

Project Specifications of Blooper App

(Version: 1.5)

Whoopie Bard <whoopie@clueless.dev>

Changu Bhai <changu.bhai@clueless.dev>

14 June, 2025



Non-Confidential



Contents

1 Document Info	2
1.1 Authors	2
1.2 Document Versions	2
1.3 Track Changes	2
2 Introduction	4
3 Functional Specifications	4
3.1 User Registration	4
3.1.1 Requirement Specifications	4
3.1.2 Wireframes	4
3.1.3 cut Subsubsection 3.1.3	7
3.1.4 Subsubsection 3.1.4	7
3.2 Use Cases	8
3.2.1 Application Administration	8
3.2.2 Generation of Bloopers	9
3.2.3 Blooper Control	10
3.2.4 MIS	10
4 Effort Estimations	10
5 Risks	11
6 Modules and Deliverables	12
7 Program Specifications	13
7.1 User-registration Module	13
7.1.1 Screens	13
7.1.2 Classes	15
7.2 Subsection 7.2	16
7.2.1 Subsubsection 7.2.1	16
7.2.2 Subsubsection 7.2.2	16
7.2.3 Subsubsection 7.2.3	16
8 Database Schema	16
8.1 Relational Databases	16
8.1.1 NEW+BlooperDB	17
9 Deployment Strategies	19



1 Document Info

1.1 Authors

Following authors were involved in preparation of this document.

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

1.2 Document Versions

Following table summarizes the details contained in various versions of this document.

Versions	Changes	Author	Date
v1.5	Added use-cases	whoopie	14 June, 2025
v1.4	Added section Track Changes	changu	9 February, 2025
v1.3	Added DB schema	whoopie	24 December, 2024
v1.2	Updated partial program specifications	whoopie	15 November, 2024
v1.1	Completed functional specifications	changu	28 October, 2024
v1.0	Added introductory text	whoopie	2 October, 2024

1.3 Track Changes

No.	Type	Changes
1	✖ CUT	No two user-accounts should be possible to be created using same mobile number.
2	+ NEW	Multiple user-accounts should be able to be created using the same mobile num...
3	✖ CUT	Subsubsection 3.1.3
4	✖ CUT	List Item 3.1.4.3.2.1
5	+ NEW	List Item 3.1.4.3.2.4
6	✖ CUT	{ "status": 421, "message": "Field supplied is not valid" }
7	+ NEW	{ "status": 422, "message": "Invalid field supplied" }
8	+ NEW	BlooperDB
9	+ NEW	code: varchar(8), null
10	✖ CUT	description: varchar(128), default null

... →



Project Specifications of Blooper App

No.	Type	Changes
11	✗ CUT	blooper.views

Non-Confidential



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.

[↑ User-registration Module](#)

3.1.1 Requirement Specifications

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

1. 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.
2. Users with an email address and mobile number should be able to create an account in Blooper.
3. No two user-accounts should be possible to be created using same email address.
4. ~~CUT~~ No two user-accounts should be possible to be created using same mobile number.
5. ~~NEW~~ Multiple user-accounts should be able to be created using the same mobile number.
6. User should be able to register from certain geographical regions only.

[↑ Track Changes: 1](#)

[↑ Track Changes: 2](#)

Q: How to prevent people from using VPN to beat geographical restrictions?
A: It is not possible to do.
Q: Is there any third party service, which provides such facilities?
A: Not known.

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.

[↑ Wireframes for User Registrat...](#)

[↑ User-registration Screen](#)

[↑ Login](#)



The diagram illustrates a user registration form. It features a header 'User Registration' enclosed in a dashed border. Below the header are four input fields: 'Full Name' (with a single-line input box), 'Email Address' (with a single-line input box), 'Mobile Number' (with a single-line input box), and 'Address' (with two stacked input boxes). At the bottom left is a 'Register' button, and at the bottom right is a 'Reset' button. A dashed line separates the form from a footer message: 'Already have an account? Log in here.'

Figure 1: User-registration Form

3.1.2.1.1 Form-Fields

Following fields are expected to be present in this form.

Field-Names	Required	Max-Size	Allowed Values	Unique
Email Address	Yes	char(128)	.a-zo-9-+@	Yes
Mobile Number	Yes	char(12)	+0123456789	Yes
Full Name	No	char(128)	A to Z, and a to z	No
Address	No	char(512)	A-Z, a-z, o-9, - +	No

↑ Row 1

- **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.

