Challenge

Develop Interactive Reactive Forms Inside SPA

Challenge

- Challenge: Develop a user registration form for the `Keep-Note` application.

Points to Remember

- The Angular Reactive form should be created to add a new user. The form input elements must be created using Angular material components.
- Use `formGroup` and `formControlName` directives to bind the form model with the template.
- Use built-in and custom validator functions to validate form input values. Use `<mat-error>` to display validation error messages.
- Custom styles should be added while designing the form.
- The newly added user should be saved in the `notes.json` file available in the **`keep-note-data folder`** using the user service.

Instructions for Challenge

- Click here for the boilerplate.
- Please read the README.md file in the boilerplate for further instructions about the challenge.
- Fork the boilerplate into your own workspace.
- Clone the boilerplate into your local system.
- Open the command terminal and set the path to the folder containing the cloned boilerplate code.
- Run the command npm install to install the dependencies.
- Use the solution code of the `Keep-Note` application developed for the challenge of the sprint: `Developing Interactive Template-Driven Forms Inside SPA`

Notes:

1. The solution to this challenge will undergo an automated evaluation on `CodeReview` platform. (Local testing is recommended prior to testing on the `CodeReview` platform).
2. The test cases are available in the boilerplate.

Context

As you are aware, `Keep-Note` is a web application that allows users to maintain notes. It is developed as a single-page application using multiple components.

Note: The stages through which the development process will be carried out are shown below:

Stage 1: Create basic `Keep-Note` application to add and view notes.

Stage 2: Implement unit testing for the `Keep-Note` application.

Stage 3: Create Keep-Note application with multiple interacting components to add, view and search notes.

Stage 4: Implement persistence in the Keep-Note application.

Stage 5: Style the Keep-Note application using Material design.

Stage 6: Create a simple form with validation in the `Keep-Note` application.

Stage 7: Create a complex form with validation in the `Keep-Note` application.

Stage 8: Enable navigation in the `Keep-Note` application.

Stage 9: Secure routes in the `Keep-Note` application

Context (Cont'd.)

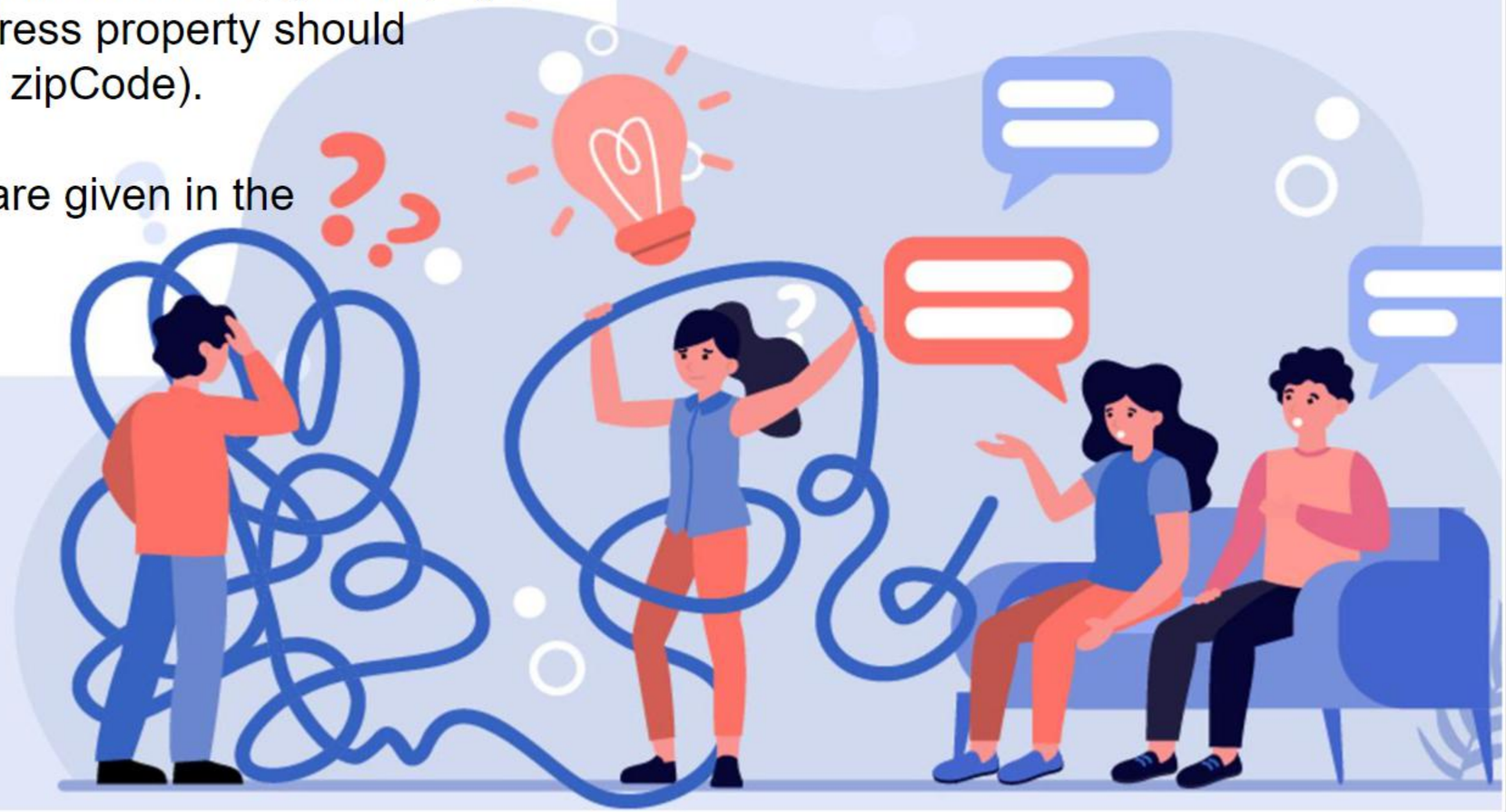
- In this sprint, we are at Stage 7.
- In the previous stage, a template-driven form was created to add a new `note`.
- In this stage, a reactive form with validations using Angular Material components should be created to register a new user.

