USE TO-DO APPLICATION TO ORGANIZE USER'S TASKS

Group 2

Mario Matos

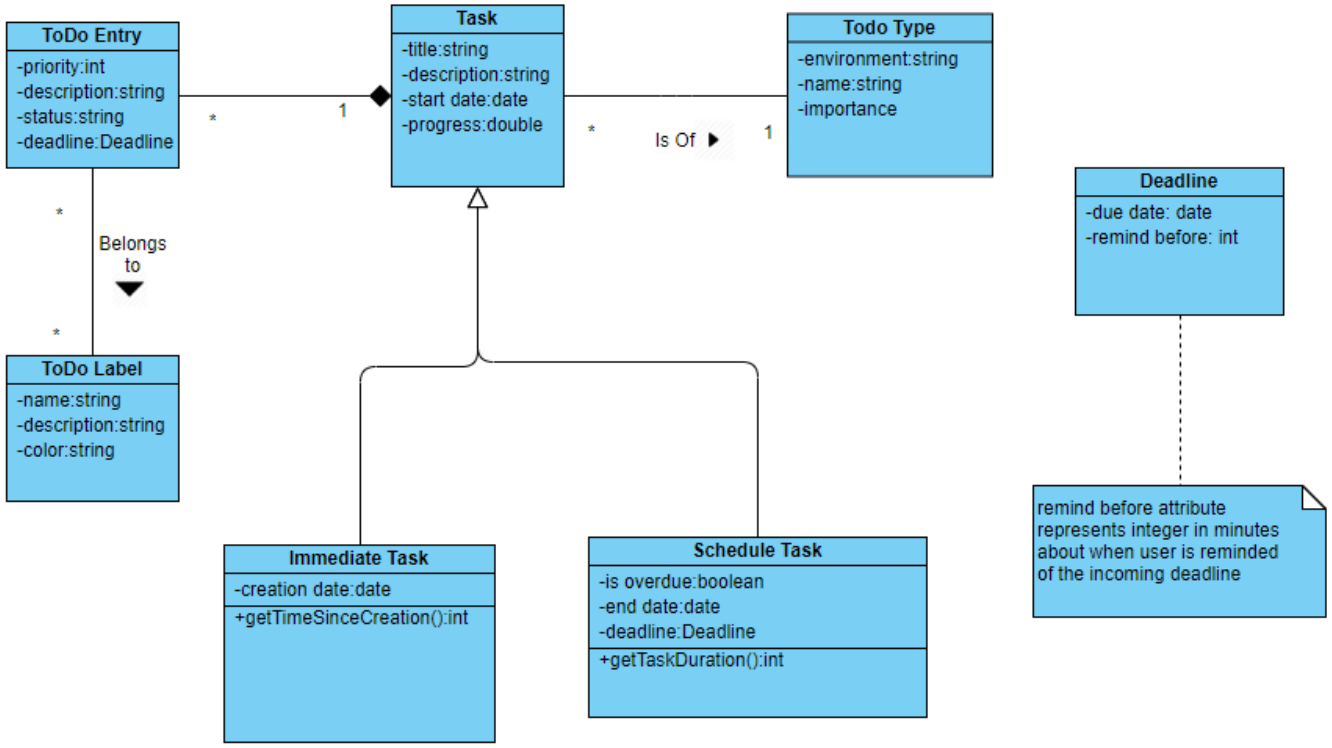
Mostafa Mohamed

Isam Sebri

PROBLEM DOMAIN

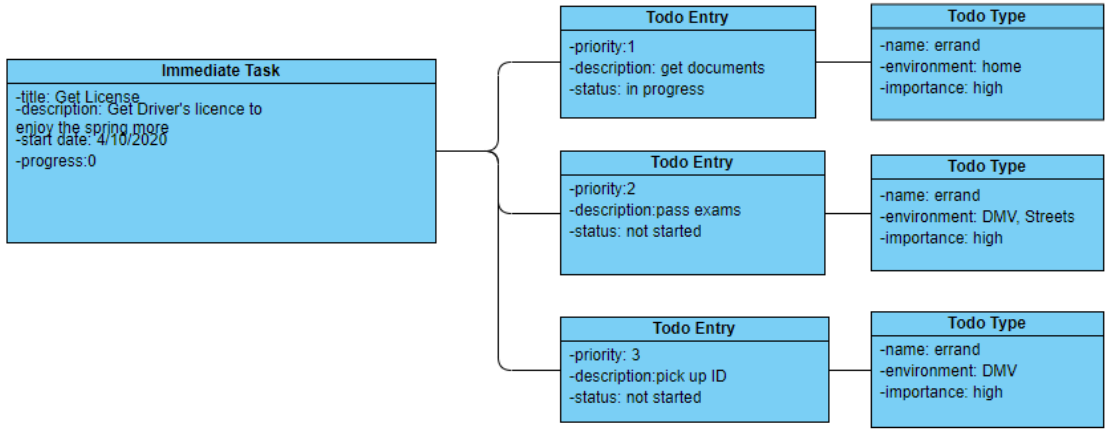
- Our group project aims to be an application that can be used by users to organize their errands and other tasks that they need to complete on a daily or scheduled basis. Sometimes it is hard to keep track of all the things an individual needs to get done in their daily routines or what things need to be scheduled to be incorporated later.
- Our application will have a dashboard for the user to see the things that he needs to complete and a way to mark things as done or in progress. For the time being however, we are focusing on creating a server with semantic API Endpoints using REST.