1. **Register:** Registers the new user.
 - (a) **Normal Flow**
 - i. System validates supplied data before creating user-account.
 - ii. If everything is in order, then system create new user-account.



- (b) **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.
2. **Reset:** Reset all fields of the registration form.
- (a) **Normal Flow**
- Clean-up contents of all fields.

[↑ Reset](#)

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.

[↑ Row 1](#)

3.1.2.2 Warning Message for Duplicate User-Account

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.

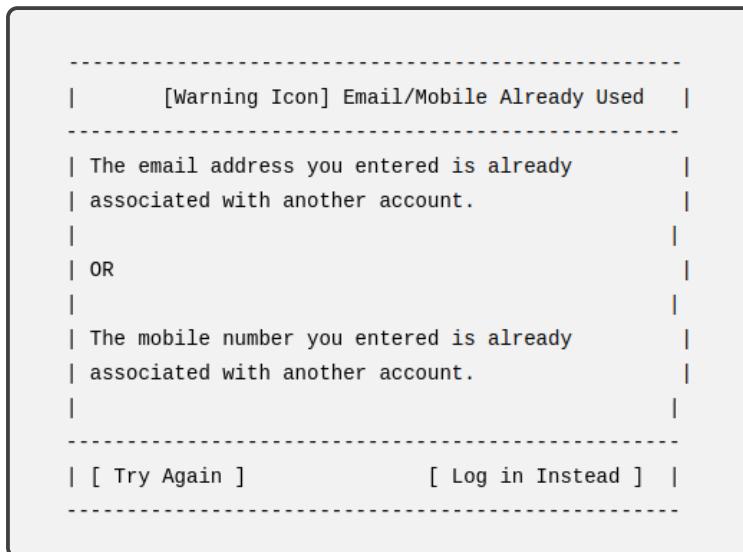


Figure 2: Message Box for Existing Record

[↑ Error-Message Box for Existin...](#)

3.1.2.2.1 Buttons

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



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?

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

3.1.3 ~~cur~~ Subsubsection 3.1.3

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

↑ Track Changes: 3

3.1.3.1 Sample Table

A Sample table is given below.

Cell 00	Header 1	Header 2	Header 3	Header 4
Row 1	Cell 1.1	21.05	Cell 1.3	Cell 1.4
Row 2	Cell 2.1	301.75	Cell 2.3	Cell 2.4
Row 3	Cell 3.1	8.18	Cell 3.3	Cell 3.4
Row 4	Cell 4.1	-76.41	Cell 4.3	Cell 4.4
Row 5	Cell 5.1	1.87	Cell 5.3	Cell 5.4
Total		256.44		

↑ Row 3

- **Row 1:**
 - First note for Row 1.
 - Second note for Row 1 which refers to [Login Here](#).
 - Third note for Row 1 (see also [Mobile Number](#)).
- **Row 2:** First note for Row2.
- **Row 3:**
 - First note for Row 3.
 - Second note for Row 3 depends on [Row 1](#).
 - Third note for Row 3.
 - Fourth note for Row 3.

3.1.4 Subsubsection 3.1.4

1. List Item 3.1.4.1
2. List Item 3.1.4.2
 - (a) List Item 3.1.4.2.1
 - (b) List Item 3.1.4.2.2
 - (c) List Item 3.1.4.2.3
 - (d) List Item 3.1.4.2.4
 - (e) List Item 3.1.4.2.5
3. List Item 3.1.4.3
 - (a) List Item 3.1.4.3.1
 - (b) List Item 3.1.4.3.2



- i. ~~CUT~~ List Item 3.1.4.3.2.1
 - ii. List Item 3.1.4.3.2.2
 - iii. List Item 3.1.4.3.2.3
 - iv. **NEW+** List Item 3.1.4.3.2.4
 - v. List Item 3.1.4.3.2.5
 - vi. List Item 3.1.4.3.2.6
- (c) List Item 3.1.4.3.3

Track Changes: 4

Track Changes: 5

Be warned!!!

