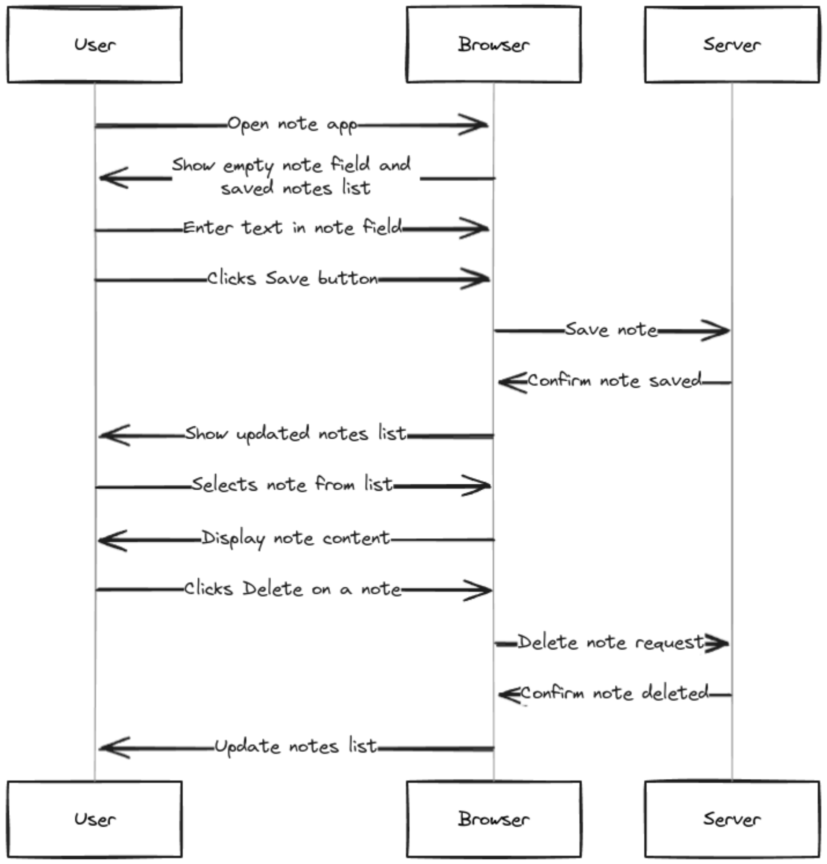
1. Describe high level design

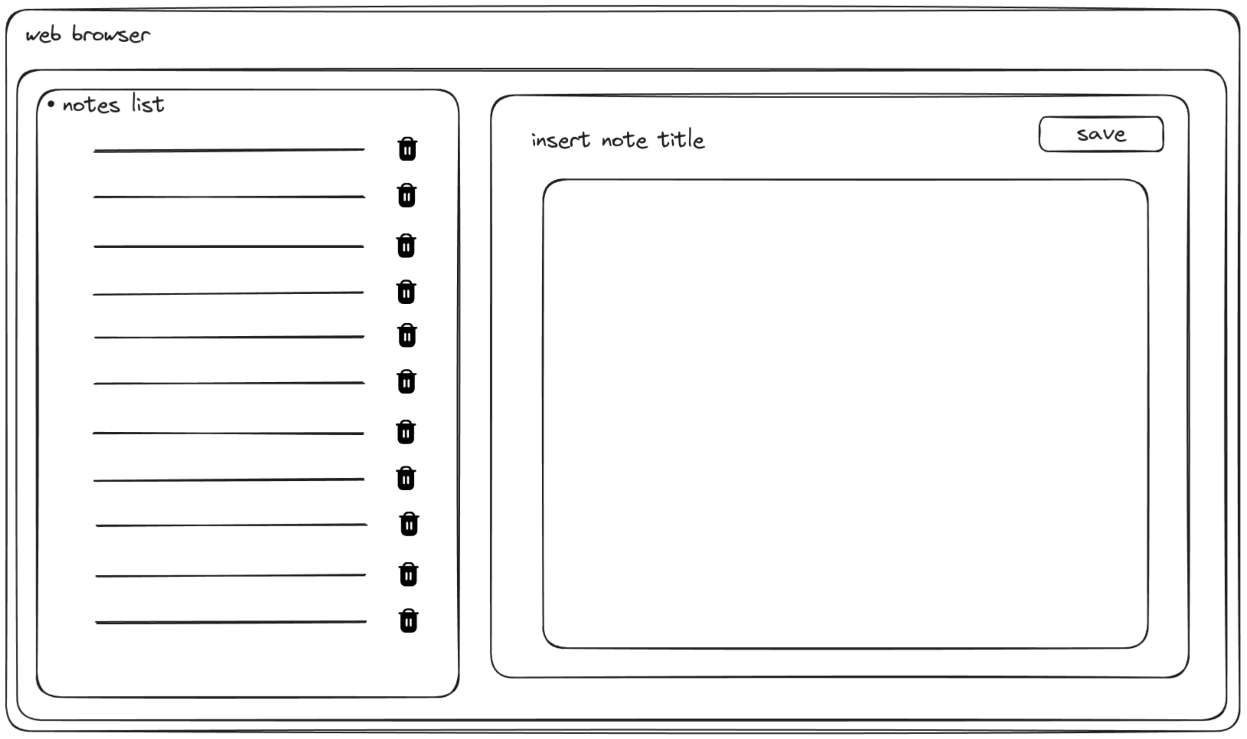
Show the main note app components and the logical interactions that will fulfill the requirements.



