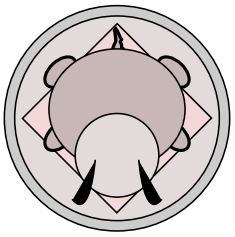

Project Specifications of Blooper App

(Version: 1.2)

Whoopie Bard <whoopie@clueless.dev>
Changu Bhai <changu.bhai@clueless.dev>

15 November, 2024

Contents

1 Document Info

1.1 Authors

Following authors were involved in preparation of this document.

Authors	Email	Nickname
Changu Bhai	changu.bhai@clueless.dev	changu
Whoopie Bard	whoopie@clueless.dev	whoopie

1.2 Versions

Versions	Author	Changes	Date
v 1.0	whoopie	Added introductory text.	2 October, 2024
v 1.1	changu	Completed functional specifications.	28 October, 2024
v 1.2	whoopie	Updated partial program specifications.	15 November, 2024

2 Introduction

Clueless Developers' Consortium has been allocated the project to develop Bloopers App. This document defines the specifications of that project. It is divided primarily into two parts - Functional Specifications and Program Specifications. For any queries regarding this project, please contact whoopie@clueless.dev.

It is sincerely hoped that this project specifications document would be found extremely useful by the client to manage and maintain the Blooper App properly for a very long period of time.

3 Functional Specifications

This section provides the detailed functional specifications of Blooper Application.

3.1 User Registration

This module defines the user-registration aspect of the Blooper. Anonymous users are allowed to self-register and proceed to use the application.

3.1.1 Requirement Specifications

Following requirements specifications are applicable to the user-registration module of the application.

- New user should be able to self-register to the Blooper application. Using a specified URL, anonymous users would be able to fetch a web-form to fill-up.
- User should be able to register from certain geographical regions only.

> Q: How to prevent people from using VPN to beat geographical restrictions?

Q: Is there any third party service, which provides such facilities?

- Users with an email address and mobile number should be able to create an account in Blooper.
- No two user-accounts should be possible to be created using same email address.
- No two user-accounts should be possible to be created using same mobile number.

3.1.2 Wireframes

Following wireframes are proposed for the user-registration screen.

3.1.2.1 User-registration Form

This screen is displayed to a new user whose user-account is yet to be created.
Initially, all fields turn out empty. The user needs to fill the required fields in this form.

Figure 1: User-registration Form

↰ paragraph 4.1.1.1

3.1.2.1.1 Form-Fields

Following form-fields are expected to be presented to the user for self-registration.

Form-Fields	Allowed Values	Max Size	Required	Unique
Address	A-Z, a-z, 0-9, - +	char(512)	No	
Email Address	.a-z0-9-_+@	char(128)	Yes	Yes
Full Name	A to Z, and a to z	char(128)	No	
Mobile Number	+-0123456789	char(12)	Yes	Yes

- **Email Address**:
  - It would be unique across the user-database.
  - That means, no two users can have same email address.
- **Mobile Number**:
  - Mobile number should be unique across the application-database.
  - No two users using this application can have same mobile number.

3.1.2.1.2 Buttons

Following buttons are expected to be made available on the screen.

- **Register**: Registers the new user.
  1. **Normal Flow**:
     - System validates supplied data before creating user-account.
     - If everything is in order, then system create new user-account.
  2. **Exception Flow**:
     If one or more validations fail, then system displays validation error with following details.
     - If email address is associated with any existing user-account, then system notifies that no new account can be created with same email address.
     - If mobile number is associated with any existing user-account, then system notifies the user that no new account can be created with the same mobile number.
- **Reset**: Reset all fields of the registration form.
  - **Normal Flow**:
    * Clean-up contents of all fields.

↰ subparagraph 4.1.1.1.1

↰ subparagraph 4.1.1.1.1

↰ subparagraph 4.1.1.1.1

3.1.2.1.3 Links

Following hyperlinks are expected to be present on the screen.

- **Login Here**: If account is present, click to sign-in.

3.1.2.2 Message Box for Existing Record

This screen is displayed, when one or more fields which are unique across the application is supplied as part of user-registration, but the same had been supplied earlier for another user-account too.
This could be a pop-up window.