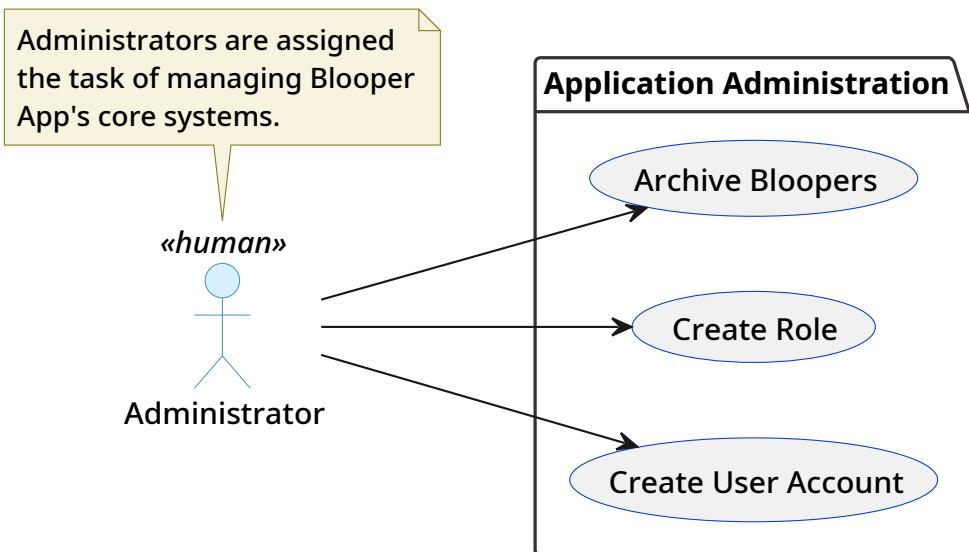
- (d) List Item 3.1.4.3.4
- i. List Item 3.1.4.3.4.1
 - ii. List Item 3.1.4.3.4.2
 - iii. List Item 3.1.4.3.4.3
 - iv. List Item 3.1.4.3.4.4

3.2 Use Cases

Following use-cases are applicable for the Blooper App.

3.2.1 Application Administration

This package contains usecases pertaining to the administrative activities of the application.



Create User Account

System provides a form to enter details of user-account.

User enters mandatory and optional data into the form.



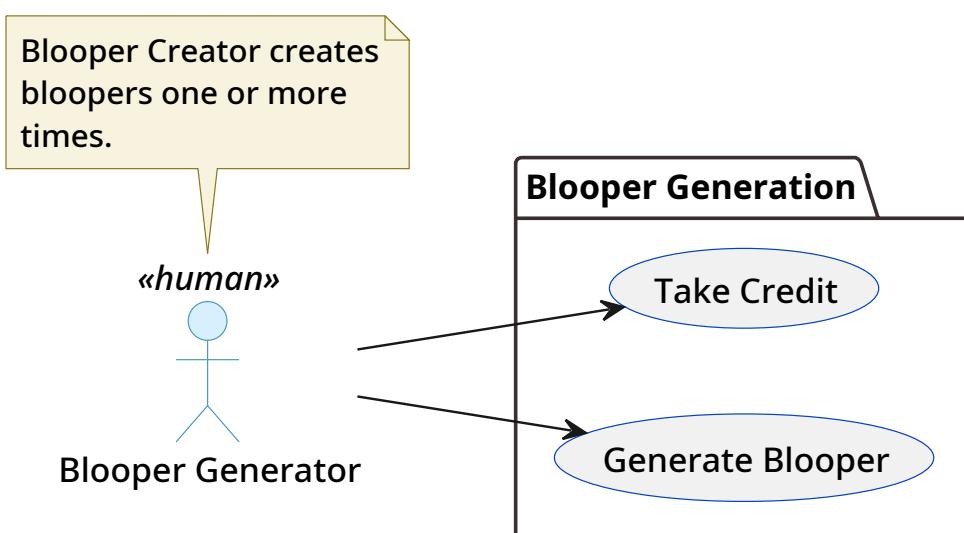
Normal Flow	Create User Account
	User submits the form.
	System persists the data in a database.

	Create Role
	System provides a form to enter details of user-roles.
Normal Flow	User enters mandatory and optional data into the form.
	User submits the form.
	System persists the data in a database.

