Planning

1. Requirements
    a. Functional
    b. Non-functional
2. Component Architecture (High level)
3. Data APIs and Protocol
4. Data Entities
5. Data Store
6. Optimization and Performance
7. Accessibility
8. Secuirity

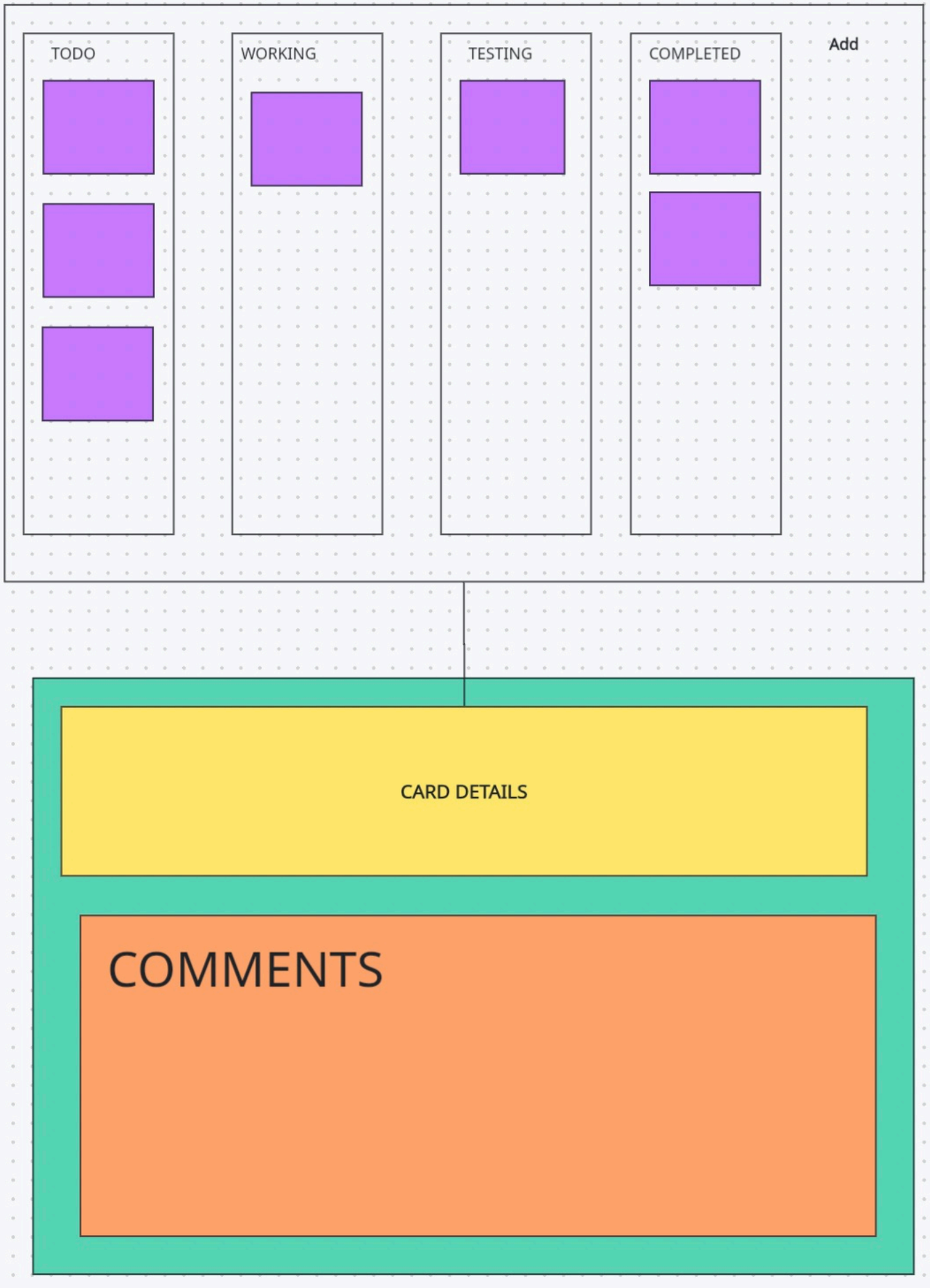
Functional requirement

1. User can able to create the board.
2. User can able to add the new Section in the board.
    a. TODO
    b. WORKING
    c. TESTING
    d. REVIEW
    e. COMPLETED
3. User can able to add / remove the cards.
4. User can able to add comment on the card
5. User can be able to update in real time.
6. It should support the offline mode.

