System Requirements Specification Index

For

Blood Donors Application

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Contents

1 BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

The Blood Donor Application is a critical tool designed to streamline the process of managing and accessing donor information efficiently.

Your task is to develop a comprehensive application that enables users to add new donors, modify or delete donor entries. Additionally, the application should offer robust search functionality to locate donors based on State, City, or Blood Group, enhancing its usability in real-world scenarios.

1.1.1 Blood Donor Application:

The Blood Donor Application allows you to:

1. Access the home page.
2. Should be able to add a new donor.
3. It should have basic fields like first name, last name, email, mobile number, gender, state, city and blood group.
4. Should be able to get the list of donors along with options to sort in ascending and descending order in each field.
5. Should be able to edit and delete any donor.
6. Should be able to search for any donor by State, City or Blood Group.

2. Template Code Structure:

2.1 Donor Controller

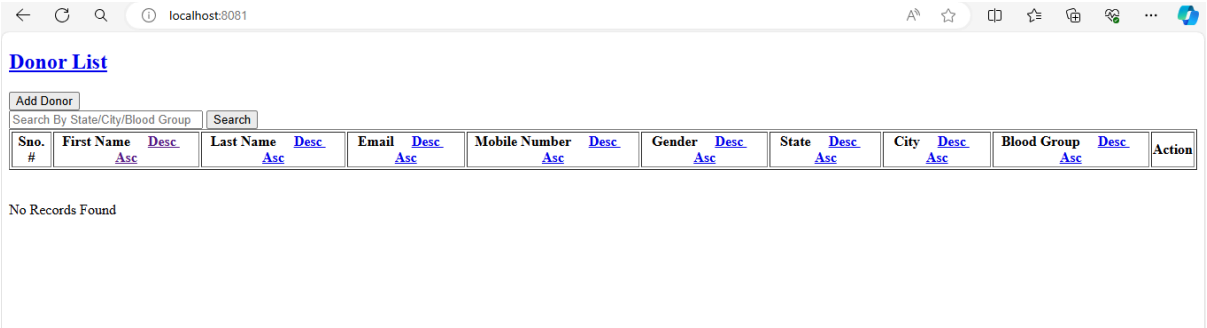
Method Exposed	Purpose
listDonor()	Should return page "list-donors" with required data.
showFormForAdd()	Should return page "add-donor-form" for adding a donor.
saveDonor()	Should save a donor and return "donor/list" with required data.
showFormForUpdate()	Should show donor details in page "update-donor-form" to edit an donor.
deleteDonor()	Should delete a donor and return "donor/list" with required data.
searchDonors()	Should search a donor and return "list-donors" with required data.

2.2 Resources Available:

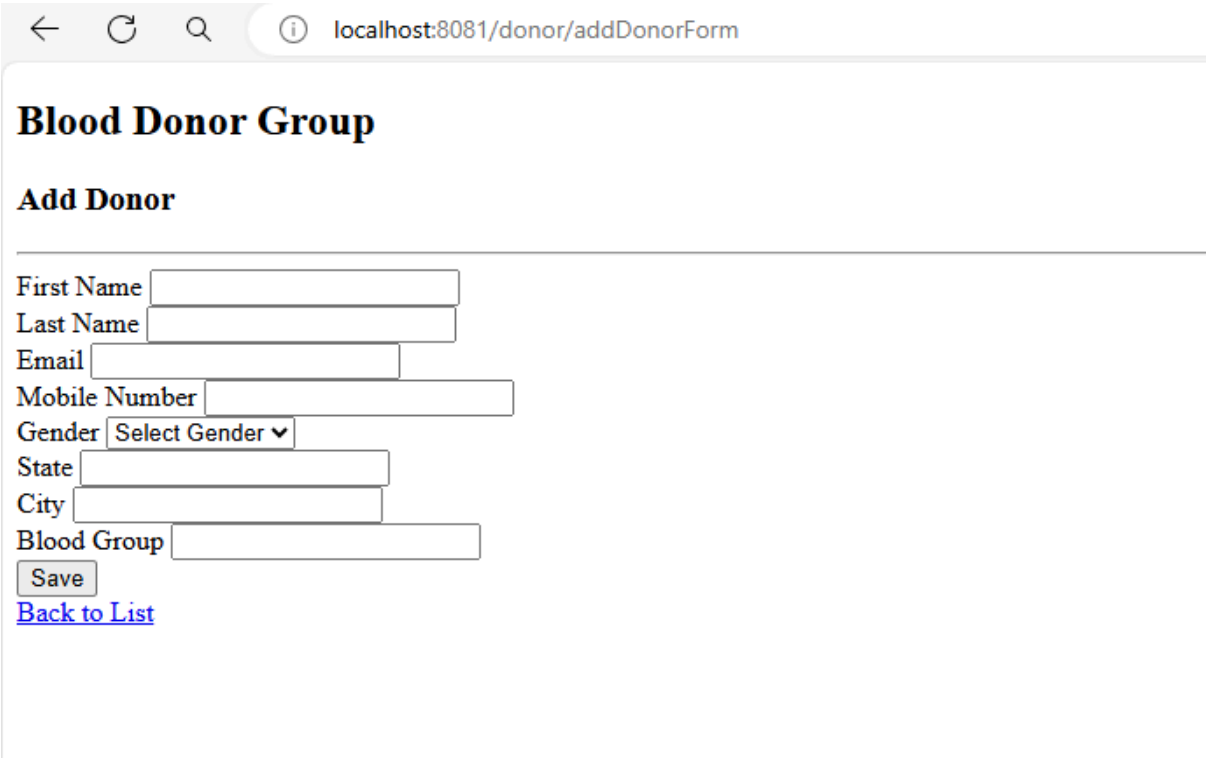
Description	View Pages Name	Remarks
Common UI		
Home Page	list-donors	Contains a homepage which shows a list of all donors along with options to add, edit , delete and search a donor.
All donors	list-donors	
Add a donor	add-donor-form	
Update a donor	update-donor-form	
Search a donor	list-donors	

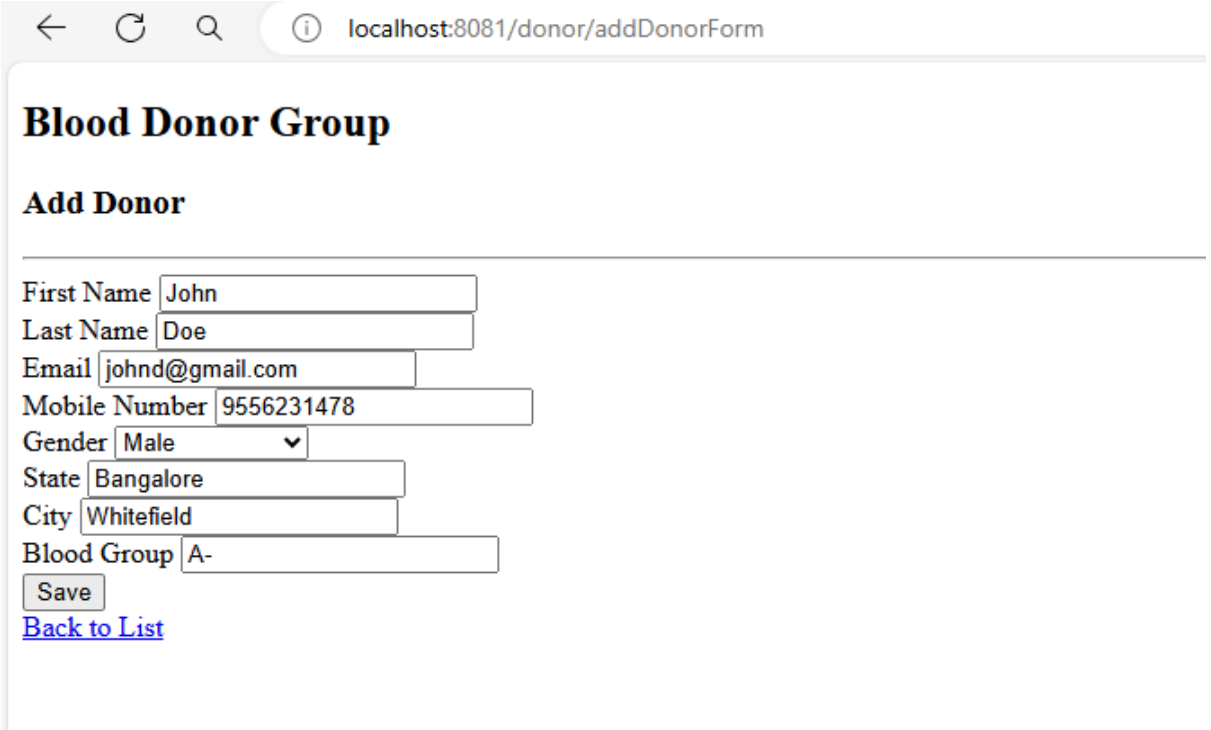
3 Suggested Wireframes:

1. Homepage – Visitor Landing Page

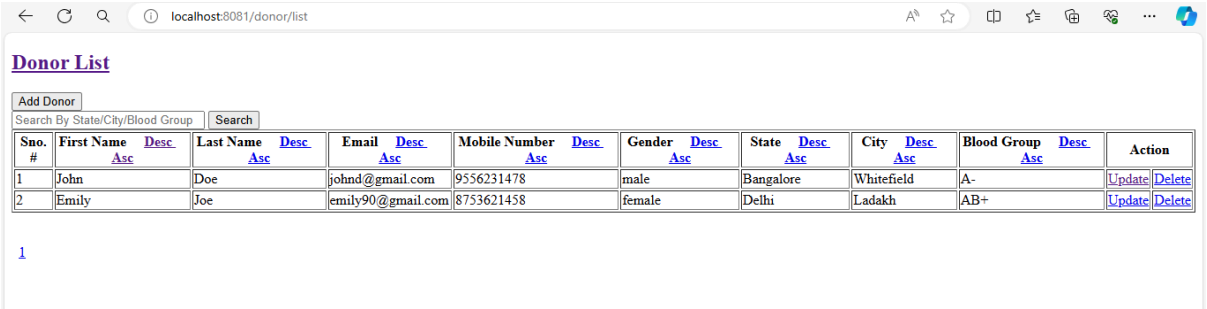


2. Create a Donor



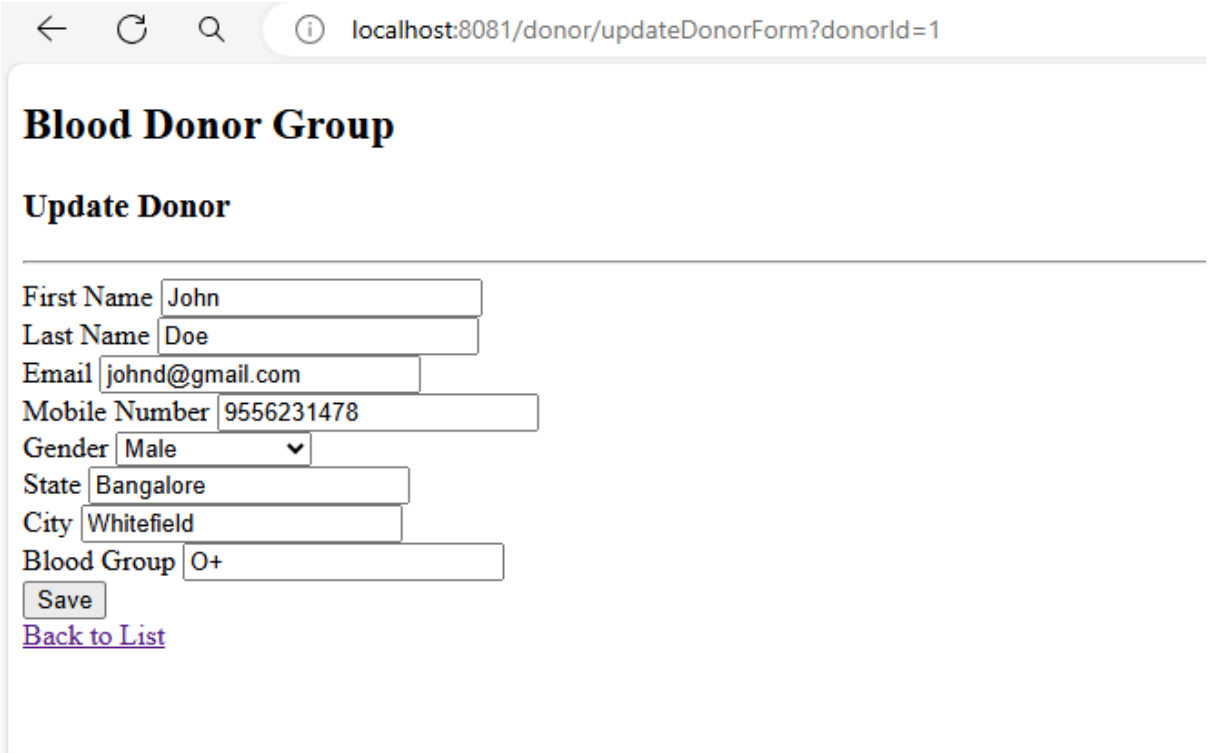


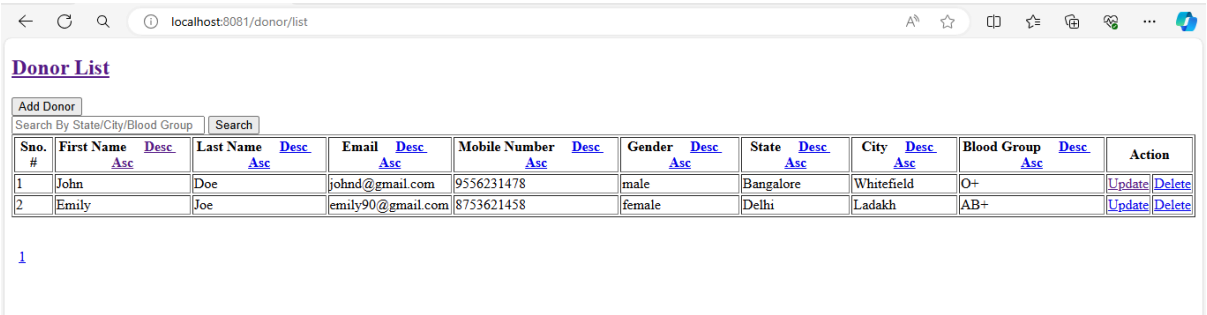
3. Donor List



Sno. #	First Name Desc Asc	Last Name Desc Asc	Email Desc Asc	Mobile Number Desc Asc	Gender Desc Asc	State Desc Asc	City Desc Asc	Blood Group Desc Asc	Action	
1	John	Doe	johnd@gmail.com	9556231478	male	Bangalore	Whitefield	A-	Update	Delete
2	Emily	Joe	emily90@gmail.com	8753621458	female	Delhi	Ladakh	AB+	Update	Delete

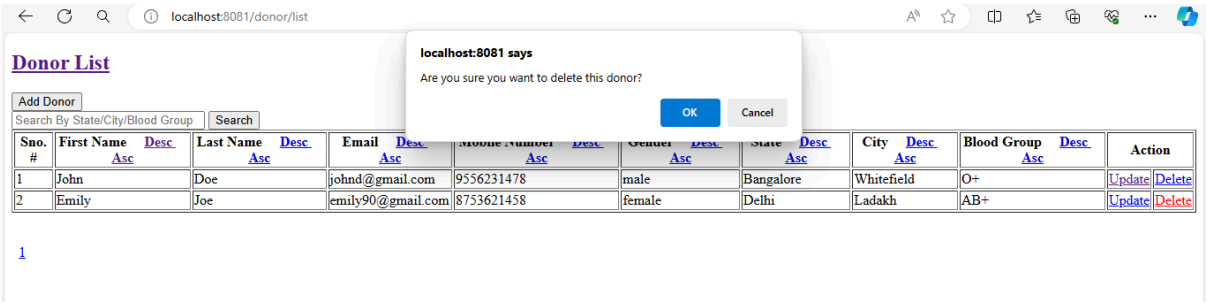
4. Update a Donor

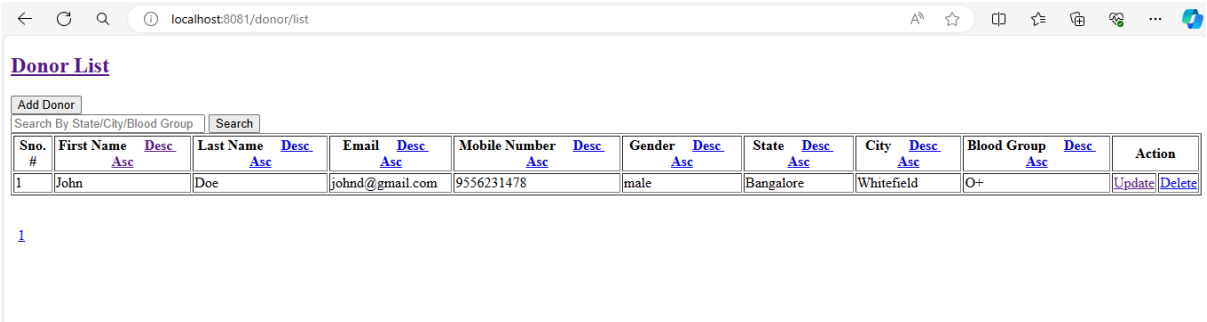




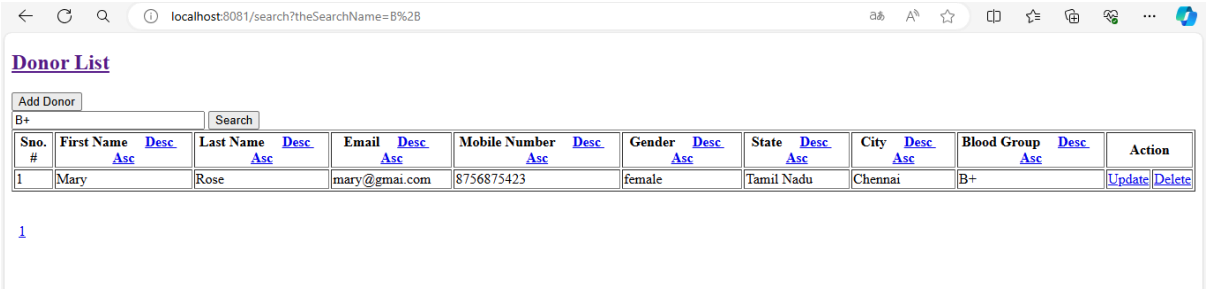
Sno. #	First Name Desc Asc	Last Name Desc Asc	Email Desc Asc	Mobile Number Desc Asc	Gender Desc Asc	State Desc Asc	City Desc Asc	Blood Group Desc Asc	Action	
1	John	Doe	johnd@gmail.com	9556231478	male	Bangalore	Whitefield	O+	Update	Delete
2	Emily	Joe	emily90@gmail.com	8753621458	female	Delhi	Ladakh	AB+	Update	Delete

5. Delete a Donor





6. Search a Donor



4 Business Validations

1. Id must be of type id.
2. First name value is not blank, min 2 and max 40 characters.
3. Last name value is not blank, min 2 and max 40 characters.
4. Email value is not blank and of type email.
5. Mobile number is not null and must be of length 10.
6. Gender should not be blank
7. State should not be blank.
8. City should not be blank.
9. Blood groups should not be blank and min 1 and max 3 characters.

5 Considerations

The Code template already contains skeleton methods for service and controller layer. Please write your logic in it.

6 Method Descriptions

1. Service Class - Method Descriptions

a. DonorService – Method Descriptions

- Constructor-based injection is used to initialize `DonorRepository`.

Method	Task	Implementation Details
`public DonorService(DonorRepository donorRepository)`	Constructor injection	- Injects `DonorRepository` through constructor and assigns it to a private final field

Method	Task	Implementation Details
`getDonors()`	To fetch all donors	- Calls `donorRepository.findAll()` - Returns a list of all donors
`saveDonor(Donor donor)`	To save or update donor	- Calls `donorRepository.save(donor)` - Persists a new or updated donor and returns the saved object
`getDonor(Long id)`	To fetch a donor by ID	- Calls `donorRepository.findById(id).get()` - Returns the donor object associated with the given ID
`deleteDonor(Long id)`	To delete a donor by ID	- Calls `donorRepository.deleteById(id)` - Returns `true` after deletion
`searchDonors(String theSearchName, Pageable pageable)`	To search donors based on name, city, state, group	- If `theSearchName` is non-empty, calls: `donorRepository.findByStateAndCityAndBloodGroup(theSearchName, pageable)` - Else, passes `null` for a full list

2. Controller Class - Method Descriptions

a. DonorController – Method Descriptions

- Declare a private variable named `donorService` of type `DonorService` and inject it using `@Autowired`.

Method	Task	Implementation Details
`@Autowired private DonorService donorService`	Field-based dependency injection	- Annotated with `@Autowired` - Injects `DonorService` instance
`@InitBinder` `public void InitBinder(WebDataBinder dataBinder)`	Trim whitespace from String inputs	- Registers `StringTrimmerEditor` to remove leading/trailing whitespaces from all incoming `String` fields before binding

Method	Task	Implementation Details
`listDonors()`	To list or search donors with pagination	- Accepts optional `theSearchName` query param - Accepts pageable data (default size = 5) - Calls `donorService.searchDonors(...)` - Adds results and metadata to the model - Returns `list-donors` view
`showFormForAdd()`	To show add donor form	- Adds a new `Donor` object to the model - Returns `add-donor-form` view
`saveDonor()`	To save or update donor based on validation	- Accepts donor object with `@Valid` and checks binding result - If validation fails: – If `donor.id != null`, returns `update-donor-form` – Else returns `add-donor-form` - If valid, saves using `donorService.saveDonor(...)`

		- Redirects to `/donor/list`
`showFormForUpdate()`	To show update donor form	- Fetches donor by ID using `@RequestParam` - Adds the donor object to model - Returns `update-donor-form` view
`deleteDonor()`	To delete donor by ID	- Accepts `donorId` as request parameter - Calls `donorService.deleteDonor(id)` - Redirects to `/donor/list`

7 Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal**
3. **To build your project use command:**
   mvn clean package -Dmaven.test.skip
4. **To launch your application:**
   java -jar <your application war file name>
5. **This editor Auto Saves the code**
6. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
7. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**
8. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
   Note: The application will not run in the local browser
9. **Default credentials for MySQL:**
   a. **Username: root**
   b. **Password: pass@word1**

11. To login to mysql instance: Open new terminal and use following command:

a. sudo systemctl enable mysql
b. sudo systemctl start mysql

NOTE: After typing any of the above commands you might encounter any warnings.

>> Please note that these warnings are expected and can be disregarded. Proceed to the next step.

c. mysql -u root -p

The last command will ask for password which is 'pass@word1'

12. Mandatory: Before final submission run the following command:

mvn test