	Archive Bloopers
Preconditions	One or more bloopers with remedial actions must exist.
	System lists down all bloopers having respective remedial actions taken.
	User selects one or more bloopers from list.
Normal Flow	User submits selected bloopers for archival.
	System moves the selected bloopers to the archive.
	System removes the selected bloopers from the list.
Alternate Flow	If one or more selected bloopers are marked for not archival, do not list them in archival screen.
Exception Flow	In case moving blooper to archive fails, do not remove it from the blooper list.

3.2.2 Generation of Bloopers

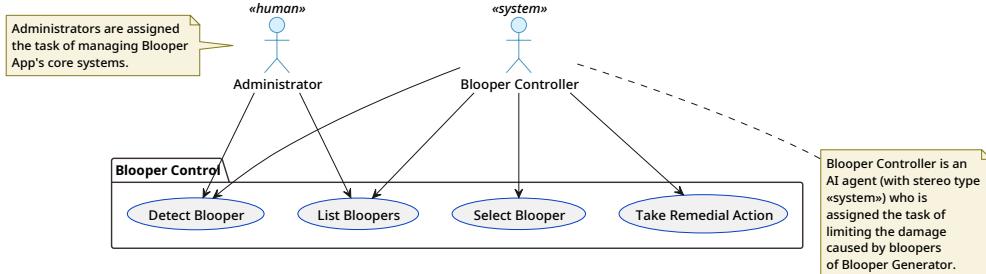
This package defines usecases involved in generation of bloopers.





3.2.3 Blooper Control

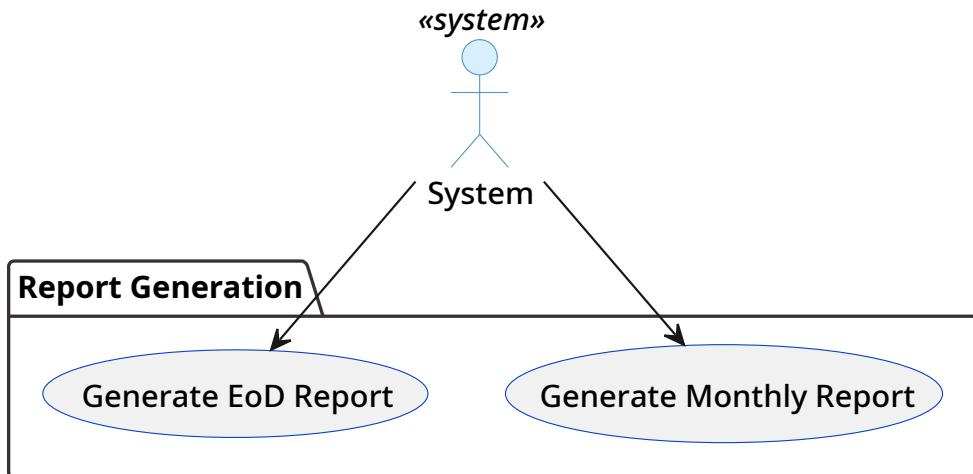
Following are the usecases pertaining to controlling bloopers.



Select Blooper	
Preconditions	One or more bloopers must exist already.
Normal Flow	Search criteria is supplied by user to search bloopers based on them. System returns a list of matching bloopers which is rendered in a table with pagination enabled.
Alternate Flow	System returns an empty list to indicate that no bloopers matched supplied search criteria.
Exception Flow	User doesn't have rights to search and list bloopers.

3.2.4 MIS

All report-related usecases are collected here.



4 Effort Estimations

The estimation of work is calculated using function-point analysis method. It was reviewed by following people:

1. Mr. Changu Bhai



2. Mrs. Mai In Jun

Following table summarizes the approximate efforts (in person-hours) required for designing, developing, preparation of unit tests, functional testing, and documentation for each module of this application.

Modules	Des.	Dev.	U. Test.	F. Test.	Doc.	Remarks
User Registration	2	9	6	2	1	
Password Reset	3	4	4	6	2	
Blooper Generation	4	10	8	2	2	Complex module
Total	9	23	18	10	5	

5 Risks

Following risks are identified for this project.

1. Developer Attrition

Risk-ID: RSK-01

Risk-Types: Time, Financial

- It is possible that skilled developers leave the project mid-way of execution.
- As a remedy, a pool of skilled resources should be kept on bench, which might increase the financial risk.