Non-Functional requirement

1. It should support variety of devices
2. It should be consistent across devices
3. It should be responsive.
4. Adaptive loading
5. Localization / Internationalizatoin
6. Web vitals

Component Architecture



Protocol

1. Rest Approach
2. GraphQL

1. createBoardAPI(token, boardName);
2. deleteBoardAPI(token, boardID);
3. addColumnToBoard(token, boardID, columnName)
4. removeColumn(token, boardID, columnID)
5. createCardAPI(token, boardID, columnID)
6. updateCardAPI(token, metadata, cardID)
7. deleteCardAPI(token, cardID)
8. moveCardAPI(token, boardID, sourceColumnID, destinationColumnID, cardID)

roach

```
Type Board = {
  id: String
  userID: String
  title: String
  metadata: [Attachments]
  columns: [Column]
}
```

Local S

1. Persist across tabs.
2. After clossing the bro
   on we can able to retr
   data.
3. Data will not be erase
   manually delete.
4. Easy to mange.
5. Storing in key value p
6. We have only 5mb sto

ne);

dID,

cardID)

umnID,

```
Type Column = {
  id: String
  userID: String
  originID: String
  title: String
  cardList: [Card]
}
```

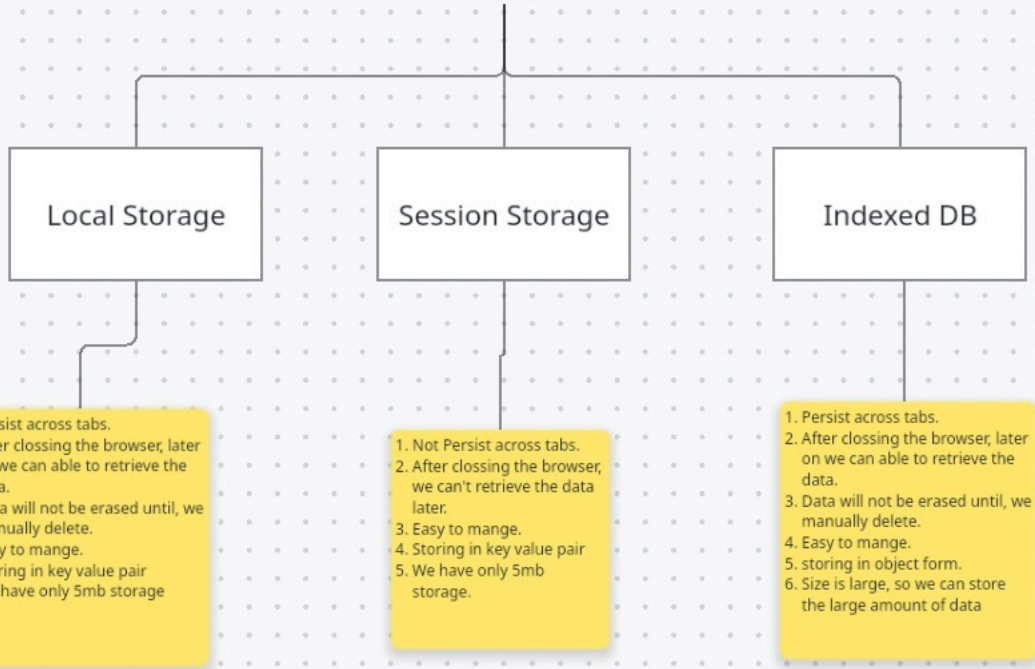
```
Type Card = {
  id: String
  userID: String
  originID: String
  title: String
  currentStatus: CardStatus
  description: String
  attachments: Attachments
  Comments: [Comments]
}
```

```
enum CardStatus = {
  TODO,
  WORKING,
  TESTING,
  REVIEW,
  COMPLETED
  PROD
}
```

```
Type Attachments = {
  id: String
  imageURL: String
  originID: String
}
```