Develop a User Registration Form for the Keep Note Application

Develop a user registration form to add a new user using Angular reactive forms. The data model for the user should include the following properties:
firstName, lastName, password, confirmPassword, gender, age, email, phone, and address. (The address property should contain details: street, city, state, and zipCode).

Note: The tasks to develop the form are given in the upcoming slide.

CHALLENGE



Tasks

- Following tasks need to be completed to develop the solution for the `Keep-Note` application:
  - Task 1: Include required modules in the App module
  - Task 2: Create the `RegisterFormComponent`
  - Task 3: Define the form group
  - Task 4: Add validators to the form controls
  - Task 5: Create the HTML form in the template
  - Task 6: Handle Form Validation
  - Task 7: Display a notification message upon successful form submission

Notes:

1. Details related to a few tasks are given in the upcoming slides.
2. The `RegisterFormComponent` will be used in the next sprint: `Implement Navigation Using Angular Routing` to integrate with the existing `Keep-Note` application.

Task 2: Create the RegisterFormComponent

- Create a new component with the name `RegisterForm`.
- This component is used to develop the registration form to register a new user.
- The Keep-Note application should now get launched with the `RegisterFormComponent`.

Note: The `RegisterFormComponent` name mentioned above is used in testing, so you must use the same names while coding.

Task 3: Define the Form Group

- Create a form group with the name `registerForm` that has the following:
  - Form controls: `firstName`, `lastName`, `password`, `confirmPassword`, `gender`, `age`, `email`, and `phone`.
  - Sub form group called `address` that has the following form controls:
    - `street`, `city`, `state`, and `zipCode`

Note: The form group name and form control names mentioned above are used in testing, so you must use the same names while coding.

Task 4: Add Validators to the Form Controls

- The following are the form controls with their validation criteria.

