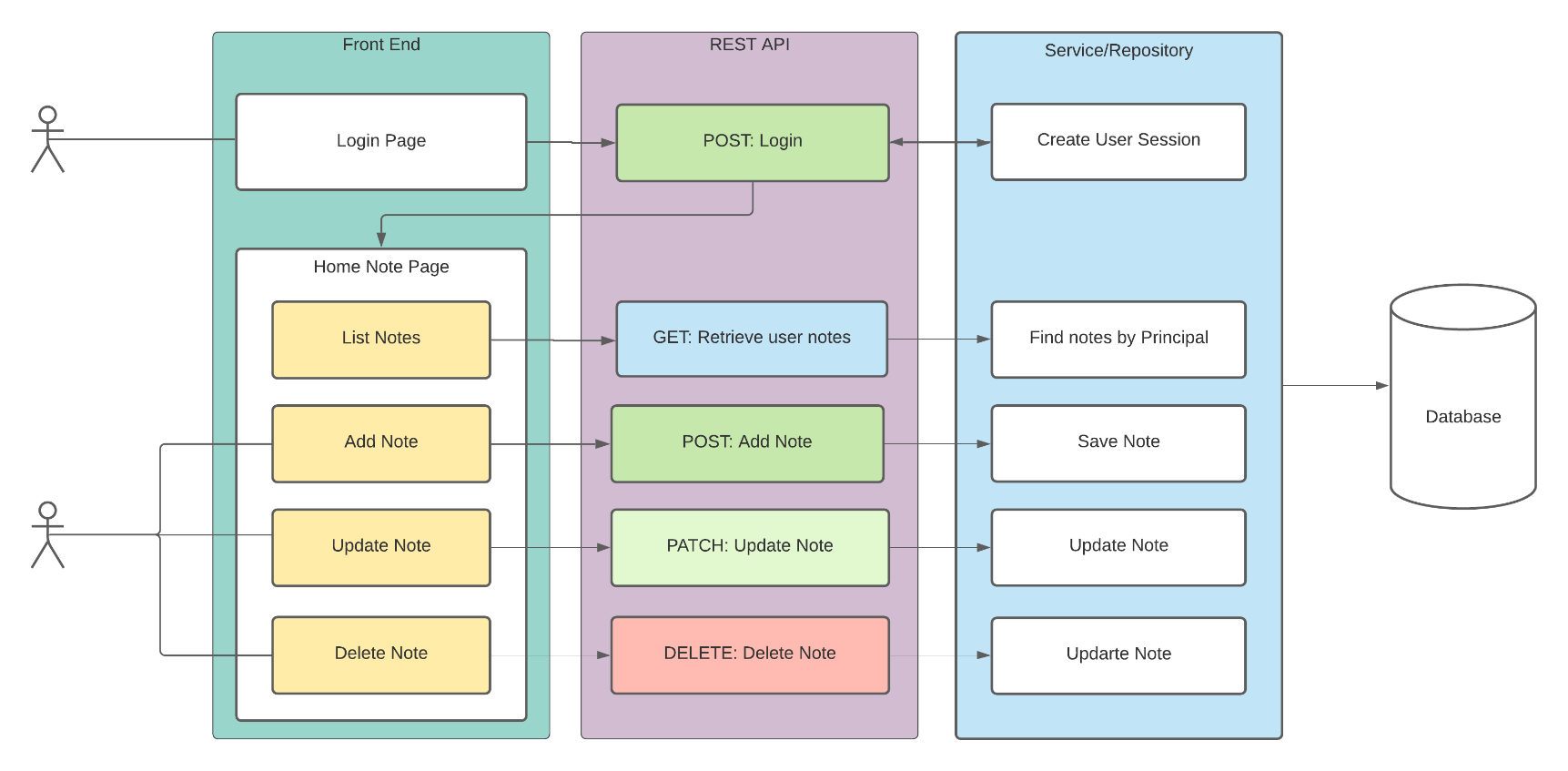