- Each to-do entry should be able to contain optional labels for easy classification depending on what they are related to in relation to the user's life. For example, labels can be 'professional', 'school', 'personal', 'leisure', etc. Also, we have upgraded our model to have different types for each Task.
- In the case of scheduled tasks, a user should be able to set a reminder that will have a duration of time to remind the user in minutes. That way a reminder can be 15, 30, 45, 60, etc minutes before it's due date.

CLASS DIAGRAM

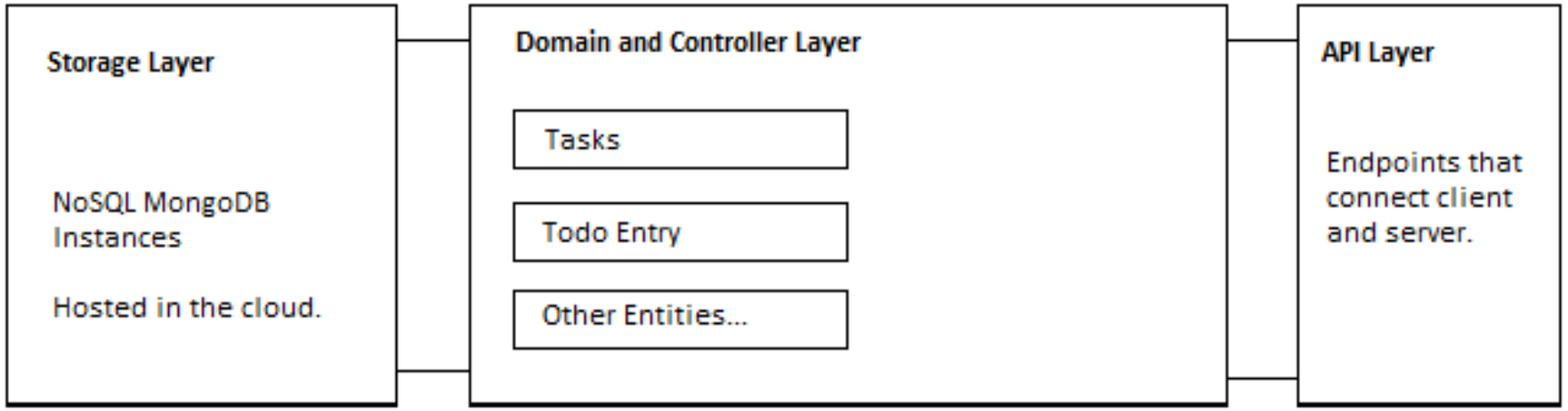


OBJECT DIAGRAM

- Again, to better visualize how our app will work we include an object diagram with instances that model the class diagram above:



Project Architecture



BUSINESS FUNCTIONS/SERVICES

- Create task
- Add steps to complete tasks (to-do entry)
- Add labels to to-do entries for classification
- Add immediate tasks and schedule tasks.
- Show not completed tasks and their to-do entries.
- Show schedule tasks when they are coming due and their due dates
- Have optional reminders for scheduled tasks as well as all individual to-do entries.

CONCEPTS DEMOSTRATED

- Software design and UML Diagram creation.
- Different relations among classes: Inheritance, association, composition.
- Polymorphism
- Functional Interfaces
- Lambda expressions and method references for implementation of Functional Interfaces.
- Streams for organizing collections of datas or collection of todos.
- Testing of different functionalities
- Annotations

API Spec

For each of the following object a GET, POST and DELETE endpoints will be generated.
All types can be infered by the name of the attribute. The response object for GET and POST
will have same body as the attributes. The request body for POST will have the same attributes.
DELETE will only have ids of objects to delete.

task

- -title
- -description
- -start date (date)
- -progress (double, 0 to 1)

todoentry

- -priority
- -description
- -status
- -deadline (id) or object
- -labels (list of ids) or list of objects object

API Spec

For each of the following object a GET, POST and DELETE endpoints will be generated.
All types can be infered by the name of the attribute. The response object for GET and POST
will have same body as the attributes. The request body for POST will have the same attributes.
DELETE will only have ids of objects to delete.

task

- -title
- -description
- -start date (date)
- -progress (double, 0 to 1)

todoentry

- -priority
- -description
- -status
- -deadline (id) or object
- -labels (list of ids) or list of objects object

API Spec

todolabel

-name

-description (optional)

-color

tasktype

-name

-environment

-importance

immediatetask

- (all task attributes)

- creation date

(for GET response include calculated time since creation)

scheduledtask

-is overdue

-end date

-deadline (id or object)

- (for GET request include task duration)

API Spec

deadlines

-due date

-remind before (int representing number of days)

GET requets:

\tasks: return a list of tasks including id, title, description and status depending on status of it's todo entries

\task\:id : return the object whose id is the id used as argument.

\task\:id\todoentries: returns a list of todo entries for a specific task. include description, priorty and status

\tododentries\:id: returns all attributes for a todo entry.

\labels: returns a list of all labels in the the system.

for immediate and scheduled tasks repeat the same first three endpoints.

todo type and deadlines are internal objects of the server and ar manipulated according to the requests above

POST requests:

POST requests will not be idempotent, that is they will add an ojbect if it doesnt exists and modify it if it does. That way it will act as insert and update at the same time.

ids in uri are optional, if included they'll be matched to an existing object, if non existing a new object will be created.

API Spec

\task\<:id> : insert or update object referenced by id. The request body will have all attributes related to a task. todo entry can be empty or have a list of todo entries.

\task\:id\todoentries\<:id> : insert or update todo entry referenced by id.

\label\<:id> : insert or update label. id in uri is optional but must be included in body.

\tasktype\<:id> : insert or update task type.

for immediate and scheduled tasks repeat the same first two endpoints.

DELETE requests:

For the sake of brevity, all objects associated with other objects will be deleted on cascade, when not associated with other objects.

\task\:id : insert or update object referenced by id. The request body will have all attributes related to a task. todo entry can be empty or have a list of todo entries.

\task\:id\todoentries\:id : insert or update todo entry referenced by id.

\label\:id : insert or update label. id in uri is optional but must be included in body.

\tasktype\:id : insert or update task type.

CONCLUSION

- Our To Do Application aims to make organizing things to do for a user a better experience and to be a robust system that will model the natural workflow of organizing their day.

It’s DEMO time…

Server implementation using Repository pattern and MongoDB for persistence of data: