# Software Requirements Specification

## Library Management System

## Table of Contents

1. Introduction  
   1.1. Purpose  
   1.2. Document Conventions  
   1.3. Intended Audience and Reading Suggestions  
   1.4. Project Scope  
   1.5. References
2. Overall Description  
   2.1. Product Perspective  
   2.2. Product Features  
   2.3. User Classes and Characteristics  
   2.4. Operating Environment  
   2.5. Design and Implementation Constraints  
   2.6. Assumptions and Dependencies
3. System Features  
   3.1. Authentication  
   3.1.1. User Registration  
   3.1.2. User Login  
   3.1.3. User Logout  
   3.2. Dashboard (Librarian)  
   3.3. Book Management  
   3.4. Author Management  
   3.5. Genre Management  
   3.6. Borrowal Management  
   3.7. User Management (Librarian)  
   3.8. Review Management
4. External Interface Requirements  
   4.1. User Interfaces  
   4.2. Software Interfaces
5. Non-Functional Requirements  
   5.1. Security Requirements  
   5.2. Usability Requirements  
   5.3. Maintainability Requirements
6. Detailed Data Model / Data Dictionary  
   6.1. User Entity  
   6.2. Book Entity  
   6.3. Author Entity  
   6.4. Genre Entity  
   6.5. Borrowal Entity  
   6.6. Review Entity  
   6.7. Entity Relationship Overview (Conceptual)
7. Key Use Cases  
   7.1. UC-001: User Login  
   7.2. UC-002: Add New Book (Librarian)  
   7.3. UC-003: Borrow a Book (Member)  
   7.4. UC-004: Register New User (Librarian)  
   7.5. UC-005: View Borrowal History (Member)

## 1. Introduction

### 1.1. Purpose

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for the Library Management System. The system aims to provide a platform for managing library resources, including books, authors, genres, and borrowals, as well as managing user accounts for librarians and members.

### 1.2. Document Conventions

This document uses standard conventions for SRS. "Shall" is used to indicate mandatory requirements.

### 1.3. Intended Audience and Reading Suggestions

This document is intended for project stakeholders, including developers, testers, project managers, and clients. Readers should have a basic understanding of web applications and library management concepts. It is recommended to read the document sequentially, though specific sections can be referred to as needed.

### 1.4. Project Scope

The Library Management System will be a full-stack web application with the following key capabilities:

* User authentication and role-based access control (Librarian, Member).
* Management of library inventory: books, authors, genres.
* Tracking of book borrowals and returns.
* Management of user accounts.
* Viewing capabilities for members.
* Administrative capabilities for librarians.

The project consists of a backend API built with Node.js and Express.js, a MongoDB database, and a frontend client application built with React and Material-UI.

### 1.5. References

* Project README (LibraryManagement/README.md)
* Server-side code (LibraryManagement/server/)
* Client-side code (LibraryManagement/client/)

## 2. Overall Description

### 2.1. Product Perspective

The Library Management System is a self-contained web application designed to replace or augment manual library operations. It interacts with users (Librarians and Members) through a web browser. The system relies on a MongoDB database for data persistence.

### 2.2. Product Features

The major features of the Library Management System are:

* User Authentication (Login, Logout, Registration by Admin)
* Dashboard for Librarians with system overview statistics.
* CRUD (Create, Read, Update, Delete) operations for Books.
* CRUD operations for Authors.
* CRUD operations for Genres.
* CRUD operations for Borrowals.
* CRUD operations for Users (accounts).
* CRUD operations for Reviews.
* Role-based access control differentiating Librarian and Member functionalities.

### 2.3. User Classes and Characteristics

There are two primary user classes:

* **Librarian (Admin):**
  + Responsible for managing the library's inventory, user accounts, and borrowal records.
  + Has full CRUD access to all system entities (Books, Authors, Genres, Borrowals, Users).
  + Can view system statistics on the dashboard.
  + Technical proficiency: Expected to be comfortable using web applications.
* **Member:**
  + Registered users of the library.
  + Can view books, authors, and genres.
  + Can view their own borrowal history.
  + Can create new borrowal requests for themselves.
  + Technical proficiency: Basic web browsing skills.

### 2.4. Operating Environment

* **Server-side:** Node.js runtime environment, MongoDB database. Can be deployed using Docker.
* **Client-side:** Modern web browser (e.g., Chrome, Firefox, Edge, Safari) with JavaScript enabled.
* The application is designed to be accessed over HTTP/HTTPS.

### 2.5. Design and Implementation Constraints

* The backend shall be implemented using Node.js and Express.js.
* The database shall be MongoDB.
* The frontend shall be implemented using React and Material-UI.
* User authentication shall use Passport.js with a local strategy (email and password).
* The system shall provide a RESTful API for communication between the client and server.

### 2.6. Assumptions and Dependencies

* Users have a stable internet connection.
* The server environment (Node.js, MongoDB) is properly configured and running.
* The client environment (web browser) supports modern JavaScript and CSS.
* Environment variables (e.g., database connection URI, session secret) are correctly configured.

## 3. System Features

### 3.1. Authentication

#### 3.1.1. User Registration

* **Description:** Librarians shall be able to create new user accounts (both Librarians and Members). Members cannot register themselves directly.
* **Inputs:** User details (name, email, password, date of birth, phone, isAdmin flag, photo URL).
* **Processing:** The system shall validate the input data. The password shall be hashed and salted before storing. A new user record shall be created in the database.
* **Outputs:** Confirmation of user creation or an error message.
* **API Endpoint:** POST /api/auth/register

#### 3.1.2. User Login

* **Description:** Registered users (Librarians and Members) shall be able to log in to the system.
* **Inputs:** Email and password.
* **Processing:** The system shall verify the credentials against stored user records. Upon successful authentication, a session shall be established.
* **Outputs:** Access to role-specific features or an error message. User data (including isAdmin flag) is returned to the client.
* **API Endpoint:** POST /api/auth/login

#### 3.1.3. User Logout

* **Description:** Logged-in users shall be able to log out of the system.
* **Processing:** The system shall terminate the user's session.
* **Outputs:** User is redirected to the login page or receives a confirmation message.
* **API Endpoint:** GET /api/auth/logout

### 3.2. Dashboard (Librarian)

* **Description:** Librarians shall be able to view a dashboard with an overview of library statistics.
* **Details:** The dashboard displays widgets for metrics such as "Weekly Sales," "New Users," "Item Orders," and "Bug Reports" (Note: The data sources for these specific widgets are placeholders in the current client code and may not directly reflect actual library operations like "sales" or "bug reports" without further backend integration for these specific metrics). It also shows charts for "Website Visits" and "Current Visits" (again, these are example charts).
* **Access:** Librarian role only.
* **Client Route:** /dashboard

### 3.3. Book Management

* **Description:** The system shall allow management of book records.
* **Data Attributes:** Name, ISBN, Author, Genre, Availability status, Summary, Photo URL.
* **Librarian Functions:**
  + Shall be able to add new books. (API: POST /api/book/add)
  + Shall be able to view a list of all books. (API: GET /api/book/getAll)
  + Shall be able to view details of a specific book. (API: GET /api/book/get/:id)
  + Shall be able to update existing book details. (API: PUT /api/book/update/:id)
  + Shall be able to delete books. (API: DELETE /api/book/delete/:id)
* **Member Functions:**
  + Shall be able to view a list of all books.
  + Shall be able to view details of a specific book.
* **Client Route:** /books

### 3.4. Author Management

* **Description:** The system shall allow management of author records.
* **Data Attributes:** Name, Description, Photo URL.
* **Librarian Functions:**
  + Shall be able to add new authors. (API: POST /api/author/add)
  + Shall be able to view a list of all authors. (API: GET /api/author/getAll)
  + Shall be able to view details of a specific author. (API: GET /api/author/get/:id)
  + Shall be able to update existing author details. (API: PUT /api/author/update/:id)
  + Shall be able to delete authors. (API: DELETE /api/author/delete/:id)
* **Member Functions:**
  + Shall be able to view a list of all authors.
  + Shall be able to view details of a specific author.
* **Client Route:** /authors

### 3.5. Genre Management

* **Description:** The system shall allow management of genre records.
* **Data Attributes:** Name, Description.
* **Librarian Functions:**
  + Shall be able to add new genres. (API: POST /api/genre/add)
  + Shall be able to view a list of all genres. (API: GET /api/genre/getAll)
  + Shall be able to view details of a specific genre. (API: GET /api/genre/get/:id)
  + Shall be able to update existing genre details. (API: PUT /api/genre/update/:id)
  + Shall be able to delete genres. (API: DELETE /api/genre/delete/:id)
* **Member Functions:**
  + Shall be able to view a list of all genres.
  + Shall be able to view details of a specific genre.
* **Client Route:** /genres

### 3.6. Borrowal Management

* **Description:** The system shall allow management of book borrowal records.
* **Data Attributes:** Book ID, Member ID, Borrowed Date, Due Date, Status (e.g., Borrowed, Returned, Overdue).
* **Librarian Functions:**
  + Shall be able to add new borrowal records. (API: POST /api/borrowal/add)
  + Shall be able to view a list of all borrowals. (API: GET /api/borrowal/getAll)
  + Shall be able to view details of a specific borrowal. (API: GET /api/borrowal/get/:id)
  + Shall be able to update existing borrowal records (e.g., mark as returned, change due date). (API: PUT /api/borrowal/update/:id)
  + Shall be able to delete borrowal records. (API: DELETE /api/borrowal/delete/:id)
* **Member Functions:**
  + Shall be able to add new borrowal requests for themselves. (API: POST /api/borrowal/add, client ensures memberId is current user)
  + Shall be able to view their own borrowal history. (Client filters results from GET /api/borrowal/getAll)
* **Client Route:** /borrowals

### 3.7. User Management (Librarian)

* **Description:** Librarians shall be able to manage user accounts.
* **Data Attributes:** Name, Email, Date of Birth, Phone, Role (isAdmin flag), Photo URL.
* **Librarian Functions:**
  + Shall be able to add new users (Librarians or Members). (API: POST /api/user/add, also covered by POST /api/auth/register)
  + Shall be able to view a list of all users. (API: GET /api/user/getAll)
  + Shall be able to view a list of all members (users with isAdmin=false). (API: GET /api/user/getAllMembers)
  + Shall be able to view details of a specific user. (API: GET /api/user/get/:id)
  + Shall be able to update existing user details (excluding password directly, password reset might be a separate feature or handled by updating hash/salt if password field is provided). (API: PUT /api/user/update/:id)
  + Shall be able to delete users. (API: DELETE /api/user/delete/:id)
* **Client Route:** /users

### 3.8. Review Management

* **Description:** The system shall allow management of book reviews. (Note: Client-side UI for review submission/display by members is not explicitly detailed in the provided page components, but backend infrastructure exists.)
* **Data Attributes:** Book ID, Star rating, Rating text.
* **Librarian Functions (Assumed based on CRUD API):**
  + Shall be able to add reviews (potentially administrative). (API: POST /api/review/add)
  + Shall be able to view all reviews. (API: GET /api/review/getAll)
  + Shall be able to view a specific review. (API: GET /api/review/get/:id)
  + Shall be able to update reviews. (API: PUT /api/review/update/:id)
  + Shall be able to delete reviews. (API: DELETE /api/review/delete/:id)
* **Member Functions (Assumed capabilities if UI were present):**
  + Could potentially add reviews for books.
  + Could potentially view reviews for books.

## 4. External Interface Requirements

### 4.1. User Interfaces

* The system shall provide a web-based graphical user interface (GUI) accessible via modern web browsers.
* The UI shall be responsive and adapt to different screen sizes (desktop, tablet, mobile).
* The UI shall be intuitive and easy to navigate.
* Key UI pages include:
  + Login Page
  + Dashboard (Librarian)
  + Book List and Detail Page
  + Author List and Detail Page
  + Genre List and Detail Page
  + Borrowal List and Detail Page
  + User List and Detail Page (Librarian)
  + Forms for creating/editing entities.
  + Dialogs for confirmations (e.g., delete operations).

### 4.2. Software Interfaces

* **Database:** The system shall interface with a MongoDB database for data storage and retrieval.
* **Node.js Environment:** The backend server runs in a Node.js environment.
* **Web Browser:** The client application runs in a standard web browser.
* **API:** The client communicates with the server via a RESTful API over HTTP/HTTPS.

## 5. Non-Functional Requirements

### 5.1. Security Requirements

* User passwords shall be stored securely using hashing and salting (pbkdf2Sync with SHA512).
* The system shall use sessions to manage user login state.
* Access to system functionalities shall be restricted based on user roles (Librarian, Member).
* Sensitive data in transit (e.g., during login) should be protected using HTTPS (though not explicitly enforced by the current server setup, it's a standard requirement).
* Input validation should be performed on both client and server sides to prevent common vulnerabilities (e.g., XSS, injection attacks).

### 5.2. Usability Requirements

* The user interface shall be user-friendly and intuitive, requiring minimal training for users.
* The system shall provide clear feedback to users for their actions (e.g., success/error messages via toasts).
* Navigation shall be consistent across the application.
* Forms shall be easy to understand and fill out.
* The system should load pages and data within a reasonable timeframe.

### 5.3. Maintainability Requirements

* The codebase shall be well-structured and modular (as evidenced by the client/server separation and component-based UI).
* Code shall be commented where necessary to explain complex logic.
* The use of Docker facilitates easier deployment and environment management.

## 6. Detailed Data Model / Data Dictionary

This section provides a more detailed look at the data entities used within the Library Management System. The data types are based on MongoDB/Mongoose schema definitions.

### 6.1. User Entity (UserSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the user. |
| name | String | Required | Full name of the user. |
| email | String | Required, Unique (implied by findOne logic) | Email address of the user, used for login. |
| dob | Date | Optional | Date of birth of the user. |
| phone | String | Optional | Phone number of the user. |
| isAdmin | Boolean | Required | Flag indicating if the user is an admin/librarian. |
| photoUrl | String | Required (default value provided in client) | URL to the user's profile photo. |
| hash | String | System-generated | Hashed password. |
| salt | String | System-generated | Salt used for password hashing. |

### 6.2. Book Entity (bookSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the book. |
| name | String | Required | Title of the book. |
| isbn | String | Required | International Standard Book Number. |
| authorId | ObjectId | Optional, Refers to 'Author' | Foreign key referencing the Author entity. |
| genreId | ObjectId | Optional, Refers to 'Genre' | Foreign key referencing the Genre entity. |
| isAvailable | Boolean | Required | Availability status of the book. |
| summary | String | Optional | A brief summary or description of the book. |
| photoUrl | String | Optional | URL to the book's cover image. |

### 6.3. Author Entity (authorSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the author. |
| name | String | Required | Name of the author. |
| description | String | Optional | Brief biography or description of the author. |
| photoUrl | String | Optional | URL to the author's photo. |

### 6.4. Genre Entity (genreSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the genre. |
| name | String | Required | Name of the genre (e.g., Fiction, Sci-Fi). |
| description | String | Required | Description of the genre. |

### 6.5. Borrowal Entity (borrowalSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the borrowal record. |
| bookId | ObjectId | Required, Refers to 'Book' | Foreign key referencing the Book borrowed. |
| memberId | ObjectId | Required, Refers to 'User' | Foreign key referencing the User (member) who borrowed. |
| borrowedDate | Date | Optional (defaults if not provided) | Date when the book was borrowed. |
| dueDate | Date | Optional | Date when the book is due to be returned. |
| status | String | Optional | Current status of the borrowal (e.g., "Borrowed", "Returned", "Overdue"). |

### 6.6. Review Entity (reviewSchema)

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Data Type** | **Constraints** | **Description** |
| \_id | ObjectId | Primary Key, Auto-generated, Unique, Required | Unique identifier for the review. |
| bookID | ObjectId | Required, Refers to 'Book' | Foreign key referencing the Book being reviewed. |
| star | String | Required | Star rating given for the book (e.g., "1" to "5"). Consider changing to Number. |
| rating | String | Optional | Textual content of the review. |

### 6.7. Entity Relationship Overview (Conceptual)

* A **User** (where isAdmin=false, i.e., a Member) can have multiple **Borrowal** records.
* A **Book** can be associated with one **Author** (current model allows one, could be many-to-many in a more complex system).
* A **Book** can be associated with one **Genre** (current model allows one, could be many-to-many).
* A **Book** can have multiple **Borrowal** records over time.
* A **Book** can have multiple **Review** records.
* A **Borrowal** record links one **Book** to one **User** (Member).
* A **Review** record links to one **Book**.

## 7. Key Use Cases

This section outlines some of the key use cases for the Library Management System.

### 7.1. UC-001: User Login

* **Actor(s):** Librarian, Member (Any registered User)
* **Description:** Allows a registered user to gain access to the system based on their role.
* **Preconditions:**
  + The user must have an existing account in the system.
  + The system is accessible.
* **Postconditions:**
  + If authentication is successful, the user is logged in and a session is established.
  + The user is redirected to their respective dashboard/landing page (Librarian to /dashboard, Member to /books).
  + If authentication fails, an error message is displayed, and the user remains on the login page.
* **Main Flow:**
  1. User navigates to the Login Page (/login).
  2. User enters their email address.
  3. User enters their password.
  4. User clicks the "Login" button.
  5. System validates the input fields (e.g., not empty).
  6. System authenticates the user's credentials against the User collection in the database (verifies email and password hash).
  7. If authentication is successful:  
     a. System establishes a user session.  
     b. System redirects the user to their role-appropriate landing page.
* **Alternative Flows:**
  + **A1: Invalid Credentials:**
    1. If authentication fails (email not found or password incorrect), the system displays an error message (e.g., "User not found" or "Password incorrect").
    2. User remains on the login page.
  + **A2: Empty Fields:**
    1. If email or password fields are empty upon submission, the system displays an error message (e.g., "Please enter email and password").
    2. User remains on the login page.

### 7.2. UC-002: Add New Book (Librarian)

* **Actor(s):** Librarian
* **Description:** Allows a Librarian to add a new book to the library's inventory.
* **Preconditions:**
  + The Librarian is logged into the system.
  + The Librarian has navigated to the Book Management page (/books).
  + Author and Genre data should ideally exist if they are to be linked, or the form should allow creating them/selecting from existing.
* **Postconditions:**
  + If successful, a new book record is created in the database and appears in the book list.
  + A success message is displayed.
  + If unsuccessful, an error message is displayed, and the book is not added.
* **Main Flow:**
  1. Librarian clicks the "New Book" button on the Book Management page.
  2. System displays the "Add Book" form/modal.
  3. Librarian enters book details: Name, ISBN, Summary, Photo URL.
  4. Librarian selects an Author from a list of existing authors.
  5. Librarian selects a Genre from a list of existing genres.
  6. Librarian sets the "Is Available" status (e.g., checkbox, default true).
  7. Librarian submits the form.
  8. System validates the input data (e.g., required fields).
  9. System sends a request to the backend API (POST /api/book/add) with the book data.
  10. Backend creates a new book record in the database.
  11. System displays a success message (e.g., "Book added").
  12. The "Add Book" form/modal is closed.
  13. The book list on the Book Management page is updated to show the new book.
* **Alternative Flows:**
  + **A1: Validation Error:**
    1. If any required fields are missing or data is invalid, the system displays an error message next to the respective fields or a general form error.
    2. The form remains open for correction.
  + **A2: API Error:**
    1. If the backend API returns an error (e.g., database error, unexpected issue), the system displays a generic error message (e.g., "Something went wrong, please try again").
    2. The form may remain open or close depending on the error severity.

### 7.3. UC-003: Borrow a Book (Member)

* **Actor(s):** Member
* **Description:** Allows a logged-in Member to request to borrow an available book.
* **Preconditions:**
  + The Member is logged into the system.
  + The Member has navigated to the Borrowal Management page (/borrowals).
  + The book to be borrowed must exist and be marked as available.
* **Postconditions:**
  + If successful, a new borrowal record is created with a status (e.g., "Borrowed" or "Pending Approval" if applicable).
  + The availability status of the borrowed book might be updated (e.g., isAvailable to false).
  + A success message is displayed.
  + If unsuccessful, an error message is displayed.
* **Main Flow:**
  1. Member clicks the "New Borrowal" button on the Borrowal Management page.
  2. System displays the "Add Borrowal" form/modal.
  3. Member selects an available Book from a list (dropdown/search).
  4. The Member field is auto-populated or selected (if a librarian is doing it on behalf of a member, but the use case title implies member action). The client logic ensures memberId is the current logged-in member.
  5. System auto-populates Borrowed Date (current date) and calculates a Due Date (e.g., current date + 14 days). Status is set (e.g., "Borrowed").
  6. Member submits the form.
  7. System validates the input.
  8. System sends a request to the backend API (POST /api/borrowal/add) with borrowal data.
  9. Backend creates a new borrowal record.
  10. Backend potentially updates the isAvailable status of the Book entity.
  11. System displays a success message (e.g., "Borrowal added").
  12. The "Add Borrowal" form/modal is closed.
  13. The member's borrowal list is updated.
* **Alternative Flows:**
  + **A1: Book Not Available:**
    1. If the selected book is not available, the system prevents submission or displays an error.
  + **A2: API Error:**
    1. If the backend API returns an error, the system displays an error message.

### 7.4. UC-004: Register New User (Librarian)

* **Actor(s):** Librarian
* **Description:** Allows a Librarian to create a new user account (either another Librarian or a Member).
* **Preconditions:**
  + The Librarian is logged into the system.
  + The Librarian has navigated to the User Management page (/users).
* **Postconditions:**
  + If successful, a new user record is created in the database.
  + A success message is displayed.
  + If unsuccessful (e.g., email already exists), an error message is displayed.
* **Main Flow:**
  1. Librarian clicks the "New User" button on the User Management page.
  2. System displays the "Add User" form/modal.
  3. Librarian enters user details: Name, Email, Password, Date of Birth (optional), Phone (optional), Photo URL (optional/default).
  4. Librarian selects the user's role (Is Admin: Yes/No).
  5. Librarian submits the form.
  6. System validates the input data (e.g., required fields, email format).
  7. System sends a request to the backend API (POST /api/user/add or POST /api/auth/register) with user data.
  8. Backend checks if the email already exists. If not, hashes the password and creates a new user record.
  9. System displays a success message (e.g., "User added" or "User registered").
  10. The "Add User" form/modal is closed.
  11. The user list on the User Management page is updated.
* **Alternative Flows:**
  + **A1: Email Already Exists:**
    1. If the backend indicates the email is already in use, the system displays an error message (e.g., "User already exists").
    2. The form remains open for correction.
  + **A2: Validation Error:**
    1. If any required fields are missing or data is invalid, the system displays an error message.
    2. The form remains open for correction.
  + **A3: API Error:**
    1. If the backend API returns an unexpected error, the system displays a generic error message.

### 7.5. UC-005: View Borrowal History (Member)

* **Actor(s):** Member
* **Description:** Allows a logged-in Member to view their own history of borrowed books.
* **Preconditions:**
  + The Member is logged into the system.
* **Postconditions:**
  + The Member's borrowal records are displayed.
* **Main Flow:**
  1. Member navigates to the Borrowal Management page (/borrowals).
  2. System sends a request to the backend API (GET /api/borrowal/getAll) to fetch all borrowal records.
  3. The client-side application filters the received list to display only the borrowals where memberId matches the currently logged-in Member's ID.
  4. The system displays a table or list of the Member's borrowals, including Book Name, Borrowed Date, Due Date, and Status.
* **Alternative Flows:**
  + **A1: No Borrowal History:**
    1. If the Member has no borrowal records, the system displays a message indicating this (e.g., "No borrowals found").
  + **A2: API Error:**
    1. If there's an error fetching data from the API, an error message is displayed.