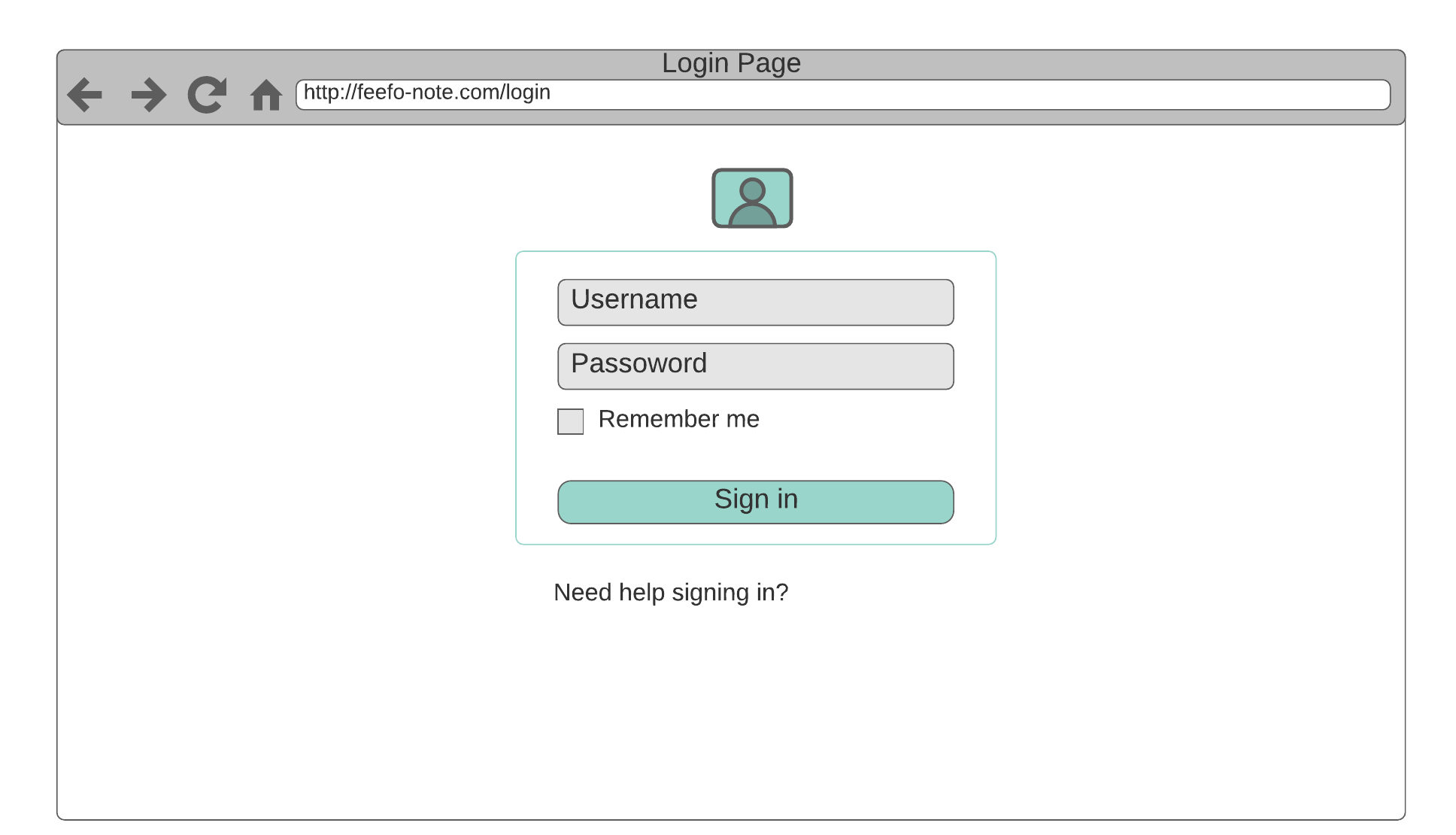