2. Delay in Communication with Client

Risk-ID: RSK-02

Risk-Types: Time

- The delay in receiving clarifications from client on critical matters would affect the timelines of delivery.
- The turnaround time for such communication threads should be well-defined and included in the contract-terms to mitigate this risk.

3. Usage of unsupported Open Source solutions

Risk-ID: RSK-03

Risk-Types: Compliance

- The data-cleaning solution used in this application is not maintained by the open source community anymore. Its last update was found to be four years old.
- As a mitigation strategy, funding should be provided to revive and maintain this open source project and release the new version with fixes for reported bugs.



6 Modules and Deliverables

Following modules and deliverables are identified for phase 1 of this project.

Deliverable-ID: DLV-01

Deliverable LDAP based User Authentication Module

Accountability Mohan Kumar

Date of Delivery November 30, 2025

1. Implementation of module for user-authentication using LDAP.
2. An extension library named blooper_ldap_ext which will allow other applications to use Blooper App as their SSO service provider by using its own LDAP backend.

Deliverable-ID: DLV-02

Deliverable Program Specification for SSO Integration

Accountability Mai In Jun

Date of Delivery November 08, 2025

Deliverable-ID: DLV-03

Deliverable Password Reset Form

Accountability <not specified>

Date of Delivery November 12, 2025

Deliverable-ID: DLV-04

Deliverable Wireframes for User Authentication UI

Accountability Uday Khedekar

Date of Delivery November 22, 2025

Deliverable-ID: DLV-05

Deliverable Wireframes for User Registration UI

Accountability Shabnam Ansari, Ramji D'souza

Date of Delivery November 16, 2025

Screen mockups for user registration module is to be developed. The wireframes for the same is available at [User-registration Form](#).

**Deliverable-ID: DLV-06****Deliverable** Realm-restriction Middleware**Accountability** Josephine Walters**Date of Delivery** November 19, 2025

A middleware to ensure that incoming requests from the authenticated users do not cross the realm-boundary. The realm-code applicable for that user would be injected automatically into the request.

Deliverable-ID: DLV-07**Deliverable** Email based Password Reset Module**Accountability** Uday Khedekar**Date of Delivery** November 16, 2025

Users should receive an email containing link to reset the password, subject to additional confirmation of date of birth, or some other secret questions set by the user.

7 Program Specifications

7.1 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 [User Registration](#).

7.1.1 Screens

Following screens are developed for user-registration purpose.

7.1.1.1 User-registration Screen

Please refer [User-registration Form](#) for wireframe of user-registration screen.

7.1.1.1.1 Buttons

- **Register**

The [Normal Flow](#) provides the details of the normal flow of actions when this button is clicked.

The [Exception Flow](#) provides the details of the exception flow of actions when this button gets clicked.

- **API Endpoint:** /api/v1/users/register
- **Method:** POSTHTTP 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.

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

This is the JSON format of input.

```
* {  
    "fullname": "Jagmeet Chautala",  
    "emailaddress": "jagmeetc@clueless.dev",  
    "mobilenumber": "1234567890"  
}
```

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

– **Normal Output**

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

– **Exception Output**

```
* CUT  
{  
    "status": 421,  
    "message": "Field supplied is not valid"  
}  
  
* NEW  
{  
    "status": 422,  
    "message": "Invalid field supplied"  
}
```

↑ Track Changes: 6

↑ Track Changes: 7

• **Reset**

On-click, it empties all the fields of the form as also explained in [Reset](#).

– **Method:** Javascript function to reset all form-fields.

7.1.1.2 Links

- [Login](#)

This hyperlink takes the user to the login page as shown in [User-registration Form](#).

- [CDC](#)



This hyperlink takes the user to an external site, which is official website of the company Clueless Developers' Consortium.

7.1.1.2 Error-Message Box for Existing Record

[Message Box for Existing Record](#) shows the wireframe containing the error message described in this section.

7.1.1.2.1 Buttons

1. **Try Again:** Clicking on this button takes the user to the pre-populated user-registration page.
2. **Login Instead:** Clicking on this button takes the user to the login page.

7.1.2 Classes

Following classes are used in this module.

7.1.2.1 User

The User class acts mostly like a data container, in which all user-related attributes are captured. But at the same time it allows certain methods like deactivate() to exist which, as its name suggests, deactivates the underlying user-account.

