Project Design

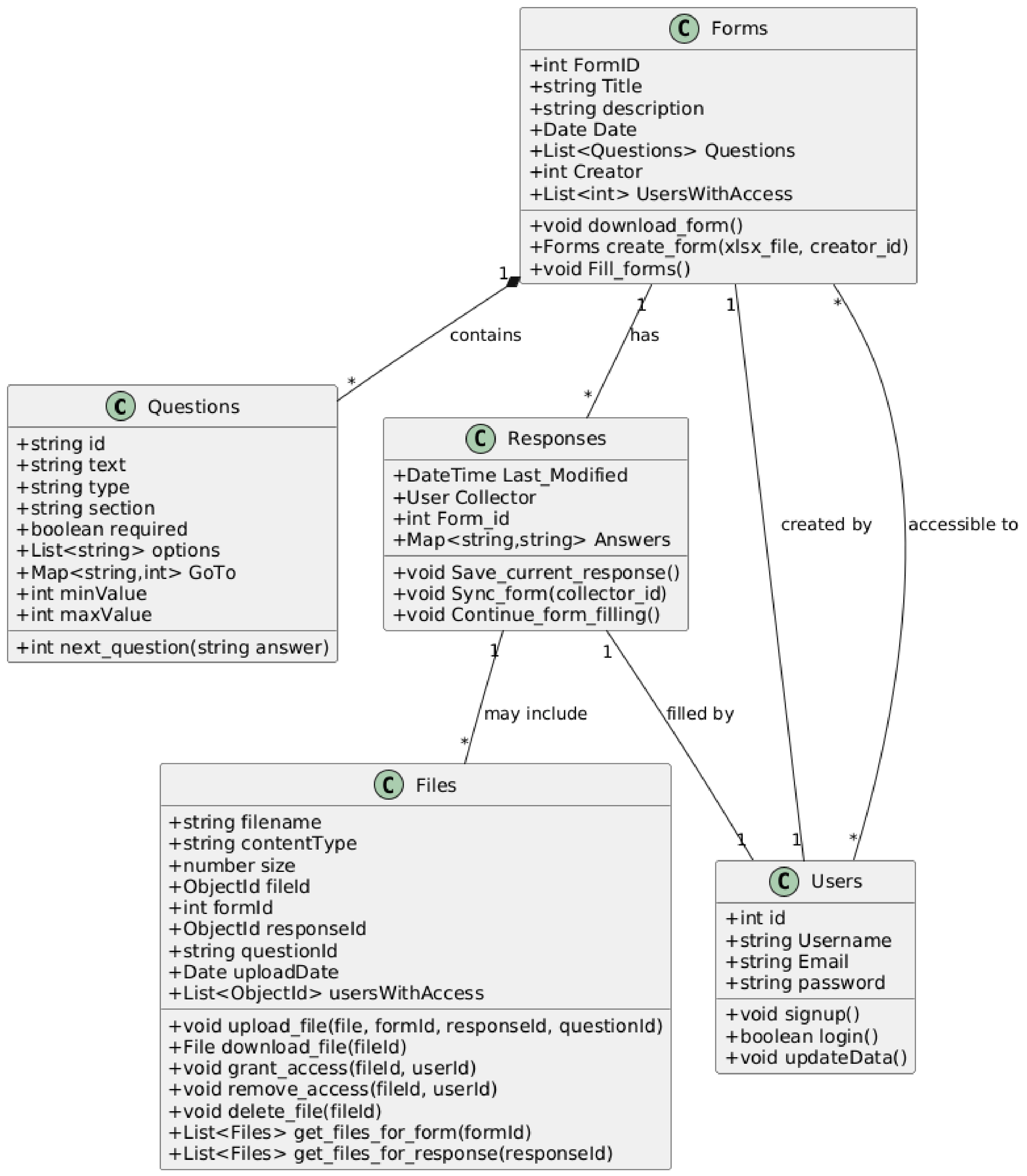
Team	**Team 38: Aishita, Harshikaa, Kushal, Sriyansh, Vishesh**

Design Model

Questions	Class State 1. id of the question (unique within a form) 2. text - string: content of the question 3. type - string: the type of question it is (mcq, text, number etc.) 4. section: text denoting the section in which this question belongs. 5. required: Boolean field to tell if the question is ‘required’ or not. 6. options - list[string]: list of options in case of a multiple choice question 7. GoTo - Map(string, int): the questions to go to for each option chosen 8. minValue: min value for questions of type range 9. maxValue: max vallue for questions of type range Class Behaviour 1. next_question(string): given the answer to this question, returns the id of the next question to be shown
Forms	Class State 1. FormID: id of the form 2. Title: title of the form 3. description: given for the form 4. Date: date of creation 5. Questions: array of type Question 6. Creator: The id of creator of the form 7. UsersWithAccess: A list of ids of users who have access to this form. Class Behaviour 1. download_form - download the data of a from to local storage 2. create_form(xlsx_file, creator_id) - create a new form from the xlsx file provided and upload it to the database 3. Fill_forms - Create a new response instance for the form and update

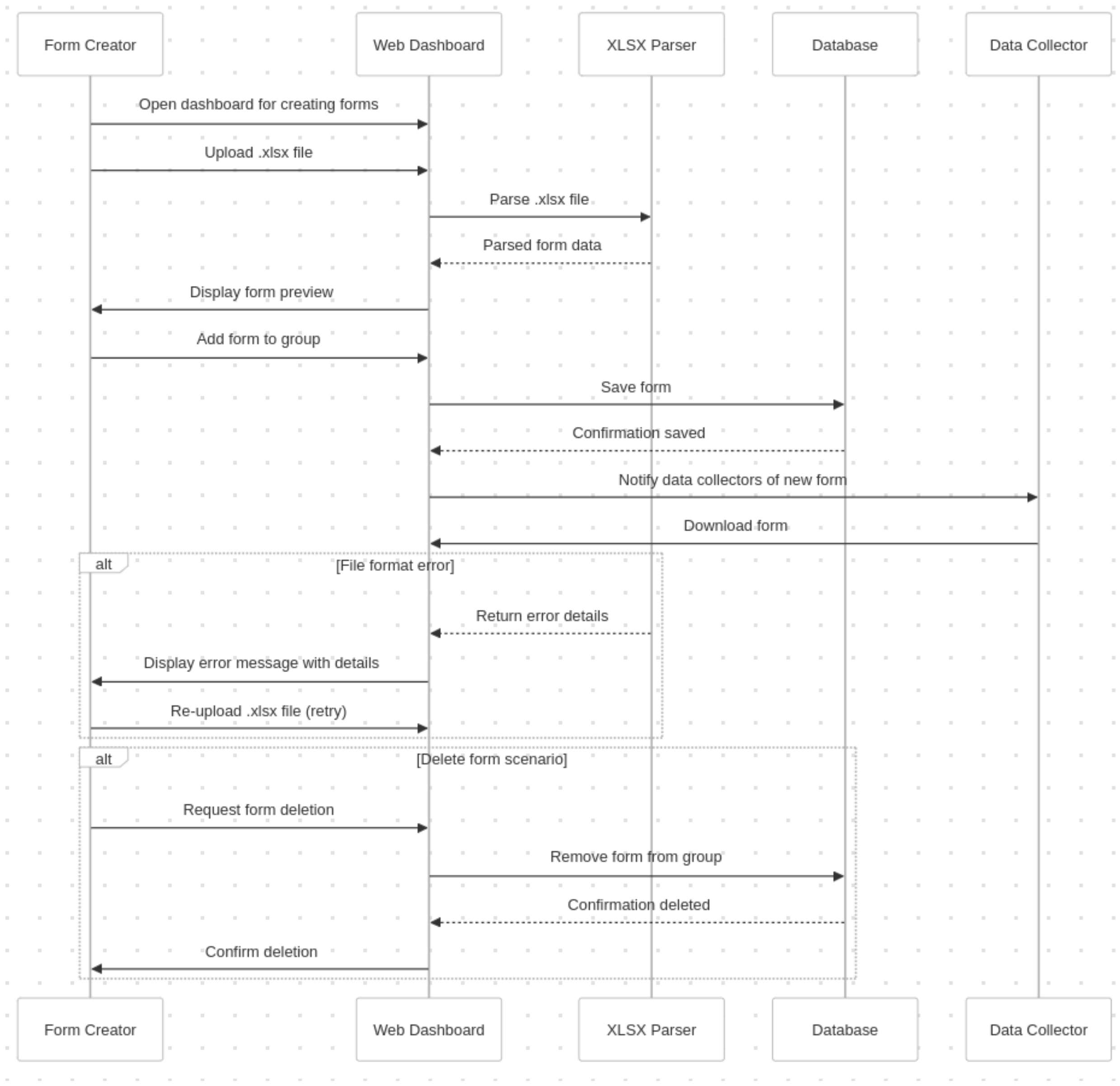
	the instance with the answers entered by the user.
Responses	Class state 1. Last_Modified - DateTime : The timestamp when the response was saved 2. Collector - User : The details of the person filling the form 3. Form_id - Int: References the form from which the questions are fetched. 4. Answers - {QuestionId : Response} : A dictionary of the question to corresponding responses Class behavior 1. Save_current_response() - Saves the current state of the response in local storage of the device for persistence 2. Sync_form(collector_id) - The instance saved in local memory is synced with the main server when there is internet connection 3. Continue_form_filling(): A partial response (stored in local memory) is now re-opened to finish filling the form or edit the previous responses.
Files	Class State 1. filename - string: original name of the uploaded file 2. contentType - string: MIME type of the file 3. size - number: size of the file in bytes 4. fileId - ObjectId: reference to the actual file in GridFS 5. formId - number: references the form this file is associated with 6. responseId - ObjectId: references the response this file is associated with 7. questionId - string: identifies which question this file answers 8. uploadDate - Date: when the file was uploaded (defaults to now) 9. usersWithAccess - array[ObjectId]: list of users who can access this file Class Behavior 1. upload_file(file, formId, responseId, questionId) - uploads a new file and associates it with a form/response 2. download_file(fileId) - retrieves a file for viewing or downloading 3. grant_access(fileId, userId) - gives a specific user permission to access a file 4. remove_access(fileId, userId) - removes a user's permission to access a file 5. delete_file(fileId) - completely removes a file from the system