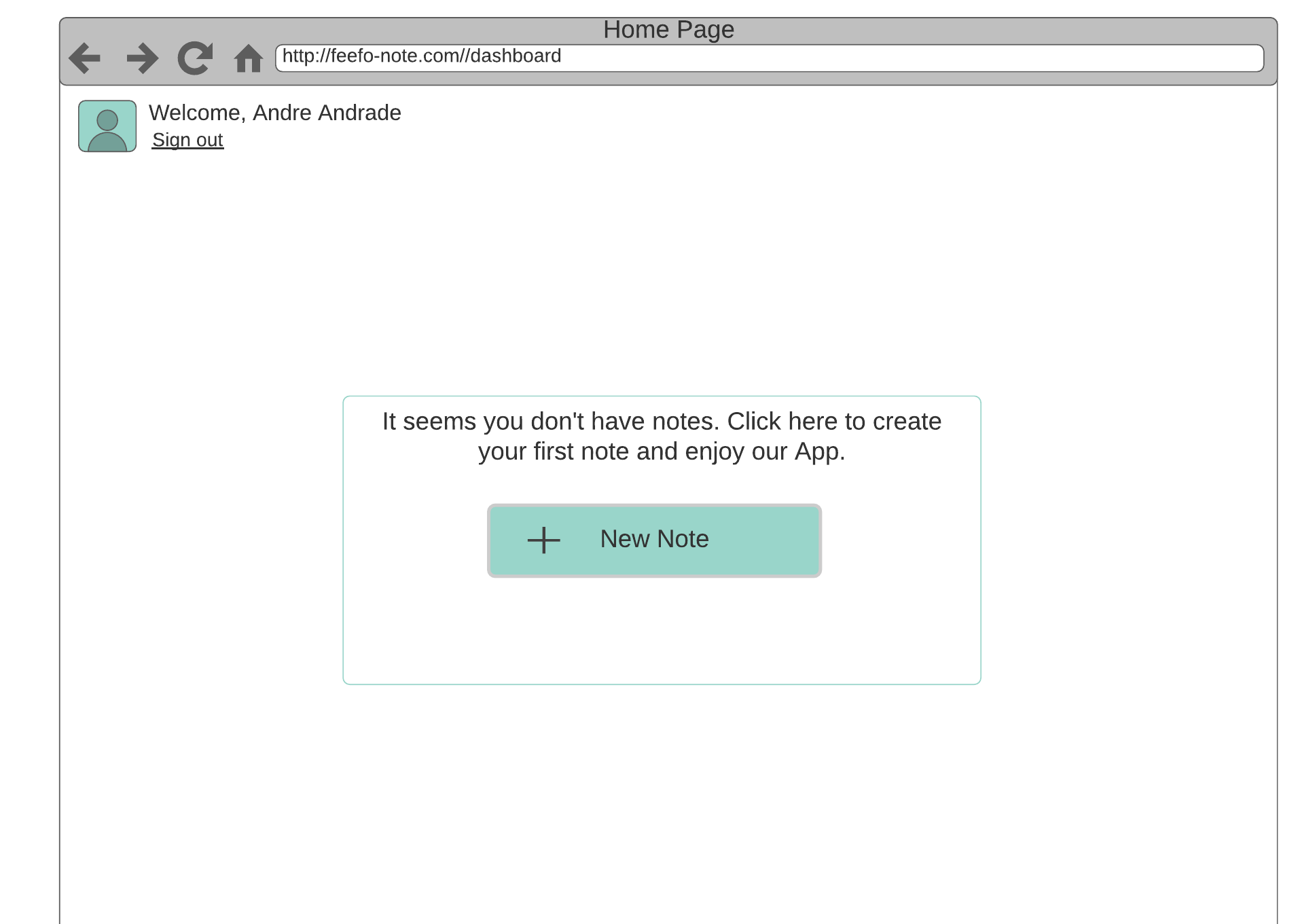
1. **High level design**