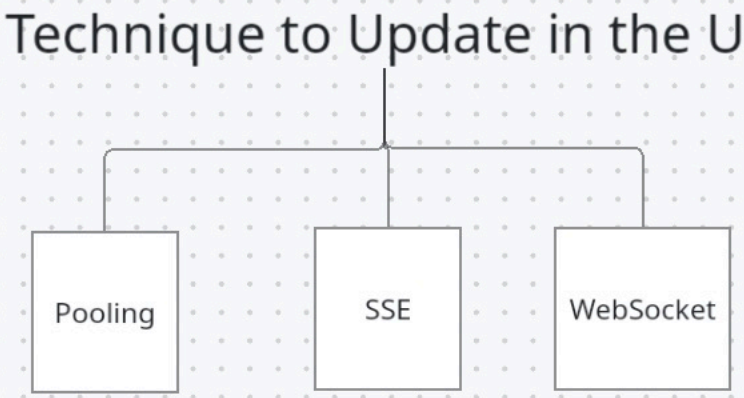
```
Type Comments = {
  id: String
  origin_id: String
  metadata: String
}
```

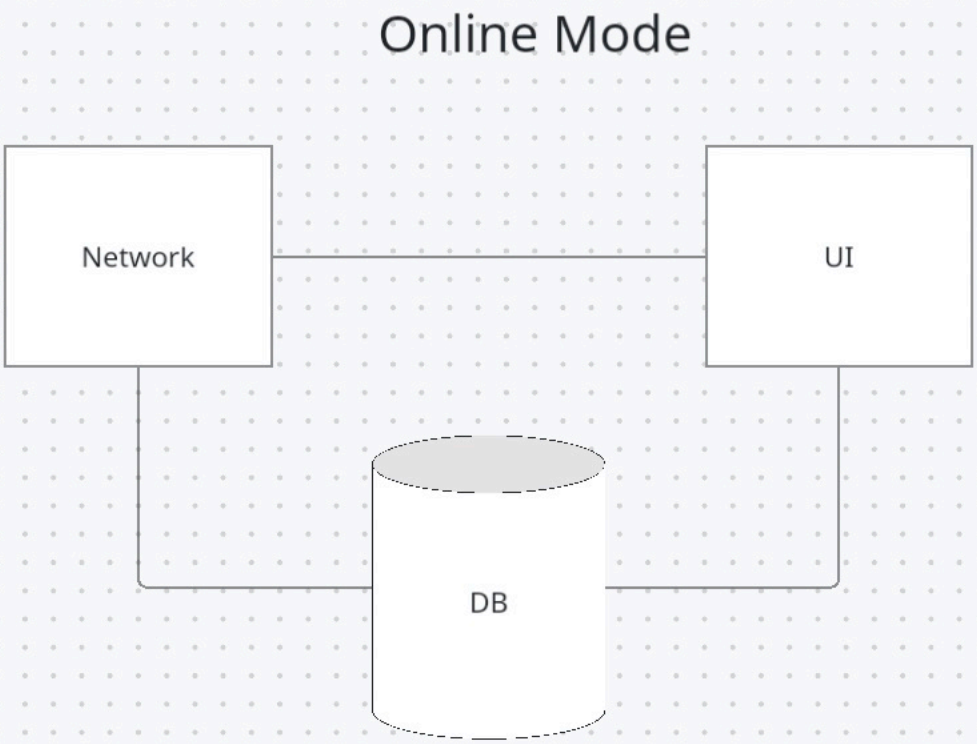
Data Store

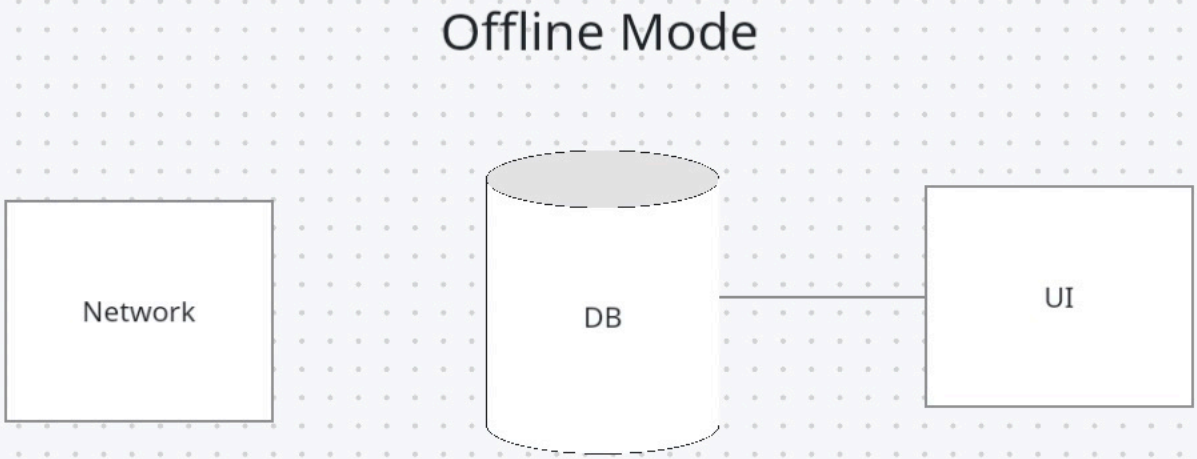


Normalization Technique

```
{
  columndId: {
    "123": column_details,
    "2342": COlumen_Detail2,
    "5343: column_details,
  },
  cardsIDs: {
      "123": column_details,
    "2342": COlumen_Detail2,
    "5343: column_details,
  },
}
```


Technique to Update in the UI
Pooling
SSE
WebSocket


Online Mode
Network
UI
DB


Offline Mode
Network
DB
UI

Offline Mode Technique

1. If we are depend on some server logic to handle the something, that logic should be preset in local also
2. Delta Structure
3. Uploading Deltas
4. Sync Failure handling
5. Idempodence
6. Conflicts Resolution
7. Attachments Sync

Offline Mode

The entire app should be able to run off the local database alone

If any logic which we were used to depend on the server , now we have to implement in the client side.

Now, Suppose we are somhow manage to write all our logic and able to save the data in local in ofline conditon, now we need to pass all these to server

Data Syncing, when user came to online

Delta Structure

It is basically a terminology,
what ever the changes user will make, we will store all of them in a queue like structure

And we will consider, all changes as a seperate entry

Uploading Deltas

Now, the problem is how are going to upload our delats, to the server in which format.

So, Basically we will upload the data in the same order, In which we recieved the delta.

