RAWDATA SUBPROJECT 2

Group members

Thomas Halberg, 03528073
Søren Ulrik Johansen, 55121
Frederik Nordam Rosenvilde Jakobsen, 55478
Jean-Alexandre Pecontal, 62150
Baptiste Jouan, 62147

https://github.com/Furnez/rawdata_sova

Introduction

The goal of this portfolio subproject 2 is to add a restful web service interface to the SOVA application (Stack Overflow Viewer Application) and to extend the functionality of this. We want the architecture to be as maintainable, testable, extendable, and scalable as possible, thus do it the "right way" even through the complexity of the model and application for now not necessary justify this. We assume, of course, that the system will need expansion in the future, due to all the nice features.

Application Design

In this part we will sketch a preliminary application design by explaining the architecture of our backend, describing the dependencies between the different layers of the system, and the structure of the domain objects. Furthermore we will describe some of the use cases/stories that our system supports.

Requirements from the assignment:

- Search for posts (questions and answers) with word and phrases.
- Search for comments to posts with word and phrase.
- Sort the search results on the rank of the founded posts.
- Make a visual (maybe graphs) of the most used words and phrases
- Make list of personal (user) search history
- Mark a post of special interest
- Make personal notes to posts (questions and answers) and comments

Requirements from the application's design process:

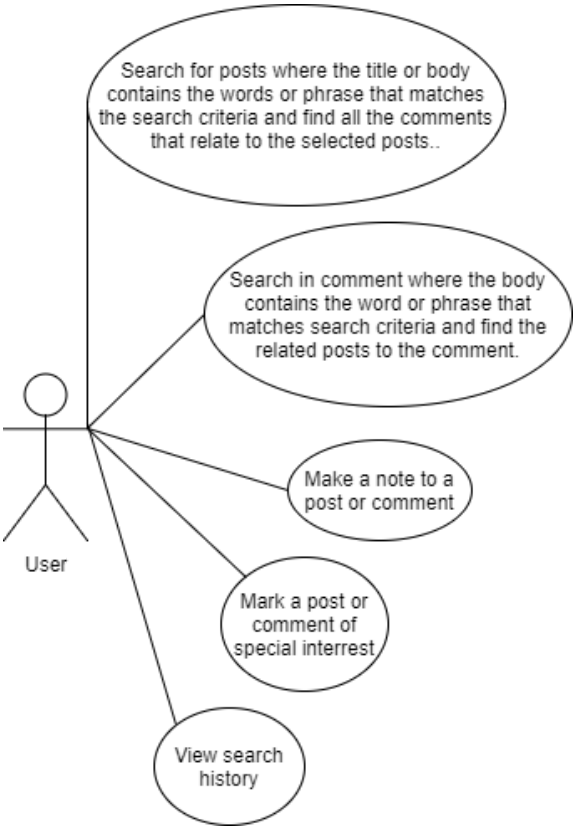
- From one user app to multi user app
- Log in with password (user authentication)
- Make a visual graphs of the most used words and phrases in form of cloud/ball figur
- Make comment to one post.
- Make a comment to a comment to the same post
- Rank a post or a comment
- Display user data to a post and to a comment

In the design process we have operated with 2 business scenarios: a single user application and a multi user application.