	6. get_files_for_form(formId) - retrieves all files associated with a specific form 7. get_files_for_response(responseId) - retrieves all files associated with a specific response
Users	Class State • id: Unique identifier for the user. • Username: The full name of the user. • Email: The email address used for login. • password: The password (should be stored securely, e.g., as a hash). Class Behavior • signup: Handles the registration process for a new user. • login: Authenticates the user based on the provided email and password. • updateData: Allows updating user information such as name, email, or password.

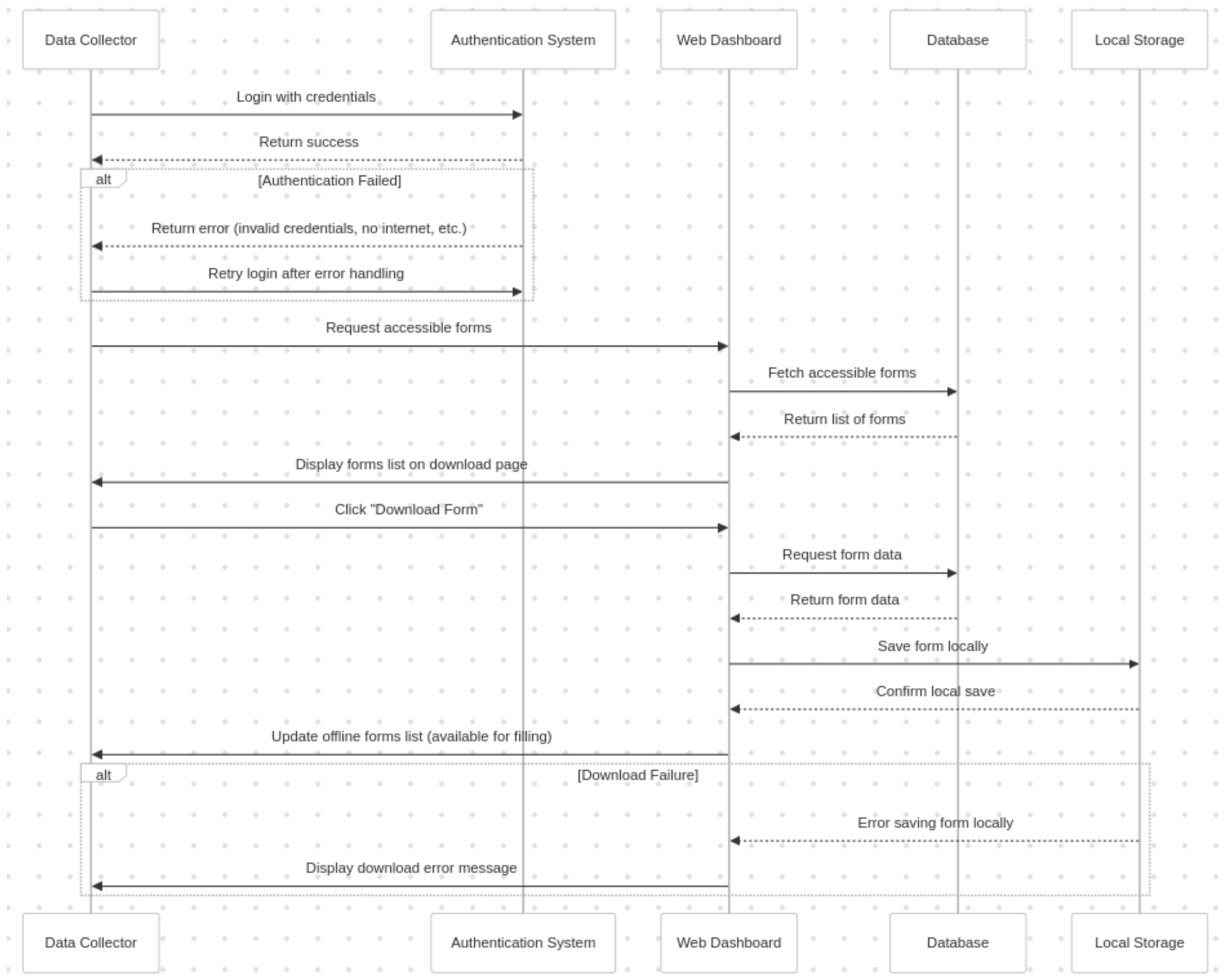


Sequence Diagram(s)

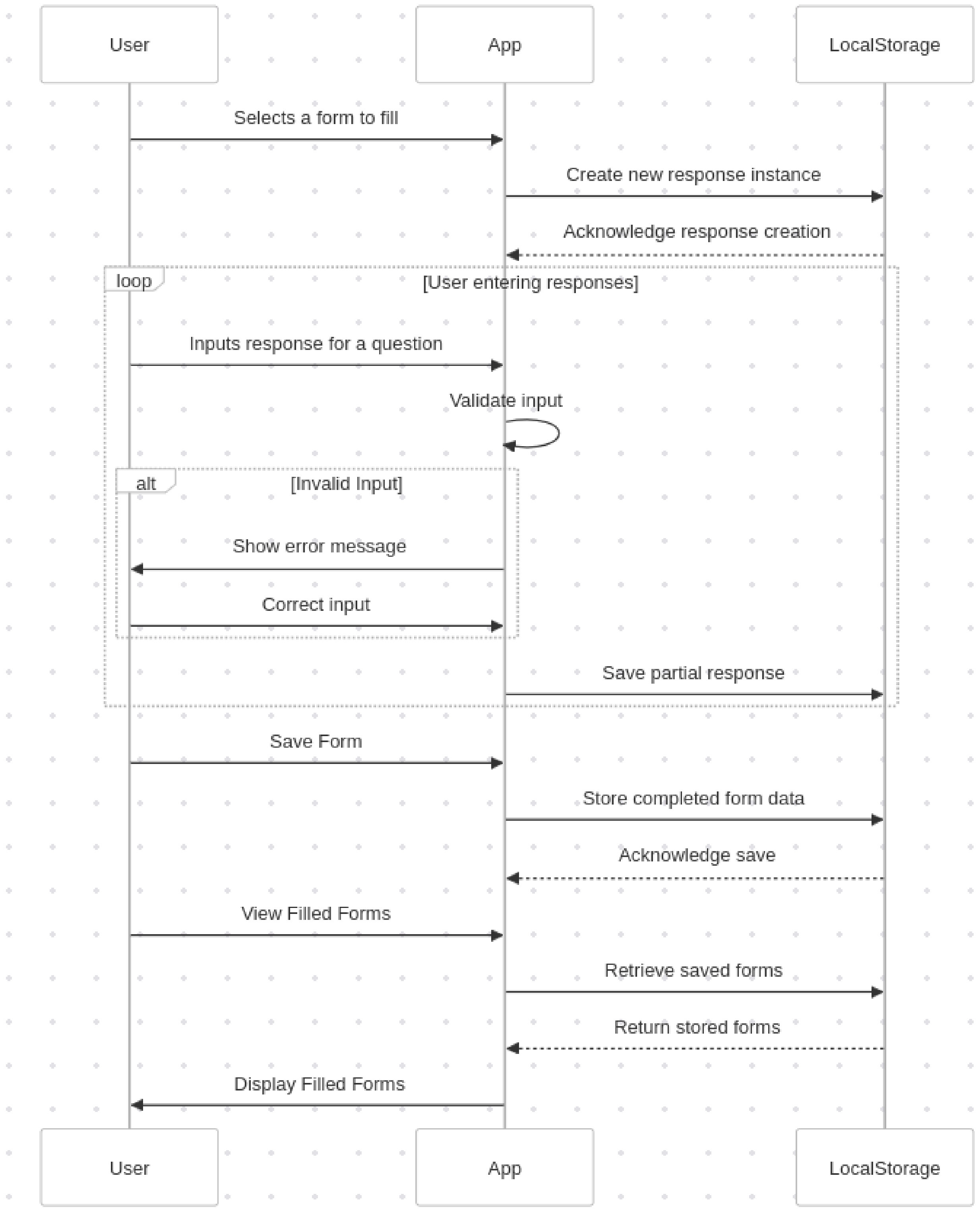
1. Make Forms:



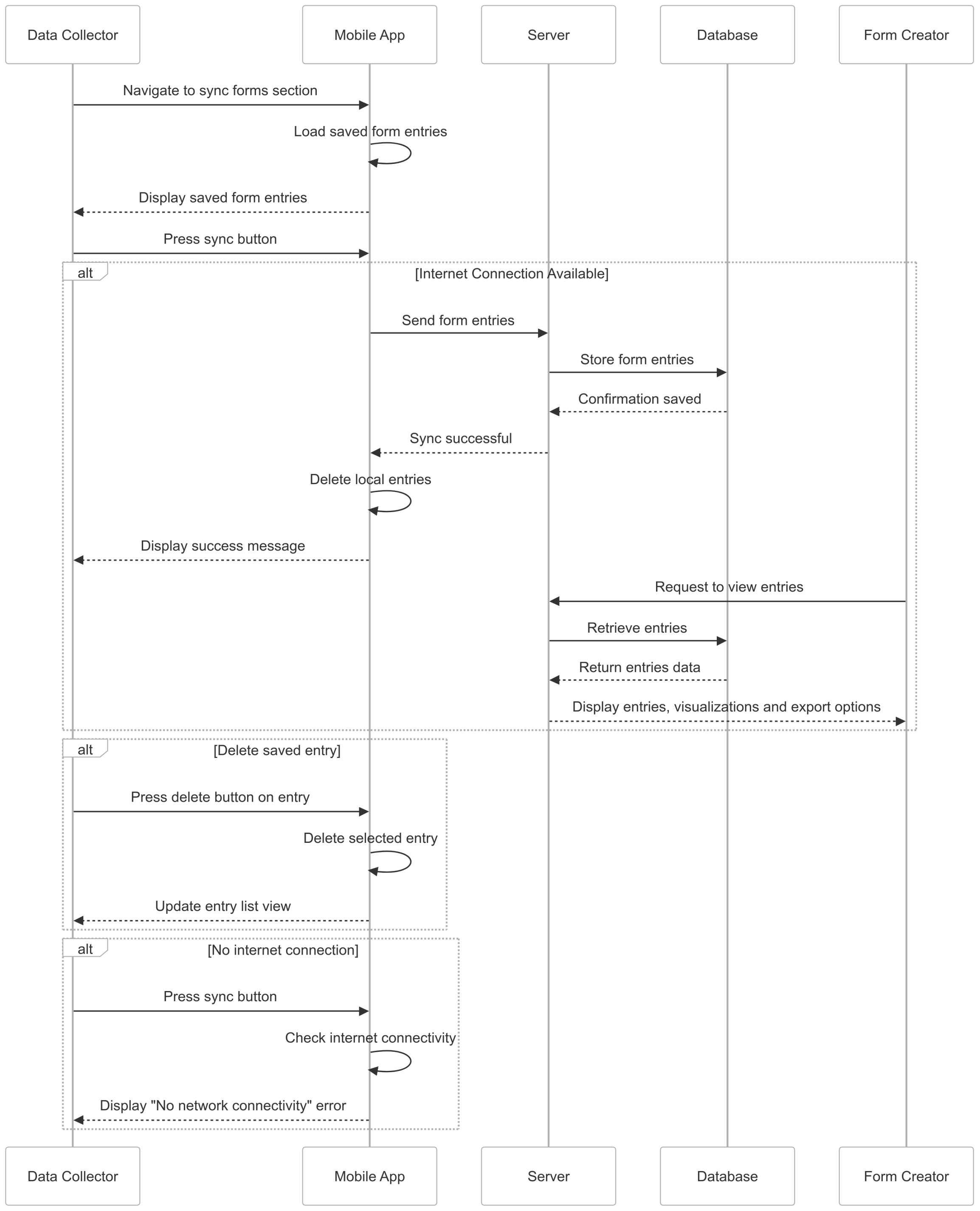
2. Download Forms


Data Collector
Authentication System
Web Dashboard
Database
Local Storage
Login with credentials
Return success
alt
[Authentication Failed]
Return error (invalid credentials, no internet, etc.)
Retry login after error handling
Request accessible forms
Fetch accessible forms
Return list of forms
Display forms list on download page
Click "Download Form"
Request form data
Return form data
Save form locally
Confirm local save
Update offline forms list (available for filling)
alt
[Download Failure]
Error saving form locally
Display download error message
Data Collector
Authentication System
Web Dashboard
Database
Local Storage

3. Fill Entry



4. Sync Entries



Design Rationale

- To ensure that only relevant questions are displayed based on previous answers, we explored two possible approaches for defining skip logic when designing forms in Excel.

1. **Prerequisites Approach**
   - Each question includes the ID of its prerequisite question, specifying a condition that must be met for it to be displayed.
   - If a question has no prerequisites, a special indicator is used.

   Advantages:

   1. Simplifies implementation—developers can pre-process dependencies using a DFS or BFS approach, iterating through questions and displaying them only when all prerequisites are satisfied.
   2. Allows the form creator to define prerequisites as they draft each question.

   Disadvantages:

   1. This is a bottom-up approach, which may feel unintuitive for form creators since they must think about dependencies for each question rather than designing the logical flow naturally.

---

2. **Go-To Approach (Adopted)**
   - Each question includes a "Go-To" statement that determines the next question to be displayed based on the user's selection.

   Advantages:

   1. Intuitive—closely aligns with how humans naturally design skip logic.
   2. Offers flexibility—form creators can define question flow logically, similar to a decision tree.

