

Notepad application

Table of Contents

- 1. Introduction
 - 1.1. Goals
 - 1.2. Technical overview
 - 2. Back end classes
 - 2.1. Controllers
 - 2.1.1. AboutController
 - 2.1.2. NoteController
 - 2.2. Services
 - 2.2.1. TwitterService
 - 2.2.2. NoteService
 - 2.2.3. UserActivityEventService
 - 2.2.4. UserActivityEventFactory
 - 2.3. Repositories
 - 2.3.1. NoteRepository
 - 2.3.2. UserActivityEventRepository
 - 3. Back end transactions
 - 3.1. Create note transaction
 - 3.2. Update note transaction
 - 3.3. Delete note transaction
 - 3.4. Get one note transaction
 - 3.5. Get all notes transaction
 - 4. Front end classes
 - 4.1. Class diagram
 - 4.2. Classes
 - 4.2.1. Class ApiClient
 - 4.2.2. Class AppComponent
 - 4.2.3. Class AppModule
 - 4.2.4. Class LoggingAspect
 - 5. Workflows
 - 5.1. Creating a note
 - 6. Front/Back end sequence diagrams
 - 6.1. Loading notes
-

1. Introduction

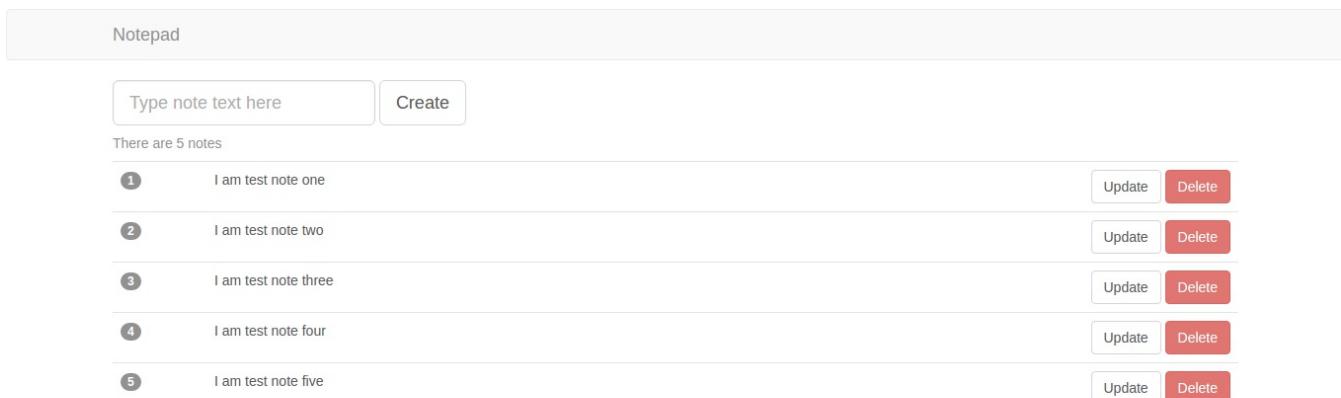
This document explains what Notepad application is and what it is not. The sections below describe the intentions behind Notepad application, boring technical details and fancy diagrams.

1.1. Goals

Making notes is crucial. It's a well-known fact that 98.37611% of the people who don't make any notes forget about things and miss opportunities. Notepad application addresses this problem by providing unbelievably ultimate functionality:

- Creating the notes
- Retrieving the notes
- Updating the notes
- Deleting the notes

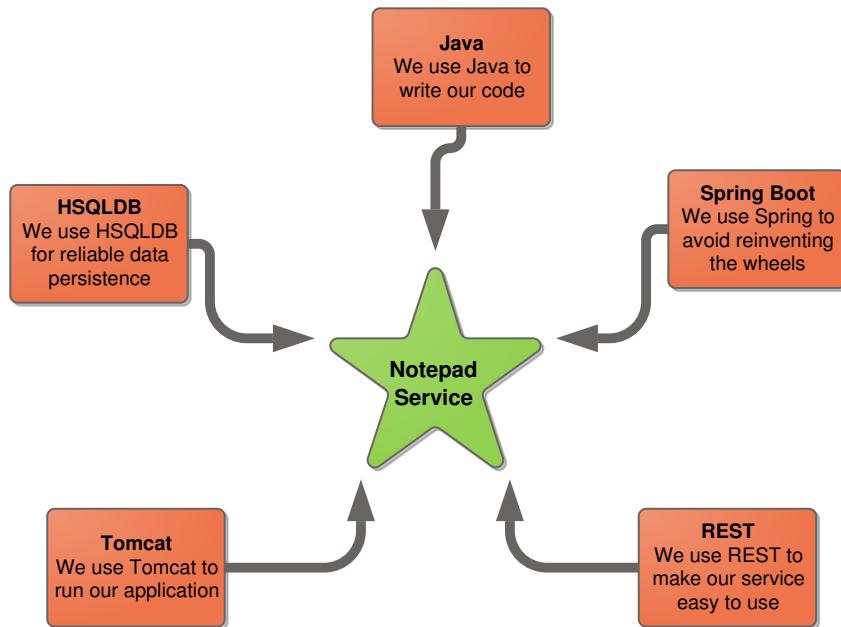
The Notepad application team firmly believes that our application makes the world better. Here's what the application looks like:



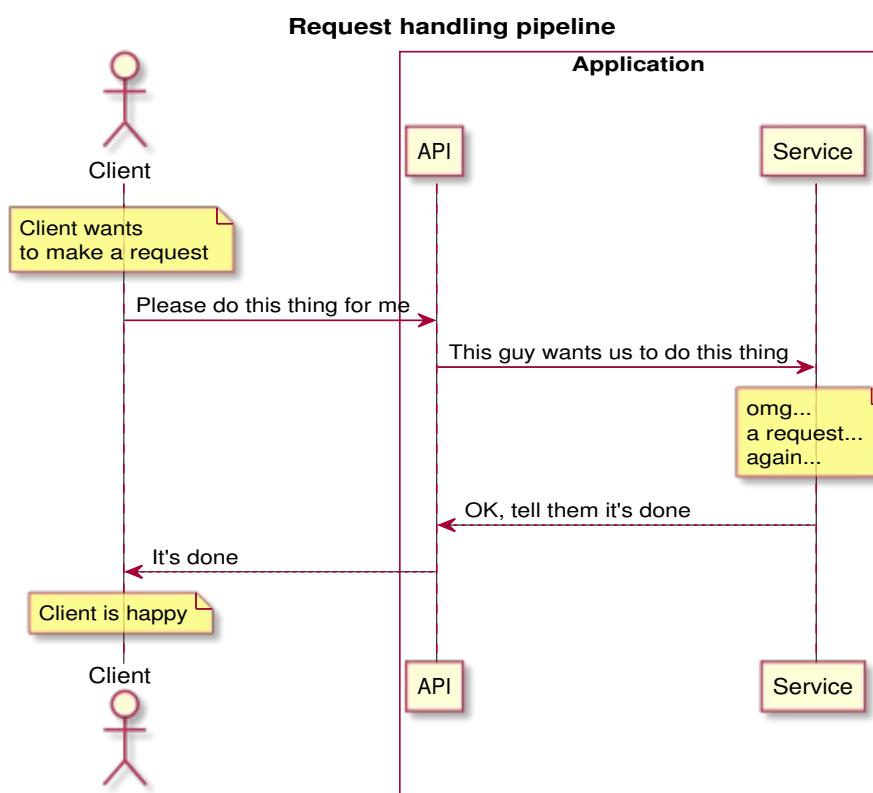
Isn't this application really nice?

1.2. Technical overview

Notepad application is a web application that exposes all its functionality via REST API. It uses Java and Spring as a foundation, and HSQLDB as a reliable production-grade data store.



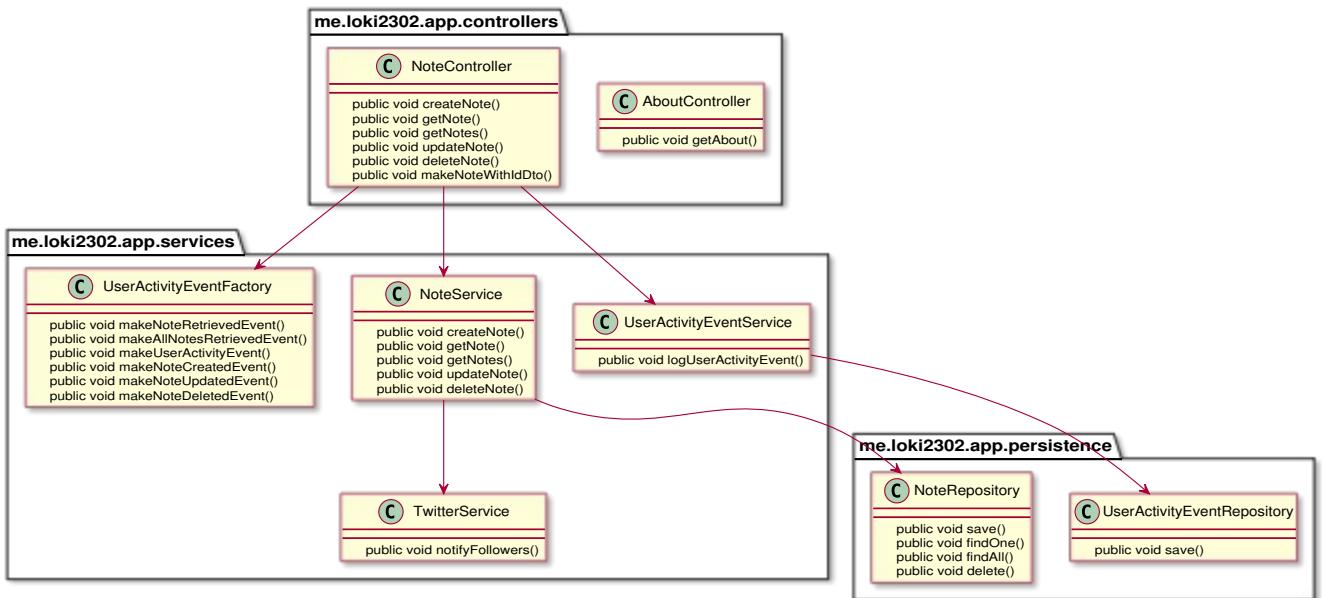
A typical request is handled like this:



2. Back end classes

This section describes how Notepad application is organized in terms of back end packages and classes. All classes are logically organized into packages:

- Controllers - API endpoints implementation.
- Services - business logic implementation.
- Repositories - data access implementation.



See how controllers never depend directly on data access layer.

2.1. Controllers

This section describes Notepad application API facade classes.

2.1.1. AboutController

Implements an "about" REST resource. Provides functionality to retrieve API details.

getAbout

```
public ResponseEntity getAbout()
```

JAVA

Provides API description and version

Returns: Response entity with API details

2.1.2. NoteController

Implements a "notes" REST resource. Provides functionality to create, retrieve, update and delete notes.

createNote

```
public ResponseEntity createNote(  
    NoteDto noteDto  
)
```

JAVA

Given all necessary note attributes, create a new note.

Name	Type	Description
noteDto	NoteDto	an object containing note attributes

Returns: a 201 response with Location header

getNote

```
public ResponseEntity getNote(  
    long noteId  
)
```

JAVA

Given a note ID, provide a note.

Name	Type	Description
noteId	long	a note ID

Returns: a 200 response with note attributes

getNotes

```
public ResponseEntity getNotes()
```

JAVA

Provide all notes.

Returns: a collection of all notes

updateNote

```
public ResponseEntity updateNote(  
    long noteId,  
    NoteDto noteDto  
)
```

JAVA

Given a note ID and all necessary attributes, update the note if it exists.

Name	Type	Description
noteId	long	a note ID
noteDto	NoteDto	an object containing note attributes

Returns: a 204 response

deleteNote

```
public ResponseEntity deleteNote(  
    long noteId  
)
```

JAVA

Given a note ID, delete a note.

Name	Type	Description
noteId	long	a note ID

Returns: a 204 response

2.2. Services

This section describes Notepad application service layer classes.

2.2.1. TwitterService

Twitter notification service

notifyFollowers

```
public void notifyFollowers()
```

JAVA

Notify Twitter followers about a very important event.

Returns: void

2.2.2. NoteService

Implements business logic for Notes.

createNote

```
public long createNote(  
    String text  
)
```

JAVA

Given note text, create a new note and provide its ID.

Name	Type	Description
text	String	a note text

Returns: a newly created note ID

getNote

```
public Note getNote(  
    long noteId  
)
```

JAVA

Get an existing note by ID

Name	Type	Description
noteId	long	a note ID

Returns: an existing note, or null if it does not exist

getNotes

```
public List getNotes()
```

JAVA

Get all notes

Returns: a list of all existing notes

updateNote

```
public Note updateNote(  
    long noteId,  
    String text  
)
```

JAVA

Update an existing note.

Name	Type	Description
noteId	long	an ID of note to update
text	String	a new text to update the note with

Returns: an updated note, or null if requested note doesn't exist

deleteNote

```
public void deleteNote(  
    long noteId  
)
```

JAVA

Delete a note by ID.

Name	Type	Description
noteId	long	an ID of note to delete

Returns: void

2.2.3. UserActivityEventService

A service that handles user activity events

logUserActivityEvent

```
public void logUserActivityEvent(
    UserActivityEvent userActivityEvent
)
```

JAVA

Log user activity event

Name	Type	Description
userActivityEvent	UserActivityEvent	a user activity event to log

Returns: void

2.2.4. UserActivityEventFactory

A factory that constructs instances of UserActivityEvent

makeNoteRetrievedEvent

```
public UserActivityEvent makeNoteRetrievedEvent(
    long id
)
```

JAVA

Construct a "note retrieved" event

Name	Type	Description
id	long	a note ID

Returns: an event instance

makeAllNotesRetrievedEvent

```
public UserActivityEvent makeAllNotesRetrievedEvent()
```

JAVA

Construct an "all notes retrieved" event

Returns: an event instance

makeNoteCreatedEvent

```
public UserActivityEvent makeNoteCreatedEvent(  
    long id  
)
```

JAVA

Construct a "note created" event

Name	Type	Description
id	long	a note ID

Returns: an event instance

makeNoteUpdatedEvent

```
public UserActivityEvent makeNoteUpdatedEvent(  
    long id  
)
```

JAVA

Construct a "note updated" event

Name	Type	Description
id	long	a noteID

Returns: an event instance

makeNoteDeletedEvent

```
public UserActivityEvent makeNoteDeletedEvent(  
    long id  
)
```

JAVA

Construct a "note deleted" event

Name	Type	Description
id	long	a note ID

Returns: an event instance

2.3. Repositories

This section describes Notepad application data access layer classes.

2.3.1. NoteRepository

A repository for Notes.

save

```
public Note save(  
    Note entity  
)
```

JAVA

Save or update a note.

Name	Type	Description
entity	Note	note to be saved

Returns: saved or updated not instance

findOne

```
public Note findOne(  
    Long id  
)
```

JAVA

Find a note by ID.

Name	Type	Description
id	Long	a note ID

Returns: a note with given ID or null

findAll

```
public List findAll()
```

JAVA

Find all notes.

Returns: a list of notes

delete

```
public void delete(  
    Long id  
)
```

JAVA

Delete a note by ID.

Name	Type	Description
id	Long	a note ID

Returns: void

2.3.2. UserActivityEventRepository

A repository for user activity events

save

```
public UserActivityEvent save(  
    UserActivityEvent entity  
)
```

JAVA

Save an instance of UserActivityEvent

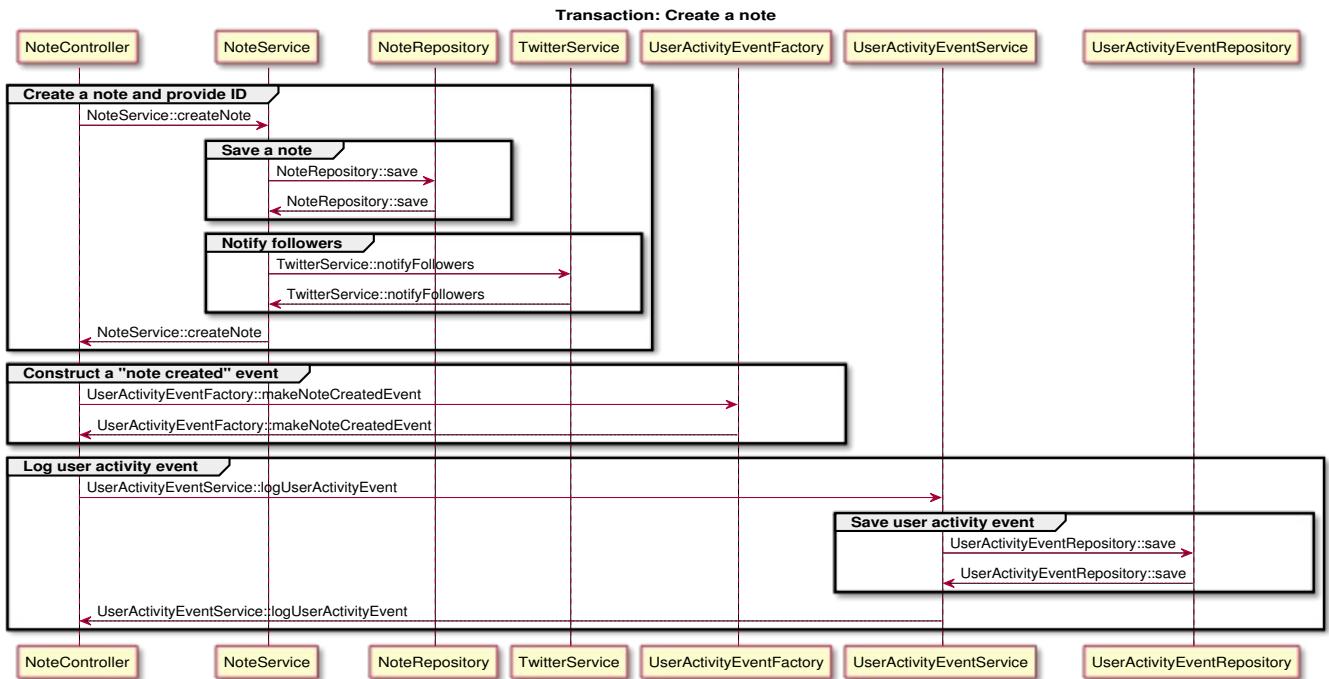
Name	Type	Description
entity	UserActivityEvent	an event to save

Returns: a saved instance of original event

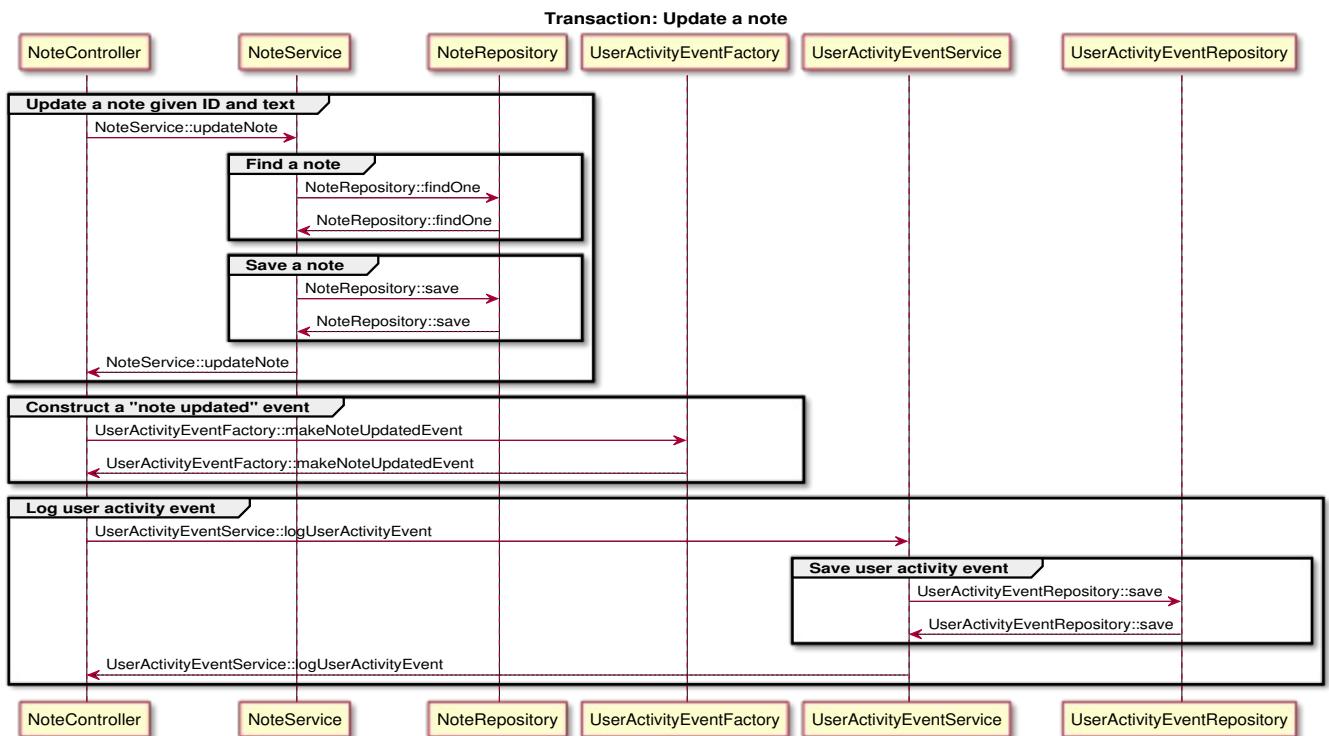
3. Back end transactions

This section describes Notepad application's transactions.

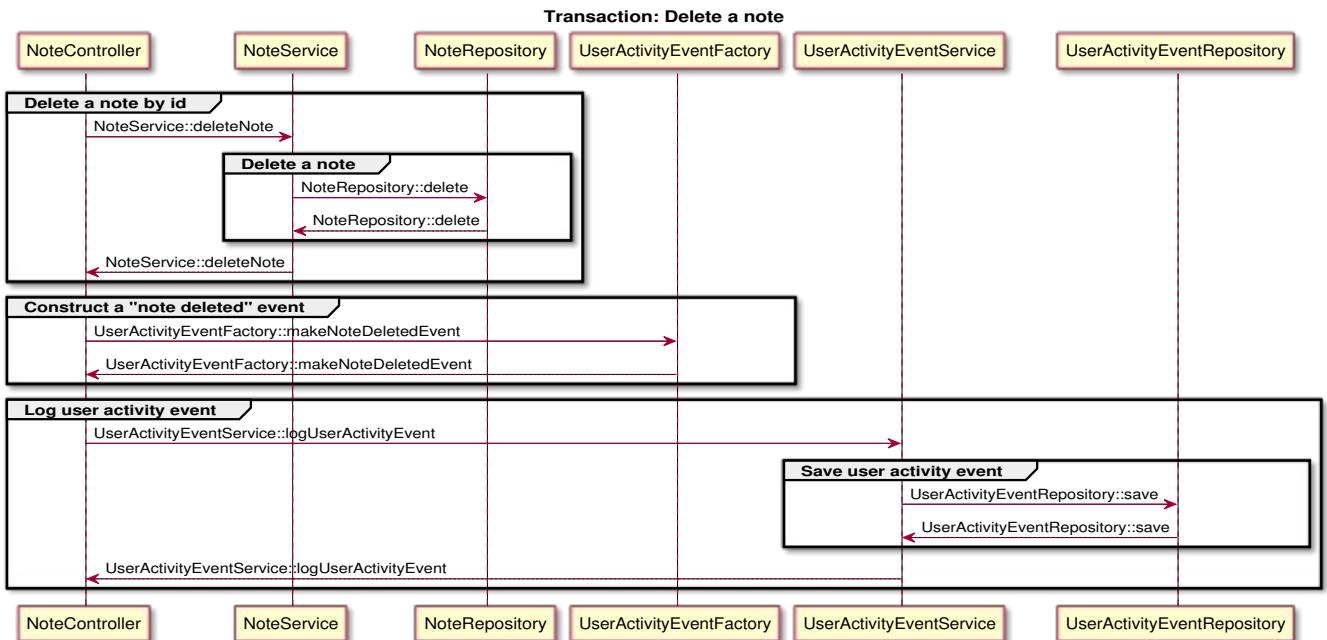
3.1. Create note transaction



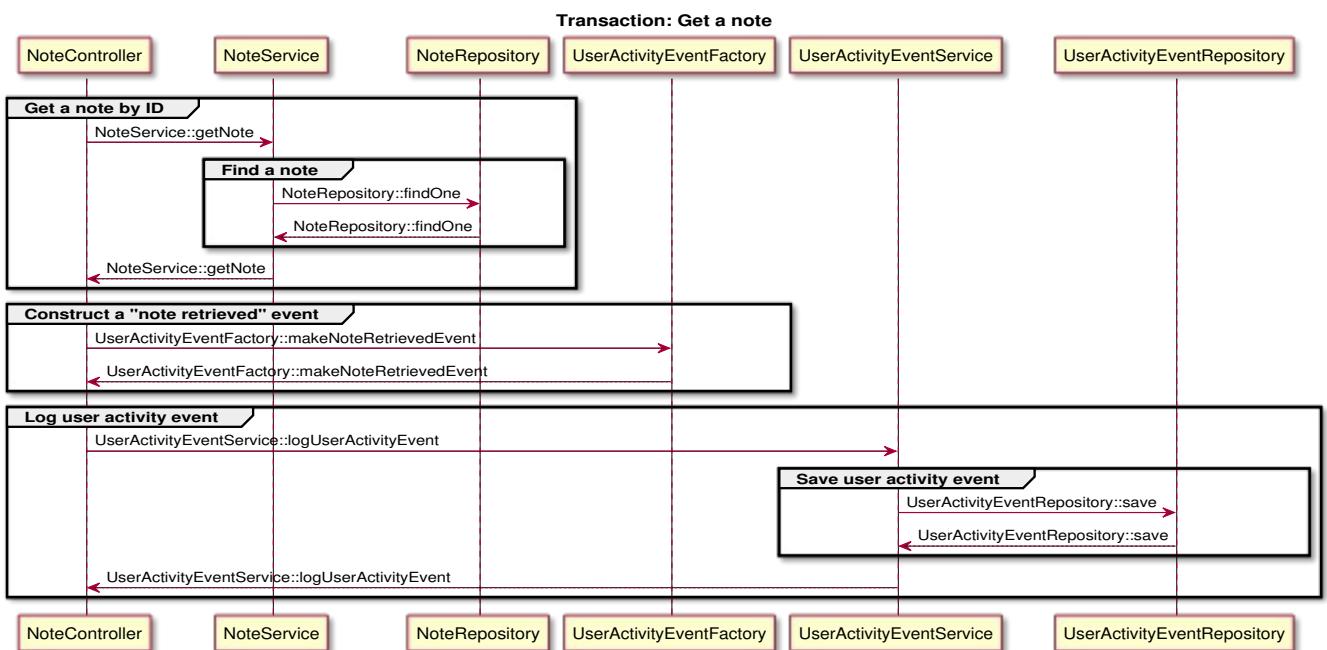
3.2. Update note transaction



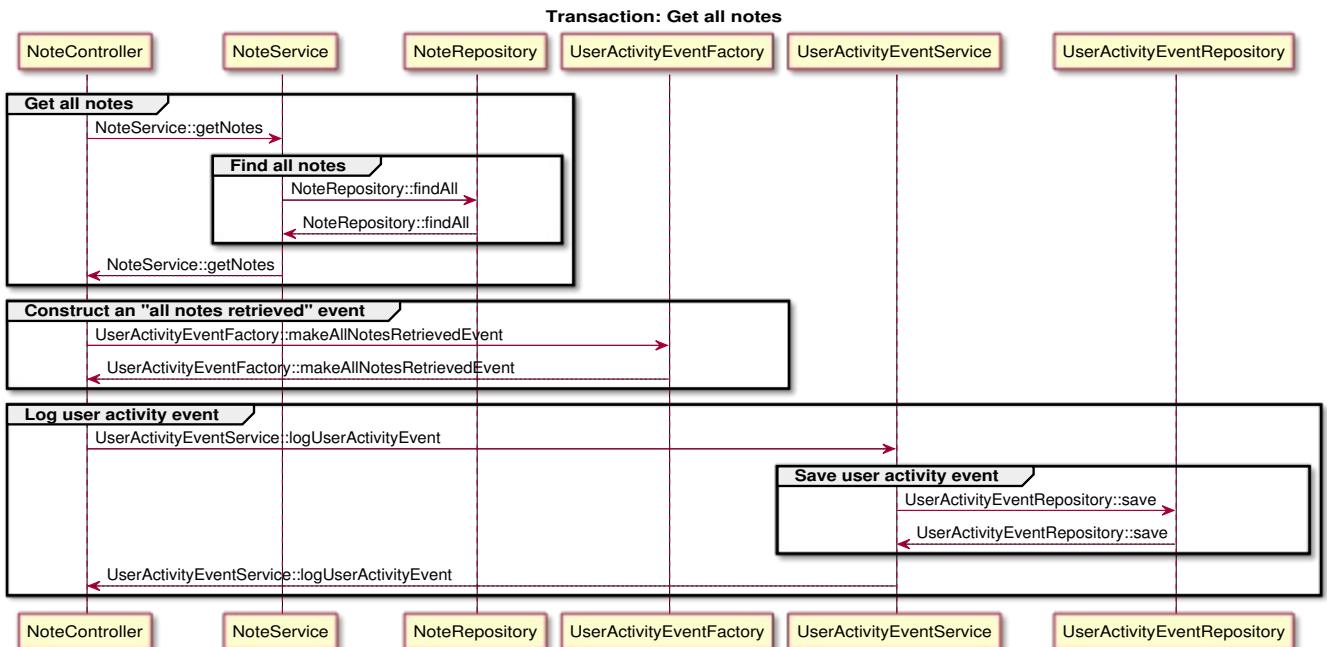
3.3. Delete note transaction



3.4. Get one note transaction



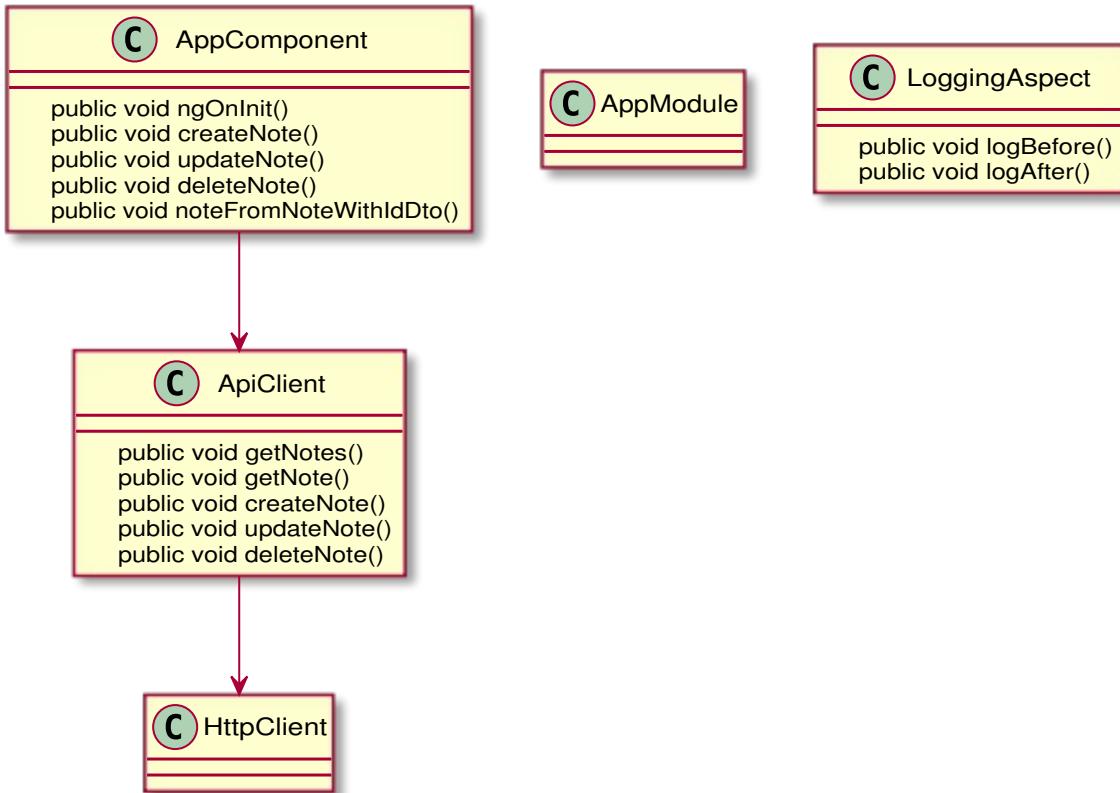
3.5. Get all notes transaction



4. Front end classes

This section describes how Notepad application is organized in terms of front end classes.

4.1. Class diagram



Is it not a nice diagram?

4.2. Classes

4.2.1. Class ApiClient

Notepad service API client.

getNotes

```
getNotes(): Promise<NoteWithIdDto[]>
```

JAVASCRIPT

Retrieve a list of all notes.

Returns: a collection of notes

getNote

```
getNote(  
  noteId: number  
) : Promise<NoteWithIdDto>
```

JAVASCRIPT

Get one note by ID.

Name	Type	Description
noteId	number	an ID of note to retrieve

Returns: a retrieved note

createNote

```
createNote(  
  noteDto: NoteDto  
) : Promise<NoteWithIdDto>
```

JAVASCRIPT

Create a note.

Name	Type	Description
noteDto	NoteDto	an object containing all

an object containing all mandatory note attributes

Returns: a created note

updateNote

```
updateNote(  
  noteId: number,  
  noteDto: NoteDto  
) : Promise<NoteWithIdDto>
```

JAVASCRIPT

Update an existing note.

Name	Type	Description
noteId	number	an ID of note to update
noteDto	NoteDto	an object containing updated note attributes

Returns: an updated note

deleteNote

```
deleteNote(  
  noteId: number  
) : Promise<void>
```

JAVASCRIPT

Delete an existing note.

Name	Type	Description
noteId	number	an ID of note to delete

Returns: Promise<void>

4.2.2. Class AppComponent

A component that represents the one and only application page.

ngOnInit

```
ngOnInit(): Promise<void>
```

JAVASCRIPT

Initialize - retrieve a collection of all notes.

Returns: Promise<void>

createNote

```
createNote(): Promise<void>
```

JAVASCRIPT

Create a note based on user input.

Returns: Promise<void>

updateNote

```
updateNote(  
  noteId: number  
) : Promise<void>
```

JAVASCRIPT

Update a note based on user input.

Name	Type	Description
noteId	number	an ID of note to update

Returns: Promise<void>

deleteNote

```
deleteNote(  
  noteId: number  
) : Promise<void>
```

JAVASCRIPT

Delete a note based on user input.