Use case diagram for a single user application

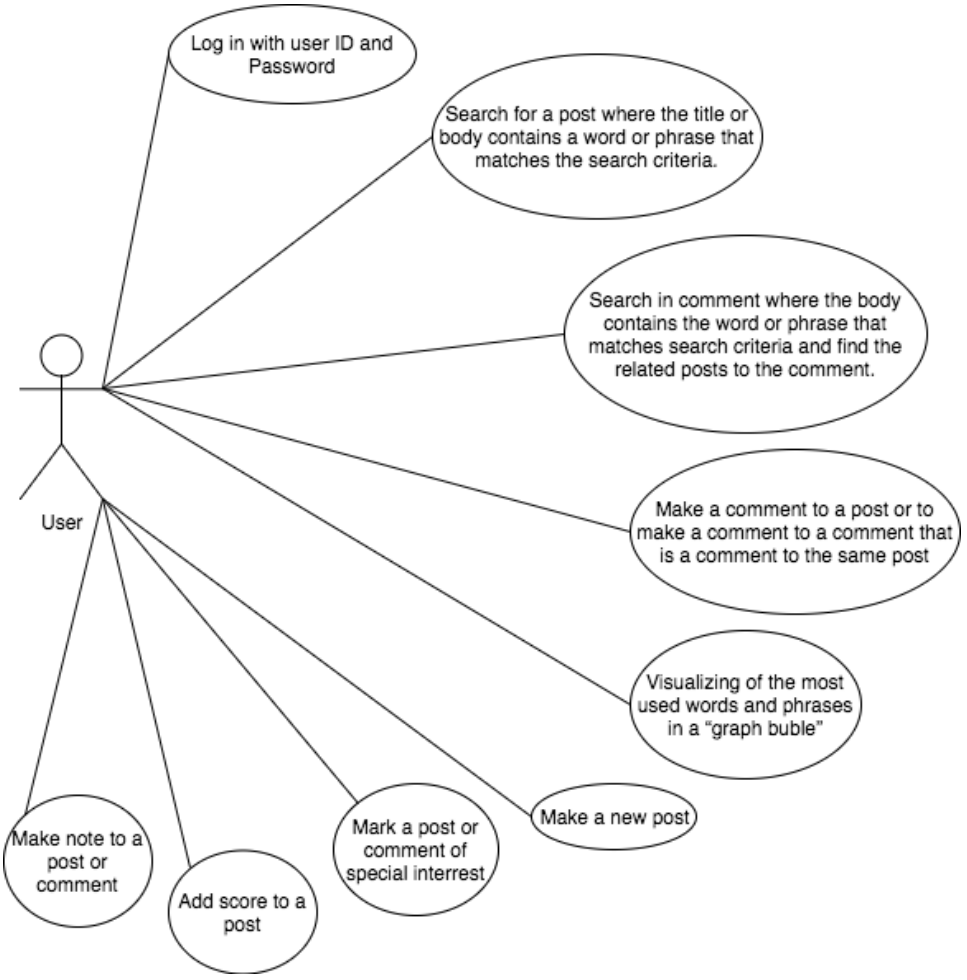
First we made use cases for a single user application, which uses the data, we got in sub portfolio project 1 . This application is a single user application, where the user can search for an answer to a subject or a problem primary in posts, the search include search in comments. The search can be a word or a phrase search. The user can mark a post of special interest, make a personal note to a post or a comment and see the search history.

a single user application has limitations, it's meaningless to make a post, make a comment to a post or a comment to an existing post and the ability to rank a post or a comment.



Use case diagram for multiple users

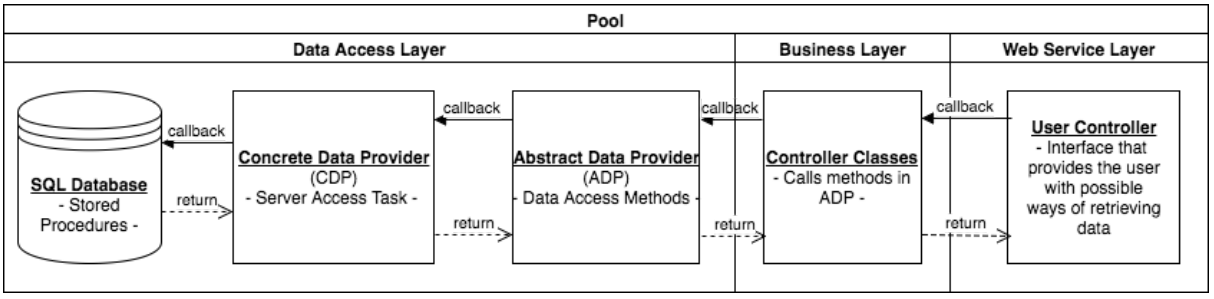
The of the multiple user application include the functionality of the single user application. The following functionality has been added, the users can register and log in with their own ID and Passwords, user data can be displayed, the user can make a post, and make a comment to a post or make a new comment to a comment related to a post, the user can rank a post or a comment. The user can see and use the word or graph visualiser.