Form Control	Validation
First Name	Should not be left blank and has a minimum length of 2 characters
Last Name	No Validation
Password	Should not be left blank and should have a minimum 8 characters with at least one symbol, one upper-case letter, one lower-case letter, and one number
Confirm Password	Should not be left blank and should be the same as Password
Age	Should be initialized to 0 and must be greater than or equal to 18
Email	Should not be left blank and should accept a valid email id
Phone	Should accept only a 10-digit number starting with 7, 8, or 9
Gender	No Validation (Select from the given set of values: Male, Female and Others)
Address: Street, City, and State	No validation
Address: ZipCode	Should be a 5-digit or 6-digit number

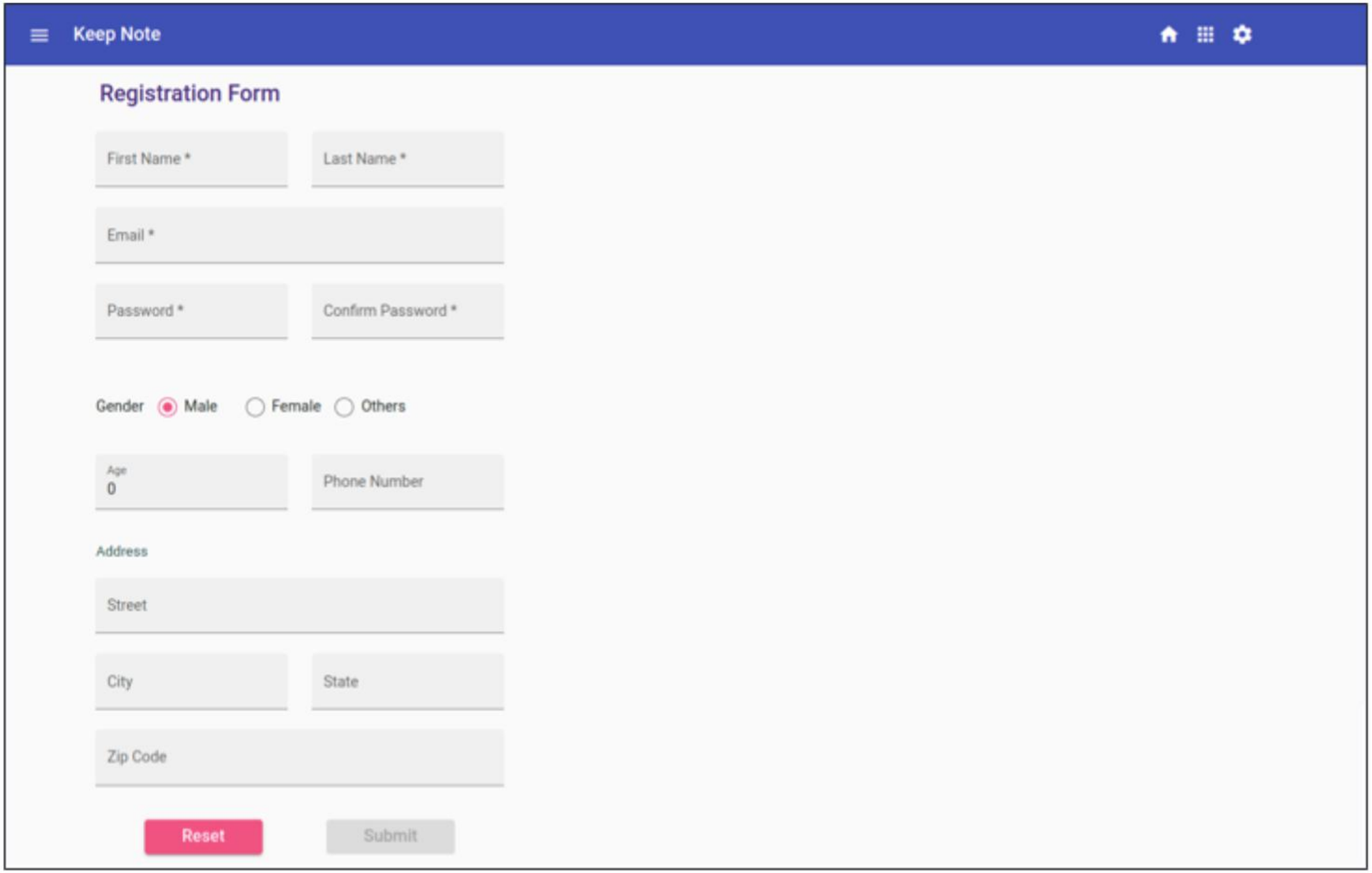
Task 4: Add Validators to the Form Controls (Contd.)

