December 4, 2024

NetNote Backlog

We are building a novel, distributed *note taking* application that runs in a client/server setting.

1 Stakeholders

User: A user of the *client* application that uses NetNote to organize notes.
Admin: An operator of a NetNote *server* application that allows others to host their notes on a central system.

Note The application does not provide any specific functionality for admins. We assume that servers can be operated by individuals in a decentralized way. We only distinguish *Users* and *Admins* in some user stories for clarity.

2 Terminology

Note: A note is a data structure that represents a plain-text string. The content of a note does not *need* to follow a particular structure, but it *should be possible* to use Markdown syntax, which allows a rich-text rendering.

Collection: A collection is an abstract set of *Notes* that are stored on a server. As such, a collection can be referred to via a base URL of a server, e.g., `https://server.nl/netnote/`, combined with a unique name like `sebs-notes`. The complete URL in this case would be `https://server.nl/netnote/sebs-notes/`

Server: A server is an instance of the NetNote backend component. It is running on a network device that is reachable from other devices. One server can host many *collections*.

Client: A client is the interface that is started by a *user* and connected to one or more *servers*, with the intention to maintain one or more *collections* of *notes*.

3 Non-Functional Requirements

- The application is built with Spring/JavaFX
- The application should not require the implementation of a user account system
- The server does not store any *client-specific* information. In practice this means that a client maintains a local configuration file for specific configuration options.
- Client/Server communication is performed via HTTP and JSON through REST or websockets.
- Client/Server have shared data structures and use Jackson for automated object-mapping to/from JSON.
- It must be possible to connect at least 10 clients simultaneously to the server

4 Epics and User Stories

4.1 Basic Requirements

Note: All(!) basic requirements must be met to pass the course.

As a user, I want ...

- To host my notes on a server, so I can use my notes from multiple clients.
- To see all existing notes on the server, so I can browser the available information.
- To create new notes, so I can persist information for later use.
- To add titles to note, so I can organize my information.
- To change note titles, so I can keep title and content in sync.
- To delete notes, so I can remove unneeded information.
- To sync every change in my the notes automatically with the server, so I do not need to manually save.
- To write note contents as free text, so I am not hindered by any structural requirements.
- To manually refresh my client view, so I see new information on the server.
- To search for keywords, so I can find notes with matching titles or content.
- To be able to use Markdown formatting in my notes, so I can write structured notes.
- To see rendered version of the Markdown note, so I can see a formatted version of my note.
- **Update:** ~~To see a short summary of any problem during the Markdown rendering, so I can fix the problem.~~
  (Removed as a requirement. We meant to catch `Exceptions` in the processing, which could be caused by the Markdown library or also by your own code. In case of an error, the 'WebView' should have shown the *error message* of the `Exception`.)
- To see an automatic update of the rendered view, so I see my changes reflected in real time.
- **Update:** To be prevented from using a duplicate note title, so note content is clear.
  (Note titles should be unique per collection, like a filename has to be unique in a folder.)

Non-Functional Requirements

- The basic version of the application only need to work with a single collection, which can be *hard-coded* in a configuration file that is read during application start-up.
- The client should fully support →basic Markdown syntax. Use →commonmark-java or →flexmark-java to parse Markdown and generate a (temporary) HTML file.
- The visualization in the client should be achieved via an JavaFX `WebView` pane
- The `WebView` should use a local .css file, if available, that allows to adjust the format of the rendered page.

Additional Information

- To optimize performance, not every keystroke should be synced between client and server. Accumulate changes and only sync after a change threshold (e.g., 10 characters) or a timeout (e.g., 5 seconds) is reached. Any optimization for the syncing is optional though and will not be considered in the assessment.

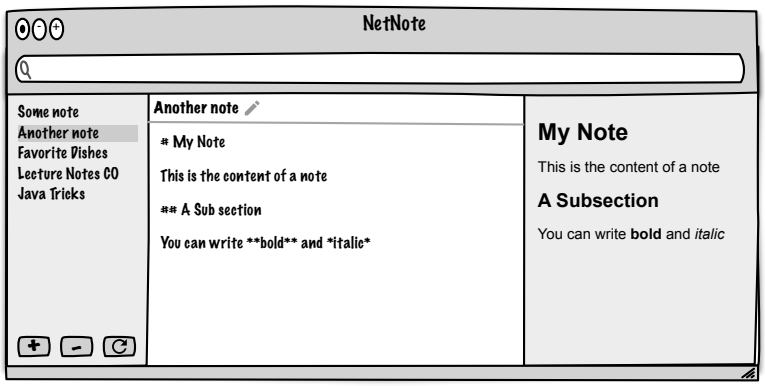

Figure 1: Basic Overview

4.2 Multi-Collection

As a user, I want ...