Dependency Diagram

The next section will explain the dependency diagram of our system.

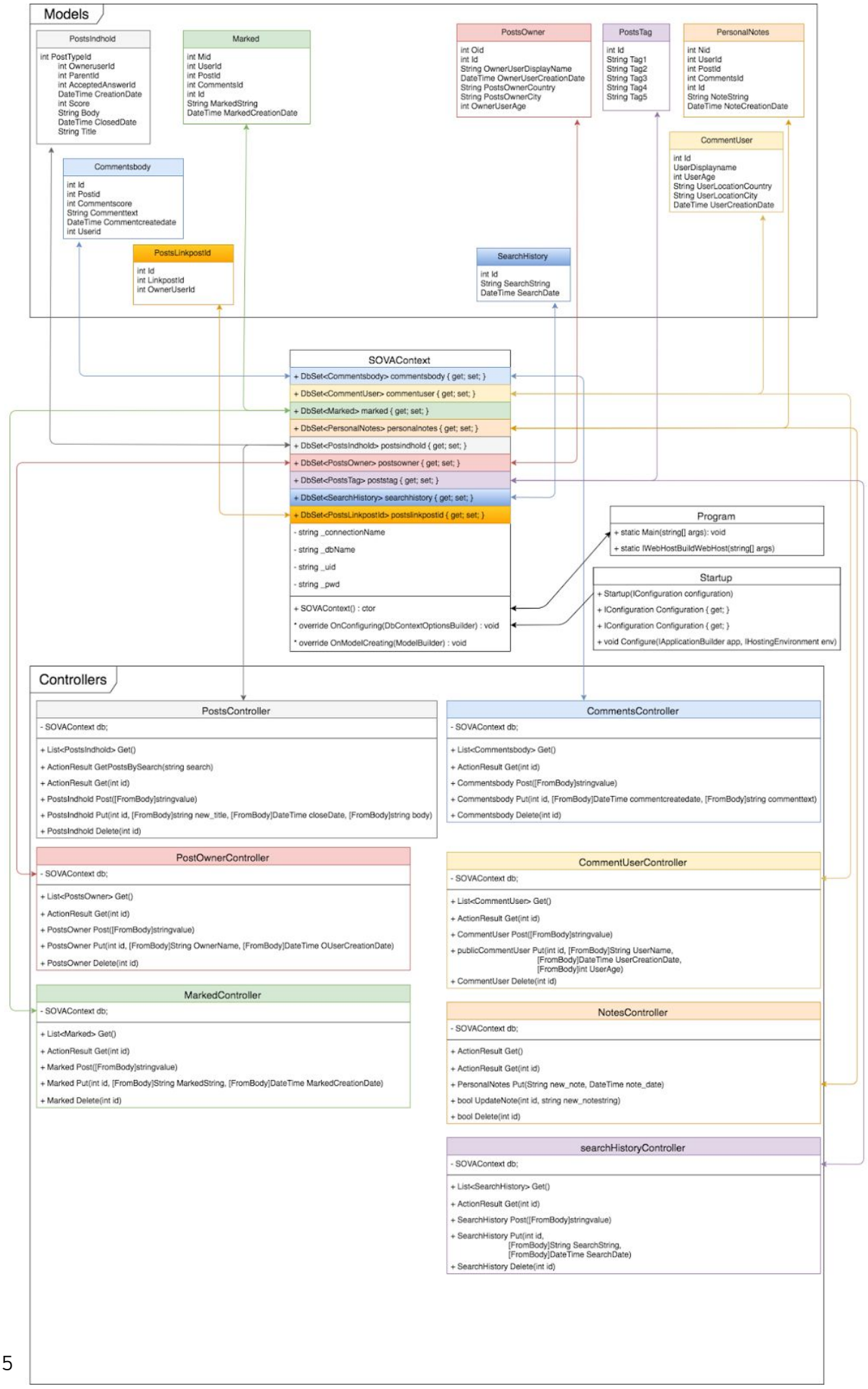
The model below shows the dependencies between the different layers of our system, the Data Layer, The Business Layer and the Web Service Layer. **The Data Access Layer** consists of three parts, The Database, that provide all the data and stored procedures, the Concrete Data Provider (CDP) that manage the server access task from the database to the IDE - visual studio in our case. Last we have the Abstract Data Provider (ADP) which are a bunch of data access methods that are abstract in the way that they each only handle parts of the data source or schemas - which makes it easier for the controller classes to handle specific tasks. The second part of the model is the **Business Layer**. This layer holds the Controller Classes. These classes calls on the methods in ADP that makes it possible to specify and categories ways of accessing specific parts of the data - in our system we have *PostController.cs, NotesController.cs, MarkedController.cs* etc.. The Last part of the model is the **Web Service Layer (WSL)**. WSL provides the user/client with an interface that mediate between the backend functionality in an accessible way, all the way back to the database. These dependencies are shown through callbacks and returns in the model.