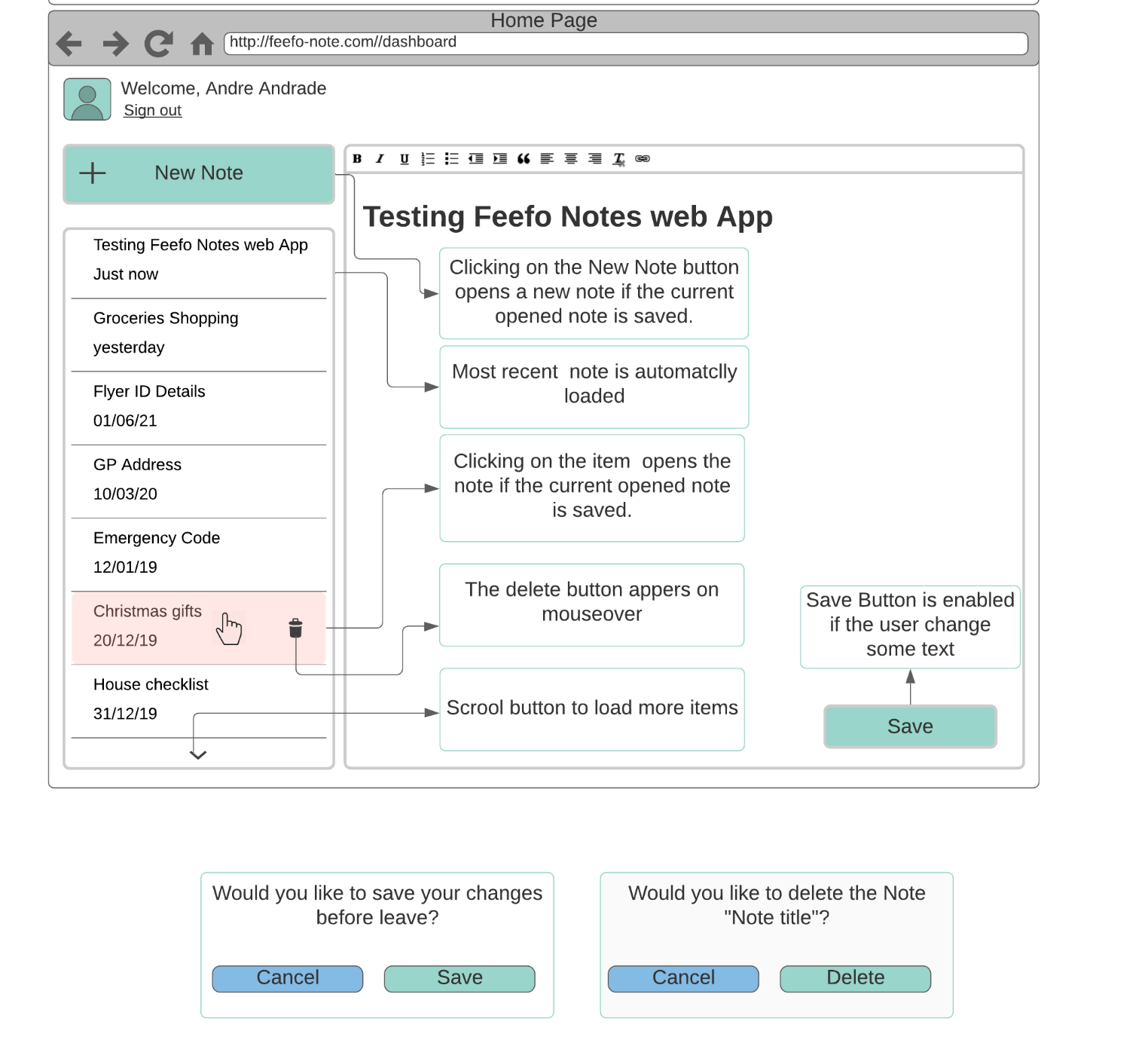


1. **Web App UI**
   1. Login

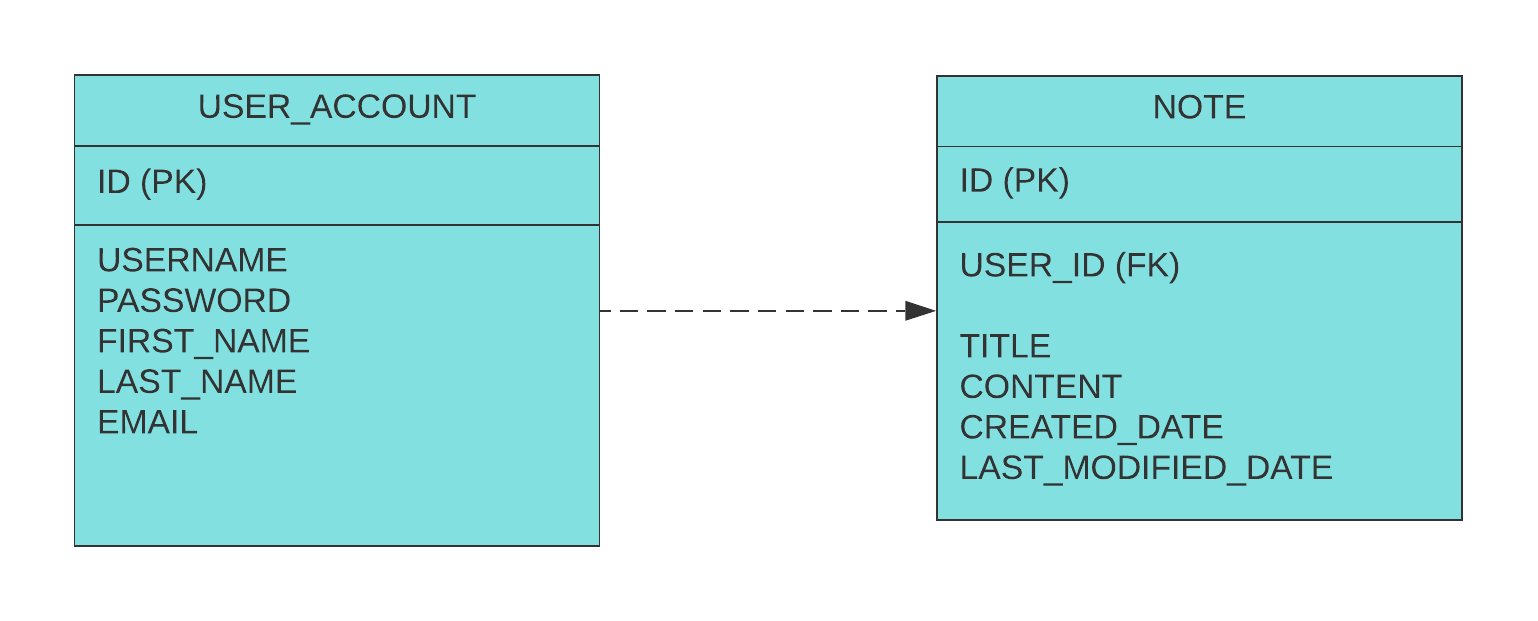


* 1. Empty list



* 1. List

1. **Data Model**

****

1. **Restful API**

* Retrieve user notes
  + GET: {domain}/api/note
    - Find Note by user. Grab the Principal on the session
    - Order by last modified date
    - If the user is not logged the web server should return an authorized exception
  + Request:
    - Header: Sends the JWT on the Authorization Header
    - Query Parameters: size (default 10) page (default 0)
  + Response:

{

"notes": [

{

"noteId": "866ee0be-5d13-41a8-8d67-c7d91f669cd7",

"title": "Groceries Shopping",

"content": "Groceries Shopping: tomatos, meat, milk",

"lastModifiedDate": "12/02/2021T10:12:00",

"CreatedDate": "02/02/2021T10:12:00"

}

]

}

* Add Note
  + POST: {domain}/api/note
    - If the user is not logged the web server should return an authorized exception
    - The web server should validate the POJO and return a Bad Request if it fails
  + Request:
    - Header: Sends the JWT on the Authorization Header
    - Body:

{

"title": "Groceries Shopping",

"content": "Groceries Shopping: tomatoes, meat, milk",

}

* + Response: HTTP 200

{

"noteId": "866ee0be-5d13-41a8-8d67-c7d91f669cd7",

"title": "Groceries Shopping",

"content": "Groceries Shopping: tomatos, meat, milk",

"lastModifiedDate": "02/02/2021T10:12:00",

"CreatedDate": "02/02/2021T10:12:00"

}

* Update Note
  + PATCH: {domain}/api/note/{ID}
    - If the user is not logged the web server should return an Unauthorized Exception;
    - The web server should validate the POJO and return a Bad Request if it fails;
    - If the note doesn’t exist, the web server should return an Object Not Found Exception;
    - If the note doesn’t belong to the logged user, the web server should return a Forbidden error
  + Request:
    - Header: Sends the JWT on the Authorization Header
    - Body:

{

"title": "Groceries Shopping",

"content": "Groceries Shopping: tomatoes, meat, milk",

}

* + Response: HTTP 200

{

"noteId": "866ee0be-5d13-41a8-8d67-c7d91f669cd7",

"title": "Groceries Shopping",

"content": "Groceries Shopping: tomatoes, meat, milk",

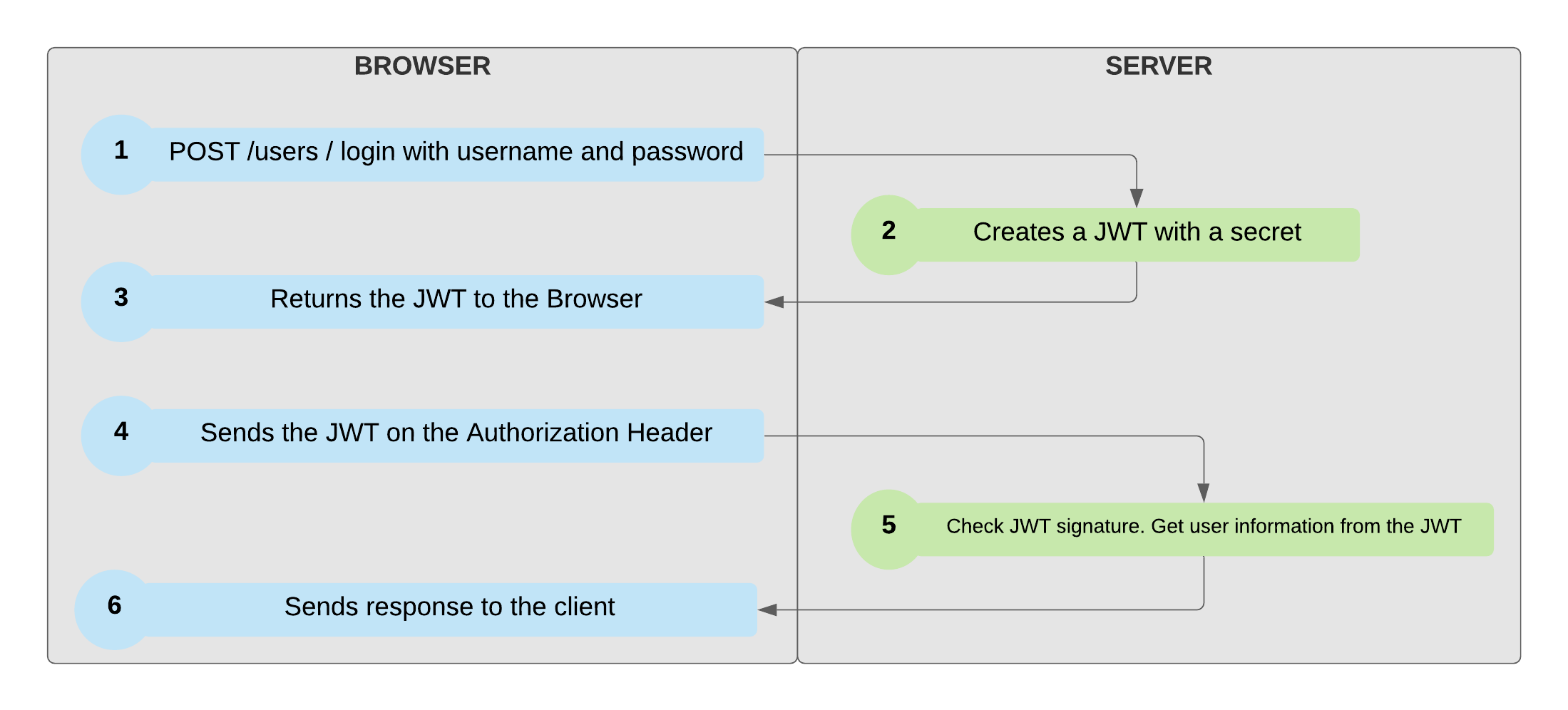
"lastModifiedDate": "02/02/2021T10:12:00",