↰ paragraph 4.1.1.2

3.1.2.2.1 Buttons

- **Try Again**: Re-submit the form by going to the registration form and by modifying its form-fields.

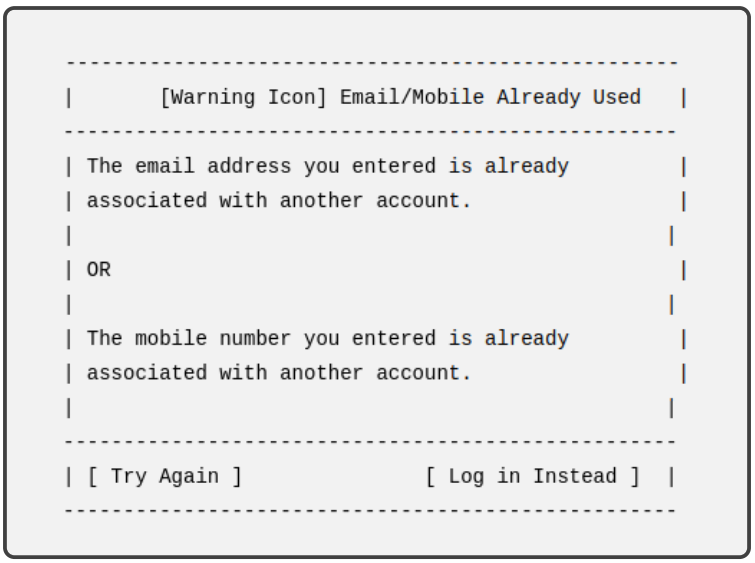

Figure 2: Message Box for Existing Record

Q: Is re-executing the GET request on the form required here?
Q: Is it not sufficient enough to handle it locally, by rendering the cached form itself?

- **Login Instead**: If an account exists already, proceed to the login form.

3.1.3 Level 3.1.3

Q: How can Level 3.1.3 be used as a section heading?

3.1.3.1 Sample Table

A Sample table is given below.

Sample Table	Header 1	Header 2	Header 3	Header 4
Row 1	1.1	1.2	1.3	1.4
Row 2	2.1	2.2	2.3	2.4
Row 3	3.1	3.2	3.3	3.4
Row 4	4.1	4.2	4.3	4.4
Row 5	5.1	5.2	5.3	5.4

- **Row 1**:
  - First note for Row 1.
  - Second note for Row1.
  - Third note for Row1.

- **Row 3**:
  - First note for Row 3.
  - Second note for Row 3.
  - Third note for Row 3.
  - Fourth note for Row 3.

4 Program Specifications

4.1 User-registration Module

This section elaborates on the program specifications of implementation of user-registration module of Blooper.
The functional requirements of this module is described in subsection 3.1.

4.1.1 Screens

Following screens are developed for user-registration purpose.

4.1.1.1 User-registration Screen

Please refer Figure 1 for wireframe of user-registration screen.

4.1.1.1.1 Buttons

- **Register**:

  The item 1 provides the details of the normal flow of actions when this button is clicked.

  The item 2 provides the details of the exception flow of actions when this button gets clicked.

  - **API Endpoint**: /api/v1/users/register
  - **Method**: POST
    HTTP POST method is used on this API endpoint for the intended result.
    POST method requires input data to be sent in HTTP request-body itself.
  - **Input**:
    Input formats are described below.
    * This is the JSON format of input.

      ```
      {
        "fullname": "Jagmeet Chautala",
        "emailaddress": "jagmeetc@example.com",
        "mobilenumber": "1234567890",
        "address": "72, B street, Park Avenue, Sonpat,
        Hryana, INDIA"
      }
      ```

    * It is also possible that some partial data is supplied by ignoring optional details.

```
{
  "fullname": "Jagmeet Chautala",
  "emailaddress": "jagmeetc@example.com",
  "mobilenumber": "1234567890"
}
```

- **Normal Output**:

```
{
  "status": 200
  "message": "Created"
}
```

- **Exception Output**:

```
{
  "status": 421,
  "message": "Invalid field supplied"
}
```

- **Reset**:
  On-click, it empties all the fields of the form as also explained in item 3.1.2.1.2.
  - **Method**: Javascript function to reset all form-fields.