Class Diagram

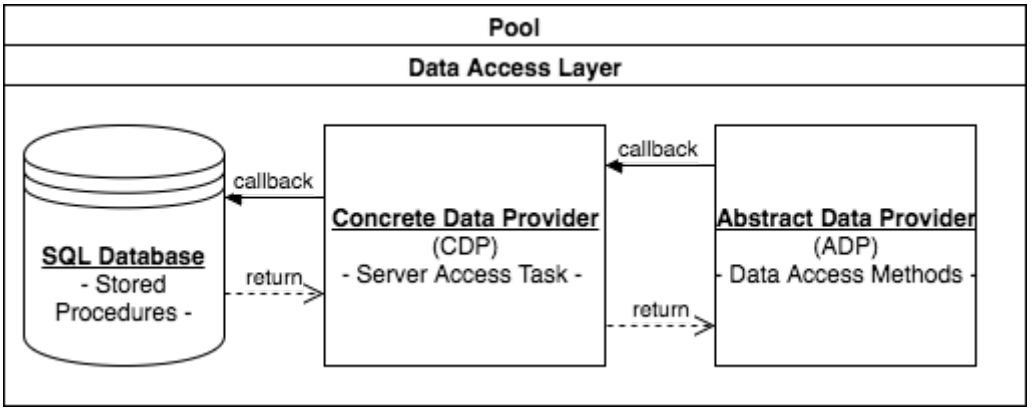
The class diagram, shown on the next page, shows the classes, their attributes and methods, and the relationship between the classes. From the top we see the **Models**, these include every class that hold database tables. Each of the Model classes have a relationship line to the SOVAContext class, that acts as a mediator between the **Models** and the **Controllers.** The **SOVAContext** class get's the attributes from each Model class, and send it to the controller class for use. SOVAContext also include methods for Configuring the database access, provided by the **Program** class. The **Controller** class acts as a mediator between the user and the database and consists of methods that respond to the HTTP request, also called "routes".
Conclusion: We have realised that our backend system contains too many controller classes, and that we profitably can merge some of these classes into a single class on purpose. This will result in the classes being more flexible and more focus on related data, in its way of solving the tasks.

If you have difficulties reading the diagram, you can find a copy of it in PDF format on the repository of the project.