Name	Type	Description
------	------	-------------

noteId	number	an ID of note to delete.
--------	--------	--------------------------

Returns: Promise<void>

noteFromNoteWithIdDto

```
noteFromNoteWithIdDto(  
    noteWithIdDto: NoteWithIdDto  
)
```

JAVASCRIPT

Construct a note from note DTO.

Name	Type	Description
noteWithIdDto	NoteWithIdDto	a note data transfer object

Returns: a constructed note

4.2.3. Class AppModule

Represents the main application module.

AppModule does not have any methods.

4.2.4. Class LoggingAspect

logBefore

```
logBefore(  
    metadata: Metadata  
)
```

JAVASCRIPT

Name	Type	Description
metadata	Metadata	

Returns: void

logAfter

```
logAfter(  
  metadata: Metadata  
)
```

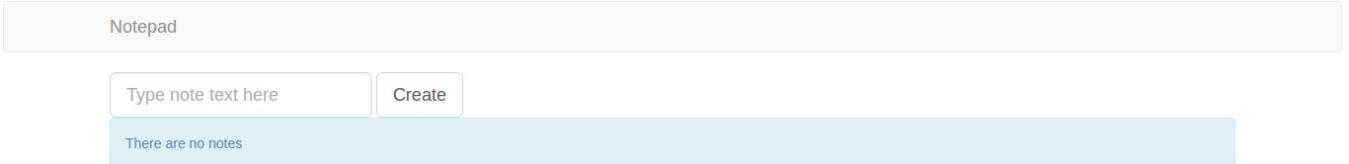
Name	Type	Description
metadata	Metadata	

Returns: Promise<void>

5. Workflows

5.1. Creating a note

Making notes is crucial. Let's get started with an empty Notepad application, where there are no notes at all. See how empty it is. It is up to us to create the very first note.



To do so, we first type note text into a text box:



After it, we click the "Create" button. And once we do that, a new note should appear on the list of notes:

Notepad

Type note text here

Create

There are 1 notes

1

My first note

Update

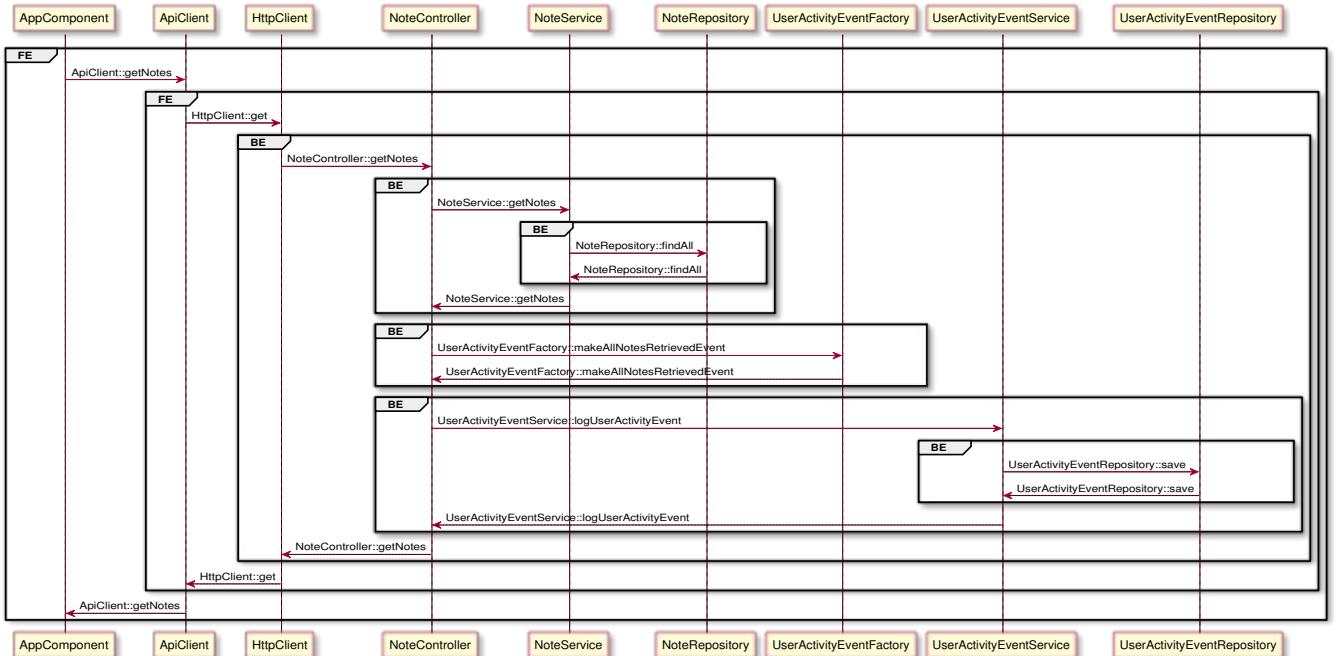
Delete

Done.

6. Front/Back end sequence diagrams

6.1. Loading notes

The following describes how "load notes" transaction works on both back- and front-end.



See how front end initiates the transaction and back end provides the data to front end.

The End.

Last updated 2016-10-25 23:38:17 -04:00