- Custom validator functions should be created inside the **`register.component.ts`** file for the following criteria.
  - Age value should be greater than or equal to 18. (Single-field validator – should be added for age form control)
    - Should return an object **`{ invalidAge: true}`** when age value entered is less than 18.
  - `password` and `confirmPassword` values should be equal.(Multi-field validator – should be added at the form level)
    - Should return an object **`{passwordMismatch: true}`** when password and confirm password values entered are not equal.
  - Proper error message should be displayed by checking these values using `<mat-error>` and `*ngIf` directive inside the template code.

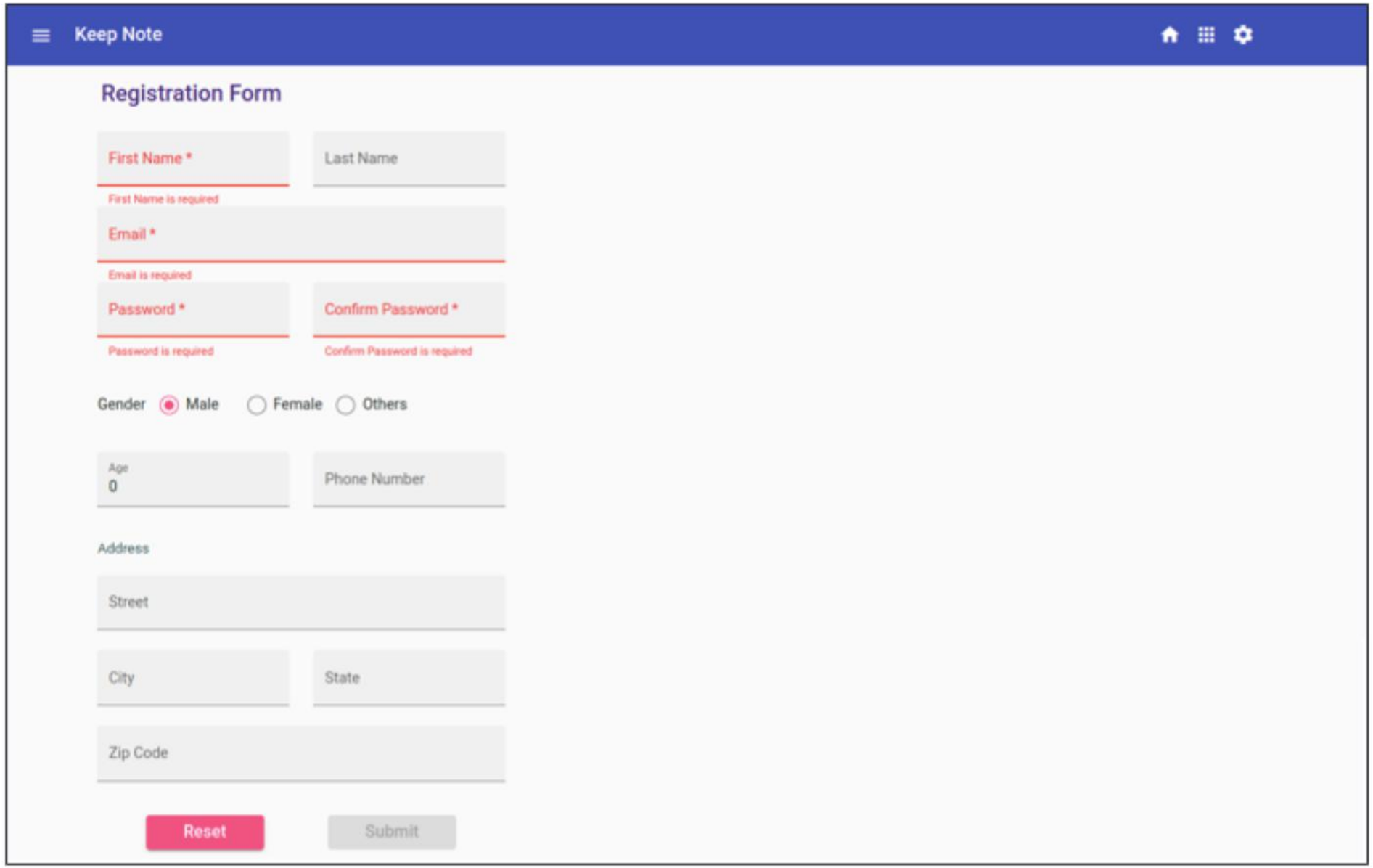
Notes:

1. Object key names – `invalidAge` and `passwordMismatch` mentioned above are used in testing, so you must use the same names inside the code.
2. Avoid using HTML5 attributes for validating the rest of the form controls.