Models
PostsIndhold
int PostTypeId
int OwneruserId
int ParentId
int AcceptedAnswerId
DateTime CreationDate
int Score
String Body
DateTime ClosedDate
String Title
Marked
int Mid
int UserId
int PostId
int CommentsId
int Id
String MarkedString
DateTime MarkedCreationDate
PostsOwner
int Oid
int Id
String OwnerUserDisplayName
DateTime OwnerUserCreationDate
String PostsOwnerCountry
String PostsOwnerCity
int OwnerUserAge
PostsTag
int Id
String Tag1
String Tag2
String Tag3
String Tag4
String Tag5
PersonalNotes
int Nid
int UserId
int PostId
int CommentsId
int Id
String NoteString
DateTime NoteCreationDate
CommentUser
int Id
UserDisplayname
int UserAge
String UserLocationCountry
String UserLocationCity
DateTime UserCreationDate
Commentsbody
int Id
int Postid
int Commentscore
String Commenttext
DateTime Commentcreatedate
int Userid
PostsLinkpostId
int Id
int LinkpostId
int OwnerUserId
SearchHistory
int Id
String SearchString
DateTime SearchDate
SOVAContext
+ DbSet<Commentsbody> commentsbody { get; set; }
+ DbSet<CommentUser> commentuser { get; set; }
+ DbSet<Marked> marked { get; set; }
+ DbSet<PersonalNotes> personalnotes { get; set; }
+ DbSet<PostsIndhold> postsindhold { get; set; }
+ DbSet<PostsOwner> postsowner { get; set; }
+ DbSet<PostsTag> poststag { get; set; }
+ DbSet<SearchHistory> searchhistory { get; set; }
+ DbSet<PostsLinkpostId> postslinkpostid { get; set; }
- string _connectionName
- string _dbName
- string _uid
- string _pwd
+ SOVAContext() : ctor
* override OnConfiguring(DbContextOptionsBuilder) : void
* override OnModelCreating(ModelBuilder) : void
Program
+ static Main(string[] args): void
+ static IWebHostBuildWebHost(string[] args)
Startup
+ Startup(IConfiguration configuration)
+ IConfiguration Configuration { get; }
+ IConfiguration Configuration { get; }
+ void Configure(IApplicationBuilder app, IHostingEnvironment env)
Controllers
PostsController
- SOVAContext db;
+ List<PostsIndhold> Get()
+ ActionResult GetPostsBySearch(string search)
+ ActionResult Get(int id)
+ PostsIndhold Post([FromBody]stringvalue)
+ PostsIndhold Put(int id, [FromBody]string new_title, [FromBody]DateTime closeDate, [FromBody]string body)
+ PostsIndhold Delete(int id)
CommentsController
- SOVAContext db;
+ List<Commentsbody> Get()
+ ActionResult Get(int id)
+ Commentsbody Post([FromBody]stringvalue)
+ Commentsbody Put(int id, [FromBody]DateTime commentcreatedate, [FromBody]string commenttext)
+ Commentsbody Delete(int id)
PostOwnerController
- SOVAContext db;
+ List<PostsOwner> Get()
+ ActionResult Get(int id)
+ PostsOwner Post([FromBody]stringvalue)
+ PostsOwner Put(int id, [FromBody]String OwnerName, [FromBody]DateTime OUserCreationDate)
+ PostsOwner Delete(int id)
CommentUserController
- SOVAContext db;
+ List<CommentUser> Get()
+ ActionResult Get(int id)
+ CommentUser Post([FromBody]stringvalue)
+ publicCommentUser Put(int id, [FromBody]String UserName, [FromBody]DateTime UserCreationDate, [FromBody]int UserAge)
+ CommentUser Delete(int id)
MarkedController
- SOVAContext db;
+ List<Marked> Get()
+ ActionResult Get(int id)
+ Marked Post([FromBody]stringvalue)
+ Marked Put(int id, [FromBody]String MarkedString, [FromBody]DateTime MarkedCreationDate)
+ Marked Delete(int id)
NotesController
- SOVAContext db;
+ ActionResult Get()
+ ActionResult Get(int id)
+ PersonalNotes Put(String new_note, DateTime note_date)
+ bool UpdateNote(int id, string new_notestring)
+ bool Delete(int id)
searchHistoryController
- SOVAContext db;
+ List<SearchHistory> Get()
+ ActionResult Get(int id)
+ SearchHistory Post([FromBody]stringvalue)
+ SearchHistory Put(int id, [FromBody]String SearchString, [FromBody]DateTime SearchDate)
+ SearchHistory Delete(int id)