- To distribute my notes across several collections, so I can organize my notes better.
- To set a collections filter, so I only see notes that exist in that collection.
- To select *all* collections at once, so I see all notes that I have access to.
- To edit the list of collections (add/delete/change), so I can control which note collections I see in my client.
- To use a user-friendly name for my collections, so I can use short and descriptive names.
- To see the server status while I create/edit a collection, so I can spot typos.
  (At least the following states are supported: server not reachable, collection will be created, collection already exists.)
- To define a *default collection*, so new notes will always be added to this collection.
- To create new notes in a specific collection, when it is the only one shown, so I do not have to move notes.
- To change the collection for a note, so I can move notes from one collection to another.

Non-Functional Requirements

- The configured collections are stored in a local config file, so they are persisted across restarts.

Additional Information

- Do not write/parse the structure of the local config file yourself.
- Create a `Config` data structure and use the Jackson mapper to convert the object to/from a JSON string.
- Look into the `FileUtils` of the →Apache Commons IO library to understand how to read/write string to a `File`.

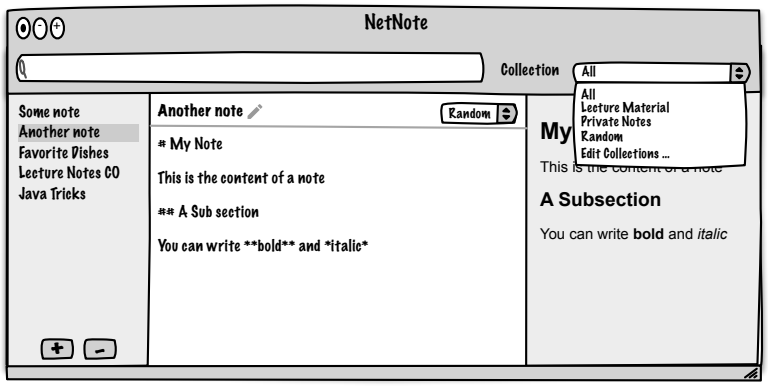

Figure 2: Overview with multiple collections

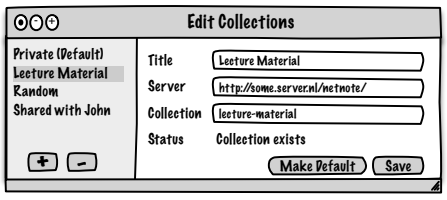

Figure 3: Edit Collections

4.3 Embedded Files

As a user, I want ...

- To embed files into notes (e.g., images), so I can add relevant media to my descriptions.
- To rename file names, so I can give them more descriptive names without re-upload.
- To delete files, so I can remove irrelevant information.
- To delete all embedded files when I delete notes, so the server does not accumulate irrelevant files.
- To refer to embedded images in my Markdown, so can see images in my preview.
  (The links to embedded files need to be processed in the generated HTML to point to the right server location)
- To download files when I click on them, so I can use the files locally on my computer.

Non-Functional Requirements

- In principle, the client does not need to store any local data.
- The notes should be requested on demand and only stored in memory.
- Embedded files are available on dedicated URLs on the server that are connected to the note (e.g., a URL like `http://server/netnote/files/notes/My%20Note/foo.jpg` represents a file `foo.jpg` in a note `My Note` of collection `notes`. URL encoding is used for special characters.
- Clients should only receive a simplified abstraction of the meta data, like *file name*, *link*, or *file type*.

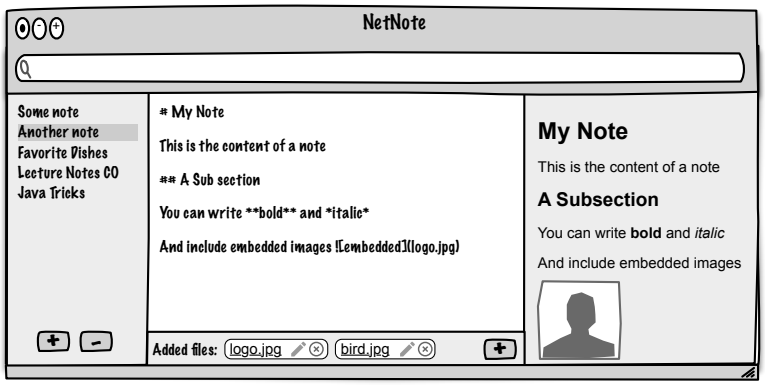

Figure 4: Embedded files

4.4 Interconnected Content

As a user, I want ...