A flow to this app can be described in those steps, a user access a browser, open the note app, makes actions that will be sent to the server, the server computes, stores and return a response.

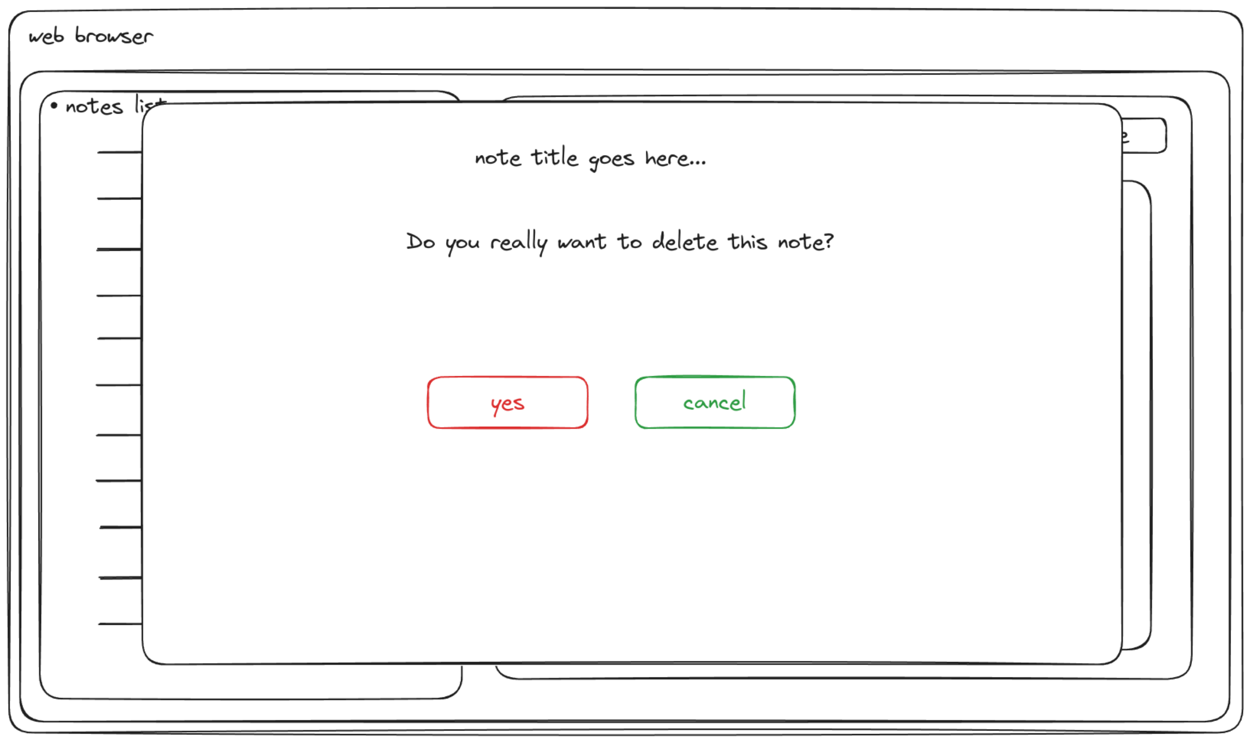
2. Web App API

Provide a wireframe design of the note web app that will fulfill the requirements.



Consider what UI components are required and how these interact with the other components.

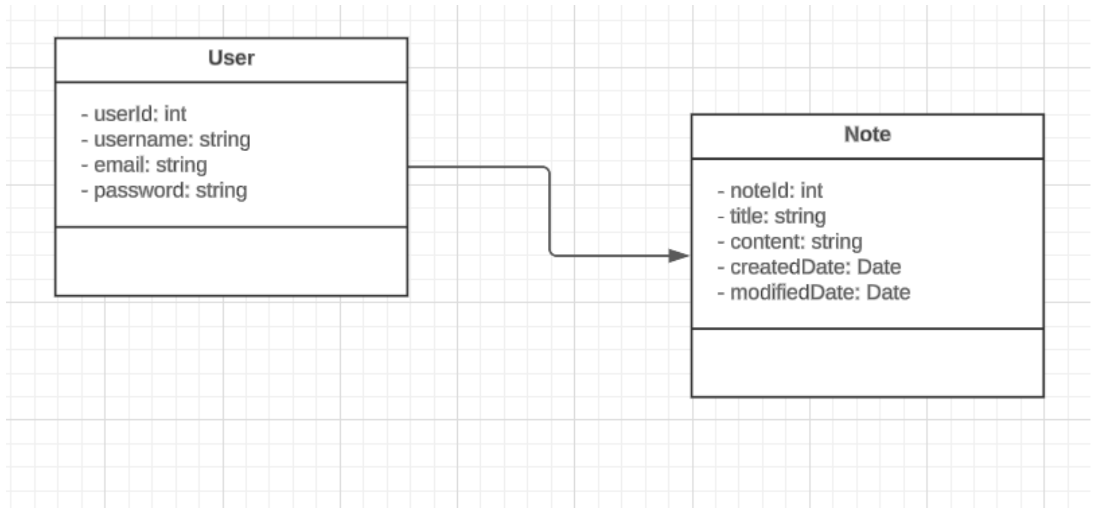
- We can have 3 basic components, a component “Notes List”, which handles the responsibility to list and manage the notes that can be deleted, a insertion component which handles the responsibility to insert notes content and save, sending the request to the server, and a delete component which can double check if the user really wants to delete a note