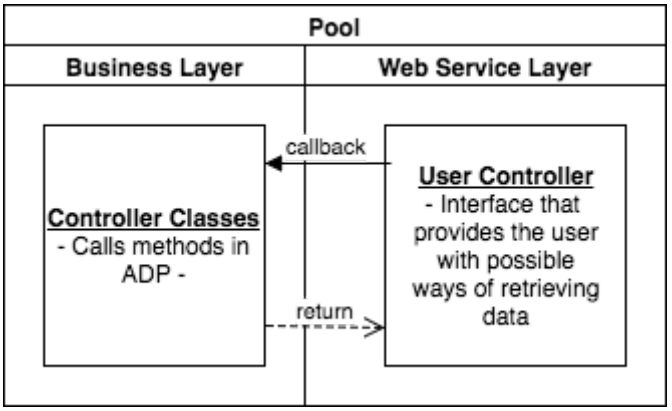
Data Access Layer (DAL)

DAL is used to to abstract away the concrete implementations, and provide a more generic interface to the sources. The DAL consists of two parts. The first part is the Concrete Data Provider (CDP), which handles the connection to and from the server. The second part is the Abstract Data Provider (ADP), which creates abstract data methods that do not rely purely on the sources, but only on a specific part of the source. Furthermore we have used Modular programming as our main method. The reason for this is that the Dataservice should be able to access and handle data from more than one source.



Web Service Layer (WSL)

The main goal for the WSL is to provide transition between the backend functionality and the user interface in an accessible way. The overall structure of this part is to reply to a user from clients, by use of the Business Layer and the Data Access Layer. At the moment we don't have an ideal interface for the Web Service Layer, but we have all the functionality through the controller classes in the Business Layer.