- To use tags in the form of `#foo` in my notes, so I can organize information and make it easier to find.
  (Make sure to identify/process/replace the tags first, or the Markdown parser will interpret them as titles)
- To use `[[other note]]` to refer to other notes in the same collection by title, so I can link related notes.
- To automatically replace all `[[..]]` references when I rename a note, so the references stay consistent.
- To have these tags/references be converted to links in the render view, so I can easily click on them.
- To click on a tag, so I can filter all notes to those who have the tag.
- To click on the link to another note, so it is easy to switch to the other note.
- To see a visual difference in the preview if a referenced note does not exist, so I can spot typos.
- To select multiple tags, so I can filter notes with multiple criteria.
- To clear all selected tags, so I can easily go back to seeing all notes.

Non-Functional Requirements

- It should be possible to filter for an unlimited amount of tags, i.e., not just a fixed number of tag filters.
- Adding subsequent tags should only list tags in the dropdown that are actually available in the remaining notes.

Additional Information

- Replace tags and note links in the rendered HTML pages with custom HTML links or buttons. The `WebView` allows to add listeners that allow to react to interactions with HTML elements.

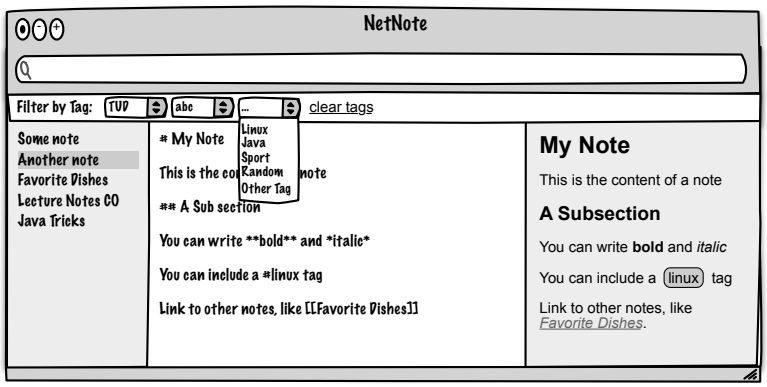

Figure 5: Client with tags, a tag browser, and internal links

4.5 Automated Change Synchronization

As a user, I want ...

- To see that *some changes* (see later) are auto-propagated across all clients, so I do not have to manually refresh.
- To see a change of a title to be propagated, so no manual refresh is required.
- To see the addition/deletion of notes to be propagated, so no manual refresh is required.
- To see any change in note content to be propagated when I view the same note, so no manual refresh is required.
- To see any change in note content also to be propagated to the preview, so text and preview are consistent.

Non-Functional Requirements

- The client uses *web sockets* to subscribe for changes.
- The client *does not poll* for changes, changes are pushed from the server.
- The client only received relevant updates, i.e., no changes for unconnected collections or no content updates for notes that are not currently being viewed.

Additional Information

- Synchronizing and handling conflicts in simultaneous *edits* is *really* difficult.
- Your application should not crash when multiple users are simultaneously editing, but making this scenario collision-safe and performant is out of scope for the CSE project. Our strong recommendation: do not even try to do more than a basic implementation, we also will not consider any optimization in the assessment.

4.6 Live Language Switch

Add support for switching the language of your client during runtime. The language selection should be persisted in the config file.

As a user, I want ...

- To see a language indicator, so I know which language is currently configured in my client on a first glance
- To see a flag icon as the language indicator, so I do not have to read additional text.
- To see all available languages through clicking the indicator, so I can find my preferred one easily
- To persist my language choice through restarts, so I do not have to pick a language each time

Non-Functional Requirements

- Should be implemented via →JavaFX Internationalization. This involves creating localized property files in the resources folder and calling →FXMLoader.setResources.
- The application must fully support at least English and Dutch. A third language must be added as a proof of concept, but can be made up.
- Please note that languages that do not flow left-to-right are much harder to integrate. While we encourage you to think about this use case, it is enough to limit your localization support to left-to-right languages.

Additional Information

- All labels in the application should match the selected language. It is not required to translate notes.

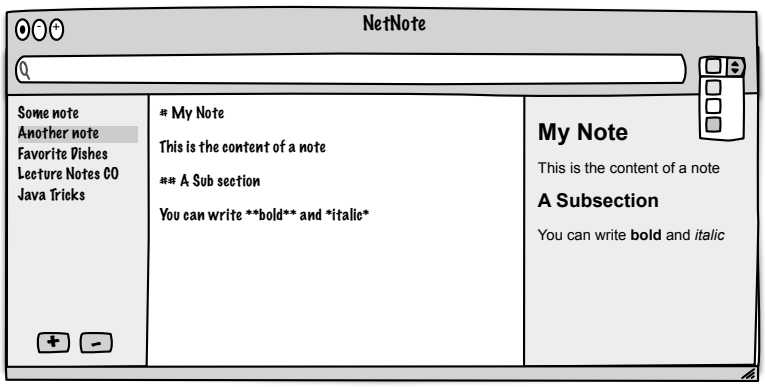

Figure 6: Language Switch