7.1.2.1.1 Attributes

Following attributes are defined in the User class:

- **username:** A string containing the username which is used to login to the application.
- **password_hash:** A string containing the hash of the user-password generated by bcrypt
- **email:** A string containing the email address of the user
- **status:** A string which has following valid values:
 - **A:** Active
 - **I:** Inactive
 - **D:** Deleted

7.1.2.1.2 Methods

Details of methods defined for this class are given below.

1. **get_current_blooper_count()**
 - **Type:** Public
 - **Arguments:** None
 - **Returns:** A number indicating the current blooper-counts of the user
 - **Raises:** NothingNo exceptions are raised in this method.
2. **get_reportees()**



- **Type:** Public
- **Arguments:** None
- **Returns:** It returns a JSON string containing the User objects associated with the reportees of this user. A sample JSON string is given below.

```
[  
  {  
    "username": "Jagmeet Chautala",  
    "email": "jagmeetc@clueless.dev"  
  },  
  {  
    "username": "Madhuri Varma",  
    "email": "madhuri.verma@clueless.dev"  
  }  
]
```

- **Raises**

– **NotFound:** No reportees are found for this user.

3. deactivate()
Deactivates the user-account by setting its attribute "status" to "I".

- **Type:** Public
- **Arguments:** None
- **Returns:** Nothing
- **Raises**

– **OperationFailed:** The user-record is already inactive.

7.2 Subsection 7.2

- 7.2.1 Subsubsection 7.2.1
- 7.2.2 Subsubsection 7.2.2
- 7.2.3 Subsubsection 7.2.3
 - 7.2.3.1 Subsubsubsection 7.2.3.1
 - 7.2.3.2 Subsubsubsection 7.2.3.2

8 Database Schema

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

8.1 Relational Databases

Blooper App would be using HisSQL as its RDBMS in the backend. Multiple tables are required to implement the features expected from 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.



Track Changes: 8

8.1.1 NEW+ BlooperDB

BlooperDB is the name of the database stored in HisSQL.

Following are the tables defined in this database. The specifications of the fields in each of the tables are given too. If certain fields have any specific notes associated with them, then they are displayed at the right side panel of each table.

1. users

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. organizations

field	type	unique	null	default
id	int			
name	varchar[64]	yes	no	
address_id <small>(P addresses)</small>	int		no	
status	char[1]		no	A

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

3. org_admins

field	type	unique	null	default
id	int			
org_id <small>(P organizations)</small>	int		no	
user_id <small>(P users)</small>	int		no	
since	date		no	

- One organization may have one or more administrators. They are mapped using this table.
- The "since" field indicates a date when a particular administrator was mapped to an organization.
- since: The date from which this user is assigned the role of administrator for this organization.



4. addresses

field	type	unique	null	default
↑ organizations: address.id				
id ↗↑	int			
address1	varchar[128]		no	
address2	varchar[128]		yes	
location_id (↗ locations)	int		no	

5. locations

field	type	unique	null	default
↑ addresses: location.id				
id ↗↑	int			
name	varchar[64]		no	
state_id (↗ states)	int		no	

- Unique(name, state.id)

6. states

field	type	unique	null	default
↑ locations: state.id				
id ↗↑	int			
name	varchar[64]		no	
NEW+ code	varchar[8]		yes	
country_id (↗ countries)	int		no	

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

7. countries

field	type	unique	null	default
↑ states: country.id				
id ↗↑	int			
code	varchar[4]	yes	no	
name	varchar[64]	yes	no	
CUT* description	varchar[128]			null

- code: Usually two-character country code.

↑ Track Changes: 9

↑ Track Changes: 10



8. ~~CUTX~~ blooper_views

field	type	unique	null	default
id ↗↑	int			
blooper_id (↗ bloopers)	int		no	
viewed_by_id (↗ users)	int		no	
viewed_at	datetime		no	now()

↳ Track Changes: 11

↳ Track Changes: 11

9. bloopers

field	type	unique	null	default
id ↗↑	int			
blooper_id (↗ users)	int		no	
loss_amount	decimal[10,2]			
bloopered_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.

9 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.

* * * * *

(Document prepared on 24 February, 2026 at 12:25:44 PM IST)