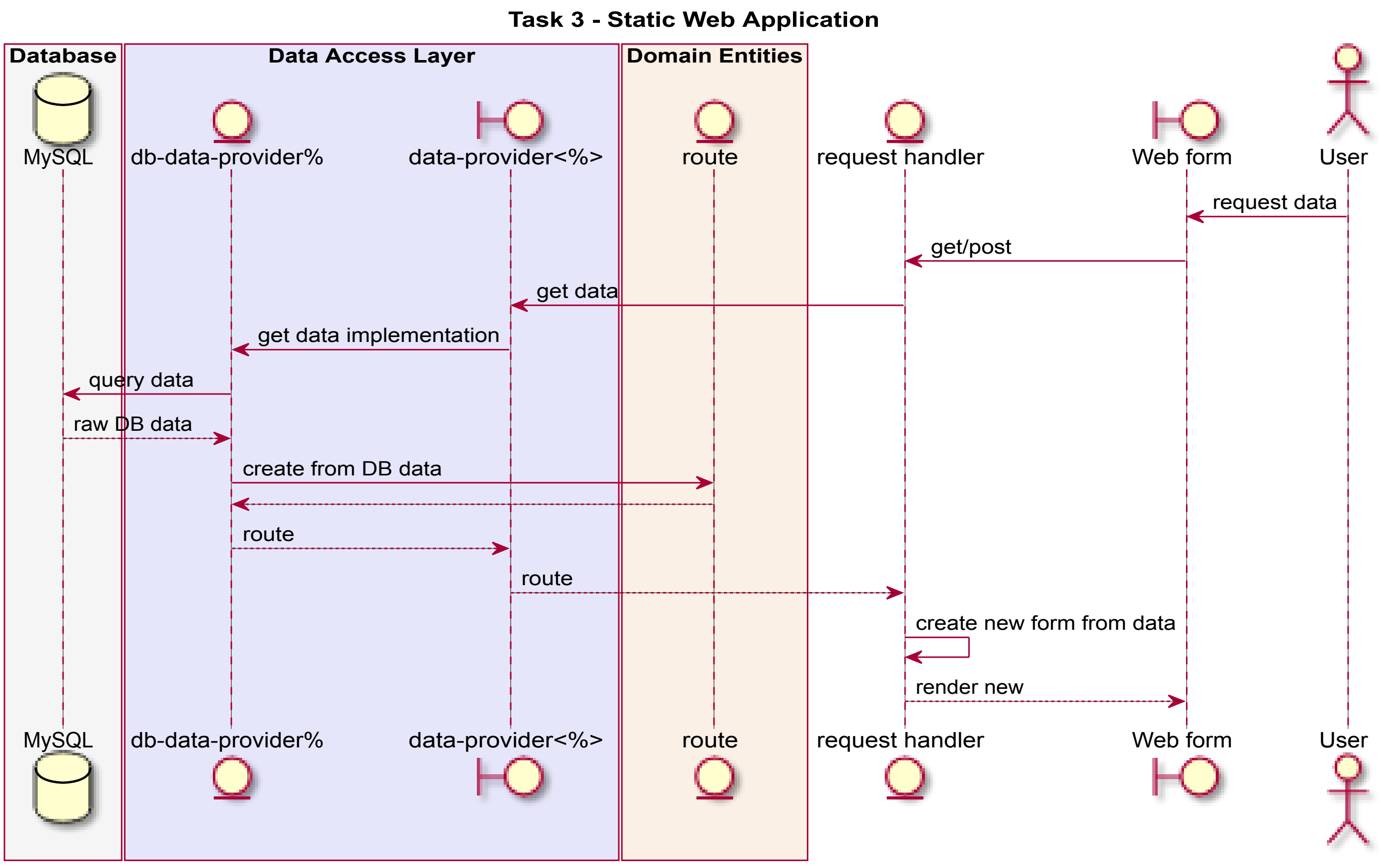


2 Task 3 - Static Web Application

- The *static web application* uses the same database server, data access layer and domain entities as the GUI Application.
- Web page handling is performed by Racket's *web-server/insta* Domain Specific Language. Running of the web server and handling of callback urls as well as routing to request handler functions are handled by *web-server/insta*.
- Requests are delivered to handler functions as *s-expression*-formatted Racket data. Responses are described by the *web-server/insta*'s nested *s-expressions* which describe HMTL content. Dynamic content is inserted via Racket's *unquote* mechanisms, e.g., `,` `(unquote)` to insert one datum or `,@` `(unquote-splicing)` to insert a list.

Example code handling a request and showing a static web form with dynamic data content embedded in a selection list:

```
(define (show-all request)
  (define (response-generator embed/url)
    (response/xexpr
     `(html
       (head (title "All"))
       (body (h1 "List of Persons")
             (form ([action ,(embed/url show-selected)])
                   (p (select ([size "5"]
                               [name "person"])
                              ,@(all-persons)))
                   (p (input ([type "submit"]))))))))
  (send/suspend/dispatch response-generator))
```



3 Task 4 - Dynamic Web Application

- The *dynamic web application* uses the MySQL database server and data model from Tasks 2 and 3.
- The JavaScript library *Bootstrap* accesses the database directly via the underlying *jQuery data-table* mechanism.
- The Racket implementations from Tasks 2 and 3 are not reused due to the following reasons:
    - a dedicated JavaScript frontend library like *Bootstrap* already has existent methods to perform the requested dynamic content manipulations
    - development of Tasks 3 and 4 can be highly parallelized since no common code base has to be considered