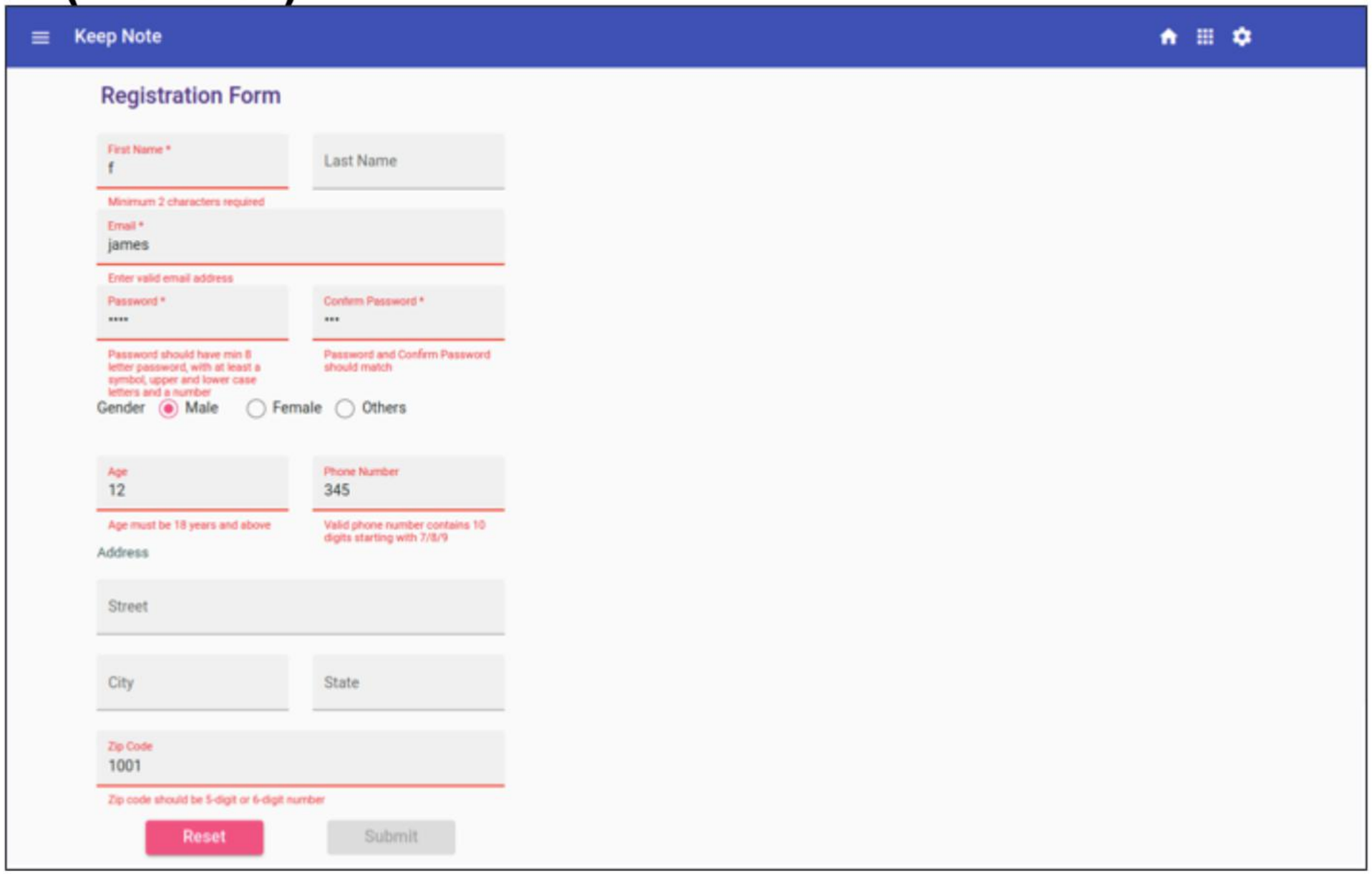
Expected Output: After Task 5 - Registration Form