"CreatedDate": "02/02/2021T10:12:00"

}

* Delete Note
  + DELETE: {domain}/api/note/{ID}
    - If the user is not logged the web server should return an Unauthorized Exception;
    - If the note doesn’t exist, the web server should return an Object Not Found Exception;
    - If the note doesn’t belong to the logged user, the web server should return a Forbidden error
  + Request:
    - Header: Sends the JWT on the Authorization Header
  + Response: HTTP 204

1. What user login and session strategies could be used?

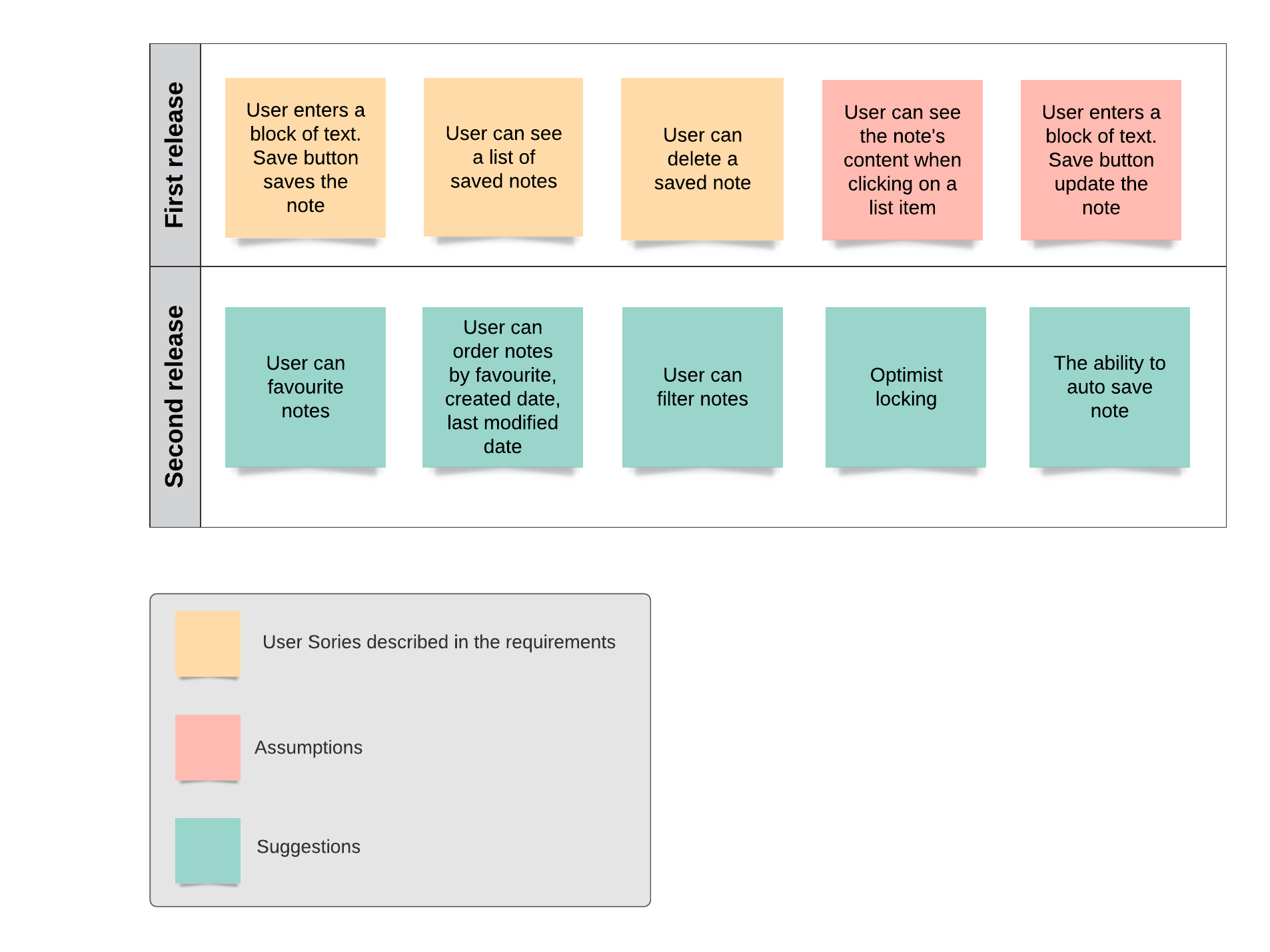


1. how would your UI solution facilitate extension and reuse?

Creating small components

* + 1. New Item
    2. List
    3. Item List
    4. Delete Button
    5. Save Button
    6. Save Modal
    7. Delete Modal

1. What was considered when deciding on a storage component?
   1. If the data to manage is very structured, then we can rely on the reliable SQL database. SQL databases are very good at managing data with a structure that rarely changes over time. We might need to add a table or column to our schema occasionally, but overall the data format stays the same. If this is your situation, pick this proven technology.
   2. If our data isn’t quite so clean and neat, we can opt for a flexible column family approach. It is similar to the column approach of a SQL database, but the columns have much more dynamic flexibility. Every row doesn’t have to have the same column structure as every other. The most common are: Cassandra, DynamoDB
   3. if our data **doesn’t conform to any structure at all,** we can use a document data store. Some common ones include: CouchDB, MongoDB
2. How might the capability for a user to share a note publicly affect the design?
   * We’ll could create a many to many relationship, with the noteId, ownerId, guestId, permission (write, read);
   * Add a new Boolean attribute on Note called shared;
   * Create new category list on the front end to show shared notes;
   * Create the option to share new items with registered users;
   * Create the ability to revoke shared notes;
   * Create new category list to show notes shared with me;
3. What additional capabilities would be useful, and how would these affect the design?



* User Can favourite note
  + New column to set favourite note
* User con order the note list
  + Using Sort component, we can order by any field
* User can filter notes
  + Using Specification component, we can filter by any field
* Optimist locking: The ability to check if the same document is opened in different browser or devices
  + Creating Node version Id and retrieving this data on API;
  + The server should check the last version;
* Auto save – The ability to auto save notes
  + Timeout should be created to call the save API