4.1.1.1.2 Links

4.1.1.2 Error-Message Box for Existing Record

Figure 2 shows the wireframe containing the error message described in this section.

4.1.1.2.1 Buttons

4.2 Level 4.2

4.2.1 Level 4.2.1

4.2.2 Level 4.2.2

4.2.3 Level 4.2.3

4.2.3.1 Level 4.2.3.1

4.2.3.2 Level 4.2.3.2

5 Database Schema

In this section, we describe the general database schema used in Blooper App.

5.1 Relational Database

Blooper App would be using HisSQL as its RDBMS in the backend. Multiple tables are required to implement the features expected from the this application. An RDBMS is required to maintain the integrity as well as consistency of the data-records pertaining to this application. Following details are pertinent to the RDBS schema used in Blooper App.

5.1.1 BlooperDB

BlooperDB is the name of the database stored in HisSQL.

1. Table: users

↰ org_admins: user_id
↰ bloopers: blooped_by
↰ bloopers: detected_by

field	*type*	*unique*	*null*	*default*
`id` 🔑↑	int			
`name`	varchar[64]	yes	no	
`email`	varchar[80]			null
`contact_no`	varchar[16]			null
`dob`	date		no	
`status`	char[1]		no	

- `email`: Only one email address is allowed in this field.
- `dob`: Date in ISO date format.
- `status`: (A)ctive, (I)nactive, (D)eleted

2. Table: organizations

↰ org_admins: org_id
↰ bloopers: affected_by

field	*type*	*unique*	*null*	*default*
`id` 🔑↑	int			
`name`	varchar[64]	yes	no	
`address_id` (🔑 addresses)	int		no	
`status`	char[1]		no	A

- `status`: (A)ctive, (I)nactive, (D)eleted

3. Table: org_admins

field	*type*	*unique*	*null*	*default*
`id` 🔑↑	int			
`org_id` (🔑 organizations)	int		no	
`user_id` (🔑 users)	int		no	
`since`	date		no	

- `since`: The date from which this user is assigned the role of administrator for this organization.

- One organization may have one ore more administrators. They are mapped using this table.
- The "since" field indicates a date when a particular administrator was mapped to an organization.

4. Table: addresses

↰ organizations: address_id

field	*type*	*unique*	*null*	*default*
`id` 🔑↑	int			
`address1`	varchar[128]		no	
`address2`	varchar[128]		yes	
`location_id` (🔑 locations)	int		no	

5. Table: locations

↰ addresses: location_id

field	type	unique	null	default
id 🔑↑	int			
name	varchar[64]		no	
state_id (🔑 states)	int		no	

- Unique(name, state_id)

6. Table: states

↰ locations: state_id

field	type	unique	null	default
id 🔑↑	int			
name	varchar[64]		no	
code	varchar[8]		yes	
country_id (🔑 countries)	int		no	

- Unique(country_id, code)
- Unique(country_id, name)

7. Table: countries

↰ states: country_id

field	type	unique	null	default
id 🔑↑	int			
code	varchar[4]	yes	no	
name	varchar[64]	yes	no	

8. Table: bloopers

field	type	unique	null	default
id 🔑↑	int			
blooped_by (🔑 users)	int		no	
loss_amount	decimal[.2]			
blooped_at	date		no	
detected_by (🔑 users)	int		no	
affected_by (🔑 organizations)	int		no	
meta_info	json		yes	

- `loss_amount`:
  - The money (in INR) lost due to this blooper.
  - Up to two decimal values, it is captured.

  `detected_by`: The user who detected this blooper.
- `affected_by`: The organization affected by this blooper.
- `meta_info`: Any additional information pertaining to this blooper would be stored here. There is no fixed schema for it. The object using this field is expected to know how to use this data.

6 Deployment Strategies

Blooper App is containerized using Docker. These docker based containers are deployed on virtual machines (VMs) running on the cloud. The VMs are usually deployed on AWS, or Azure clouds.
Kubernetes too can be used to manage the deployment of these applications.

* * * * *