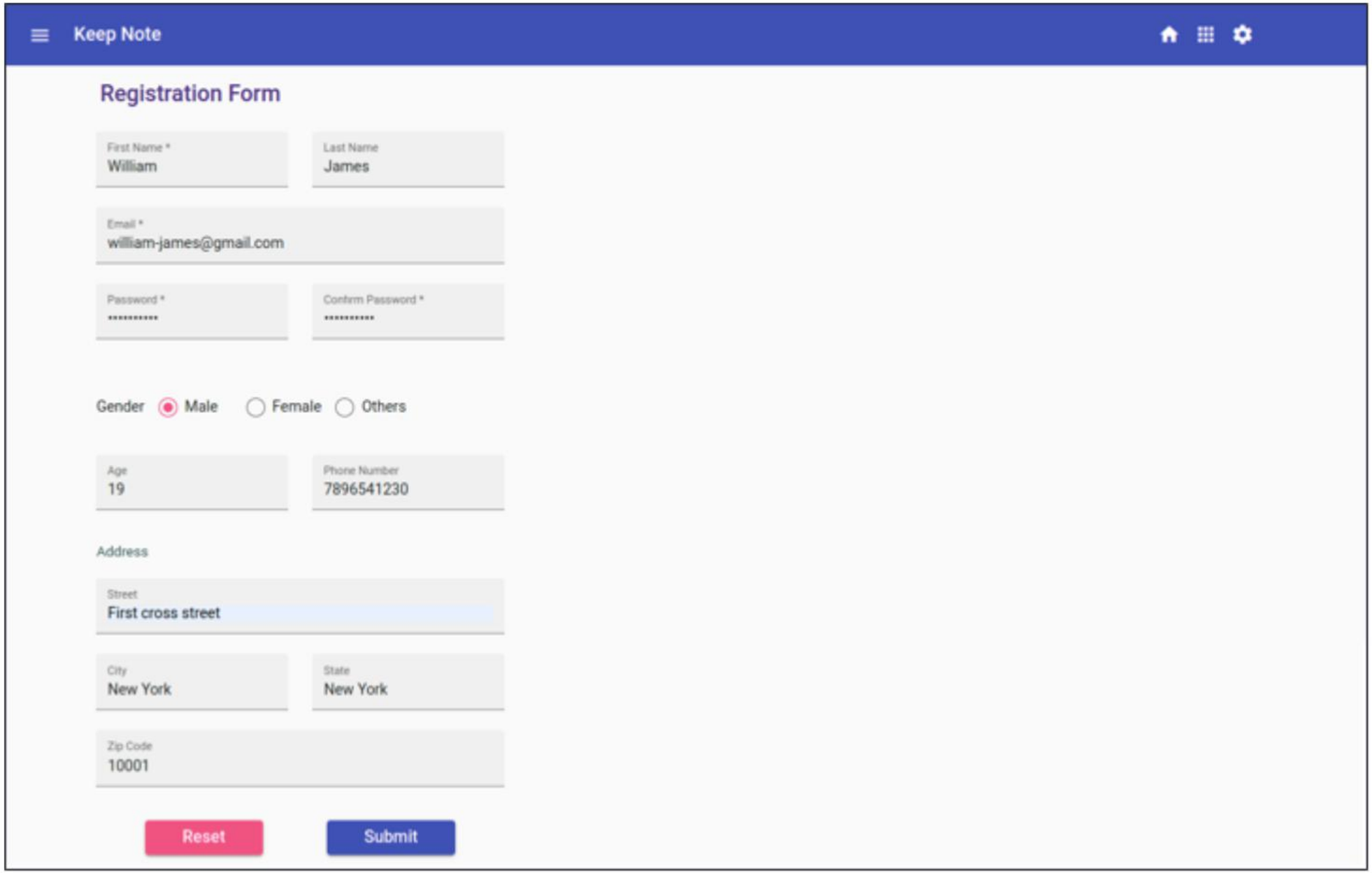
Expected Output: After Task 6 - Form With Validation Errors



Expected Output: After Task 6- Form With Validation Errors (Cont'd.)



Expected Output: After Task 6 - Form With Valid Values



Expected Output: After Task 7 - Successful Form Submission