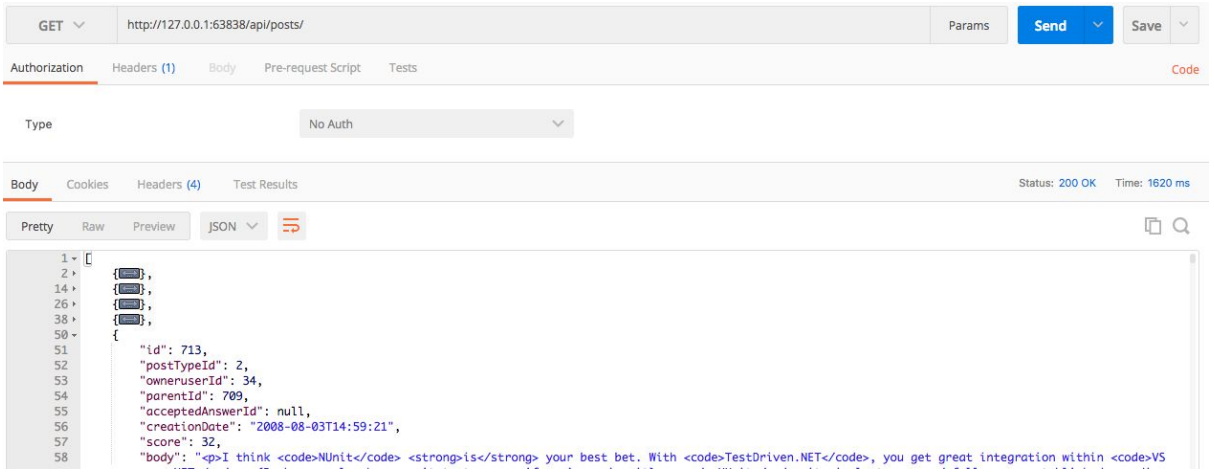


Testing

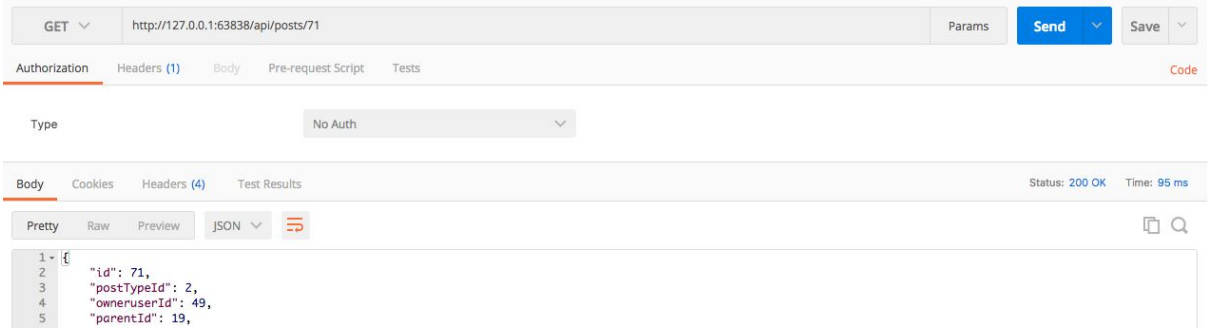
Test cases data retrieval from the API

PostsController methods

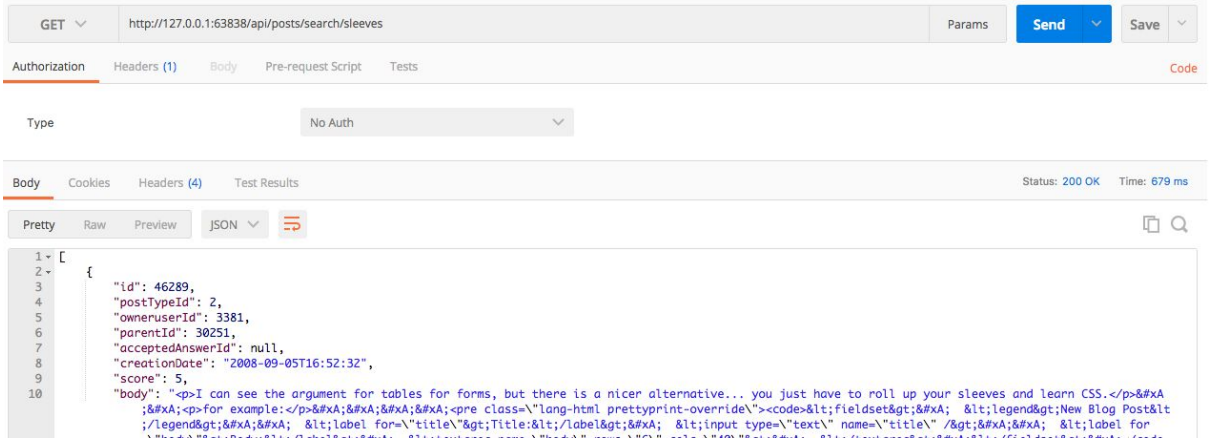
scenario 1. Retrieve a list of all posts :
Expected results: a list of all posts in the database.
Test result:



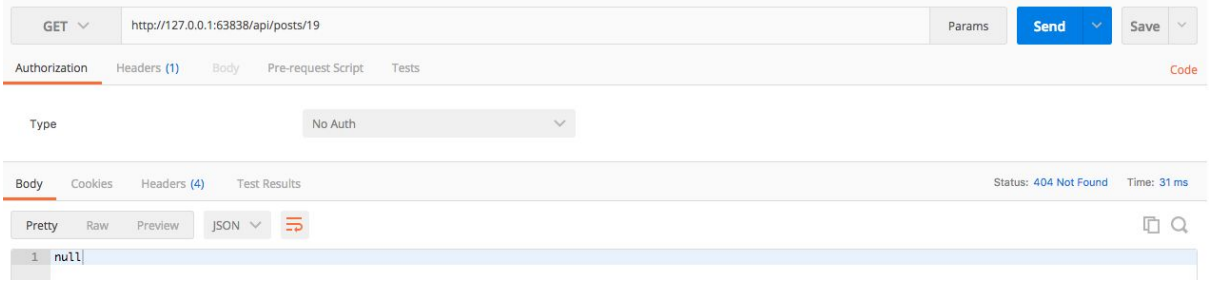
scenario 2. Retrieve a post by its id :
Expected results: the data from post with the id 71
Test result:



scenario 3. Retrieve a post by search string :
Expected results: The one post where sleeves is in the title or in the body
Test result:



scenario 4. Delete post by its id (admin functionality) :
Expected results:
Test result: post 19 is deleted
After removing the post id=19, we get null and a 404 error when searching for it :



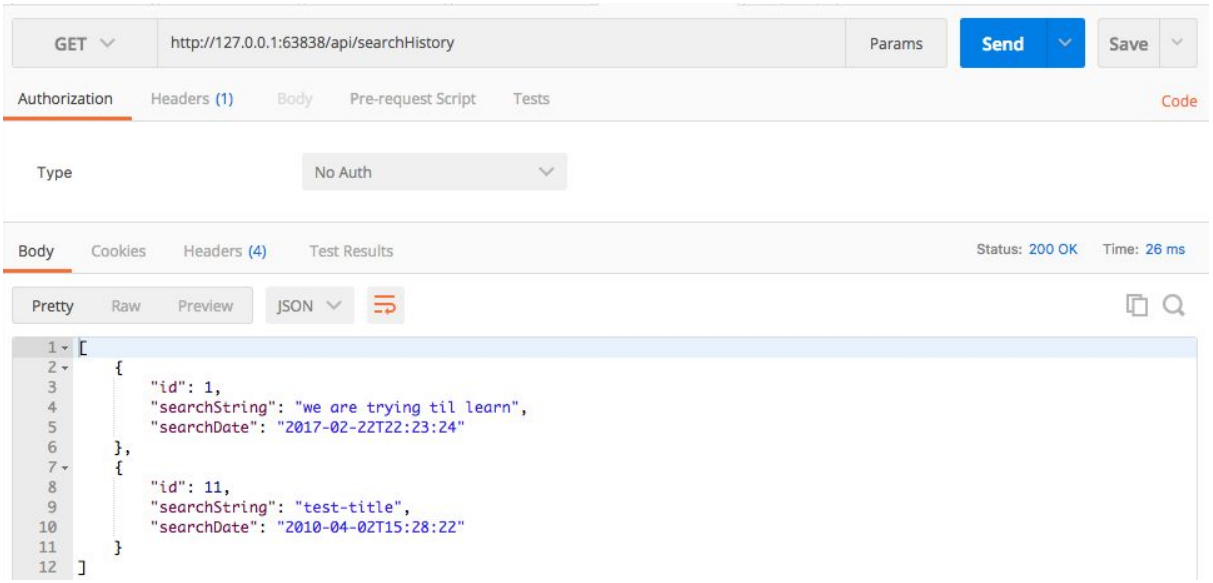
scenario 5. Update post (admin funcatinaly) :
Expected results:
Test result:
We need to work back on it as we couldn't update posts.

scenario 5. Add a new post :
Expected results:
Test result:
We need to work back on it as we couldn't create a post.

For the other Controllers (CommentsController, PostOwnerController, CommentUserController, MarkedController, NotesController and searchHistoryController), tests are the same and listed below:

Get (full list) :
Expected results: a list of all objects in the database.
Test result: All controllers can return a full list.

Exemple with searchHistory :



Get (by id) :
Expected results: a particular object in the database from its id.
Test result: All controllers can return a particular object from its id.

Post :
Expected results: Create a new object in the database
Test result: This part is not working on none of the controllers - need review

Put :
Expected results: Update object in the database
Test result:This part is not working on none of the controllers - need review

Delete :
Expected results: Delete an object from its id
Test result: All controllers can delete an object from its id - be careful while testing as post/put don't work yet.