We always execute deltas in the order received. That way you know you couldn't end up with any data hierarchy issues

Sync failure handling

Now, there could be problem occur, suppose, we have pushed one delta successfully and 2nd delta update is failed due to some reason, then how we are going to handle this

To handle this, we can follow two strategy
1. We will use something n-retry technique.
2. Simply drop that delta.

Idempotence

Suppose we are trying to create, a card we make the request and In the server this, is successfully gets created, but in the client side we didn't receive the response, may be due to some network issue or something else.

To handle this, what we can do Is, for ever operation, we will send a unique key or UUID to the server, and we will wait.

If we didn't receive any response, then again we will send the same key, such that server will understand Ok, this is the same actiion, which I just perform,

Then the server will not update the same thing again.

Conflict resolution

It means, when the multiple user trying to update the same card

Last Writer wins.

The above may fails, for some case, but It is easy to implement otherwise we have to thing some diff logic.

Our we can save the history, If user want to retrieve some data, he can easily able to do.

Attachment Sync

We can create 2 queue, because in case of upload, it will block our queue, then our basic request will also take time.

```
{
  highPriorityQueue: [...], // Form changes, status updates, etc.
  mediaUploadQueue: [...] // Images, files
}
```

Two handle this, attachment sycn such that our mail thread will not get block, we can use
Web Workers and we can split the large files in smaller chunk