   Disadvantages:

   1. Slightly more complex to implement, as it requires managing multiple branching paths.
   2. Requires knowledge of subsequent question IDs—but this can be mitigated by assigning "Go-To" rules after composing all questions.

- For a particular form we had two options to share it with data collectors who would be getting responses from the field. One was to create a unique form ID and password for each form. This can be shared by the form creator and allow users to access any form they have the credentials for. The other for creating user profiles and giving access to a form to a specific set of users defined.

1. **User Profile Approach**
    - Create user profiles for form creators and data collectors with login credentials
    - Form creators can grant specific data collectors access to their forms
    - Data collectors can see and download only the forms they have access to

    Advantages:

    1. Better security - if credentials are compromised, access can be quickly revoked
    2. Simplified user experience - data collectors can see all forms available to them in one place
    3. Centralized access management for form creators

    Disadvantages:

    1. Requires user registration and account management system
    2. More complex backend implementation
2. **Form-specific Access Approach**:
    - Each form has a unique ID and password shared by the creator
    - Anyone with the ID and password can access and submit data to the form

    Advantages:

    1. Simpler implementation - no need for user accounts
    2. Quick sharing - just send credentials to anyone who needs access

    Disadvantages:

    1. Security risks if credentials are leaked
    2. No centralized management of form access

3. More effort for users to keep track of different form credentials
4. Difficult to revoke access selectively

We have adopted the User Profile approach as it provides better security and user experience despite requiring more complex implementation on the backend.