What (if any) validation is required?

- A crucial validation could be a component to double check the intention of the user to delete a note.

3. Data Model



Describe how a note will be modeled

- To have a model to this solution, we can have a user that have notes, a note consist into a id, title, content, createdDate and modifiedDate

4. Restful API

Describe the Restful API required to fulfill the note app.

- How would the web app get the user's notes?
  - The web application will send an HTTP GET request to the backend.
    - The request will include the user's OAuth access token attached as a header.
  - The backend will read the request and extract the user's id from the OAuth token.
  - The backend will fetch all notes associated with the userId, sorted by modifiedDate from the database.
  - The backend will return these notes in the body of an HTTP Response with a status code of 200.
  - The frontend will receive the response and display the notes to the user.
- How would the web app save a user note?
  - The web application will send an HTTP POST request to the backend.
    - The request will include the user's OAuth access token attached as a header.
    - The request will include the NoteDto JSON object in the body.
  - The backend will read the request and extract the user's id from the OAuth token.
  - The backend will create the note with the specified note.setId({userId}) (where userId is stored in the OAuth token) in the database.
  - The backend will return the newly created note in the body of an HTTP Response with a status code of 201.
  - The frontend will receive the response and display the updated user's notes.
- What are the URLs for the note resource(s)? What are the verbs to expose the actions?

- GET /notes/{userId} - to fetch all user notes.
- POST /notes - to create a new note for the user.
- DELETE /notes/{note-id} - to delete a note for the user.

5. Web Server

Describe how the webserver implements that Restful API:

- Consider how each action will be implemented.
  - Get all notes:

```
@GetMapping("/notes/{userId}")
public ResponseEntity<List<NoteDto>> getAllUserNotes(final HttpServletRequest
request) {
    final var userId = getUserIdFromOAuthToken(request.getHeader("Authorization"));
    return ResponseEntity.ok(noteService.getAllUserNotes(userId));
}
```

  - Create new note:

```
@PostMapping("/notes")
public ResponseEntity<NoteDto> createNote(@RequestBody @Valid final NoteDto
noteDto,
                                          final HttpServletRequest request) {
    final var userId = getUserIdFromOAuthToken(request.getHeader("Authorization"));
    NoteDto createdNote = noteService.createNote(userId, noteDto);
    return ResponseEntity.status(HttpStatus.CREATED).body(createdNote);
}
```

- Delete note:

```
@DeleteMapping("/notes/{noteId}")

public ResponseEntity<Void> deleteNote(@PathVariable("note-id") @NotNull final
Long noteId,

                                       final HttpServletRequest request) {

    final var userId = getUserIdFromOAuthToken(request.getHeader("Authorization"));

    noteService.deleteNote(userId, noteId);

    return ResponseEntity.ok().build();

}
```

What (if any) business logic is required?

- One crucial example is ensuring that a user cannot delete another user's notes:

```
@Service

class NoteService implements INoteService {

    // ...

    @Override

    public void deleteNote(final Long userId, final Long noteId) {

        final var note = noteRepository.findById(noteId)

                                       .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND));

        if (!note.getUserId().equals(userId)) {

            throw new ResponseStatusException(HttpStatus.FORBIDDEN);

        }
        noteRepository.deleteById(noteId);
    }

    // ...
}
```

How are the notes saved?

- When a NoteDto object is received by the controller method createNote(...), it is first validated.
    - If validation fails, an HTTP Status 400: Bad Request is returned.
- The validated NoteDto object is then sent to the NoteService class along with the userId extracted from the OAuth token.
    - The NoteDto object is mapped to a Note object, with the userId field set to the userId from the OAuth token.
    - The noteRepository.save(note) method is invoked to save the note in the database.
    - The saved note is mapped back to a NoteDto object.
    - The NoteDto object is returned.
- The controller method returns the NoteDto object in the body of the response with HTTP Status Code 201: Created to the client (our frontend).