# PROYECTO PRIMER 50

# Arquitectura del sistema y documentación

Hovar Steven Rincón Vianchá
Andrés Ivan Sierra Espinel

Uptc
Universidad Pedagógica y Tecnológica de Colombia

ACREDITACIÓN INSTITUCIONAL
DE ALTA CALIDAD
MULTICAMPUS
Resolución 3910 de 2015 MEN / 6 años

SIC

# Tabla de contenidos

# I. Análisis del sistema

## a. Contextualización de la problemática

Una biblioteca municipal necesita un software que ayude a la gestión de sus libros, empleados y servicios que se prestan, para cumplir con el objetivo principal se debe tener en cuenta el siguiente flujo:

La biblioteca esta administrada por una persona que se encarga de gestionar (CRUD), sus empleados con la siguiente información: nombres, apellidos teléfono, usuario y contraseña, cabe aclarar que los empleados deben ser mayores de edad; los libros con sus respectivos datos: ISBN, nombre, autor, genero, número de copias y fecha de publicación, multa diaria.

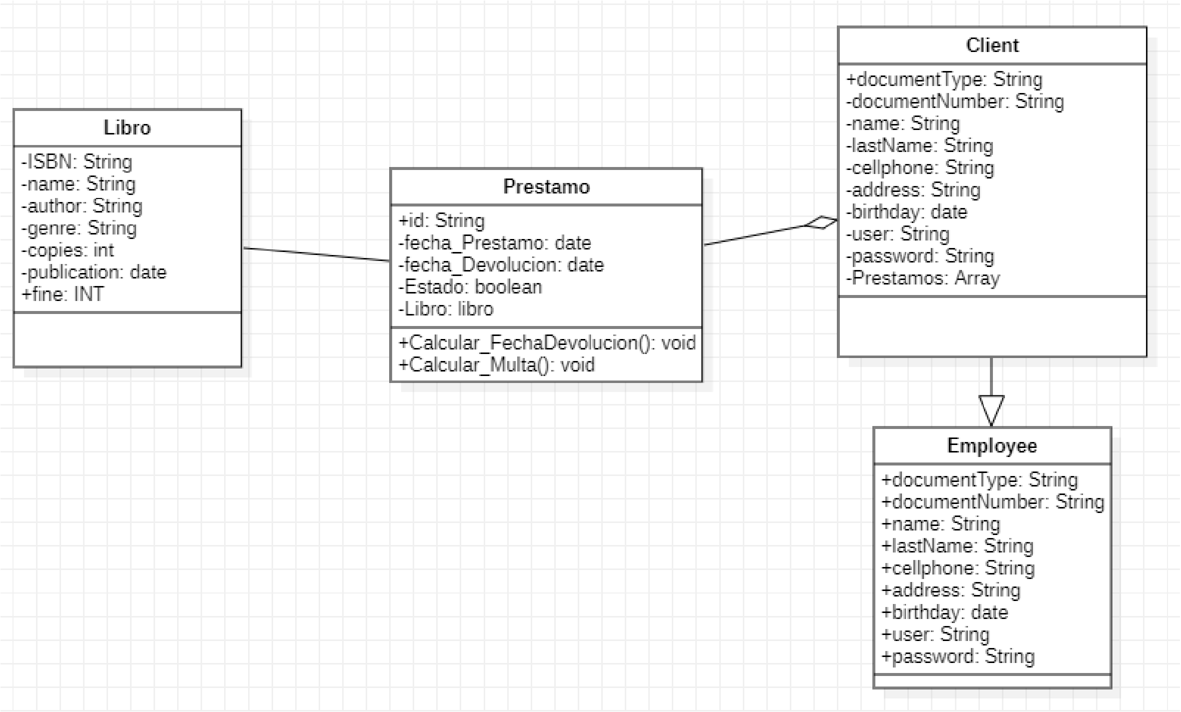
Los empleados podrán iniciar sesión en el sistema, debido a que serán los encargados de registrar los usuarios que se acerquen a la biblioteca con la siguiente información: nombre, apellido, teléfono, dirección, ciudad, departamento, usuario y contraseña. El préstamo de los libros se realizará solo a usuarios registrados, se solicitará: la fecha de préstamo, la fecha de regreso (la fecha de préstamo más 8 días). Cabe aclarar que al prestar un libro se deberá verificar que existan copias en la biblioteca. Por otro lado, el empelado también podrá recibir los libros y cambiar el estado del préstamo a terminado.

Por último, los usuarios podrán iniciar sesión en el sistema y visualizar sus préstamos para verificar su historial de libros que se han solicitado y los préstamos que se tengan activos.

## b. Diagramas UML

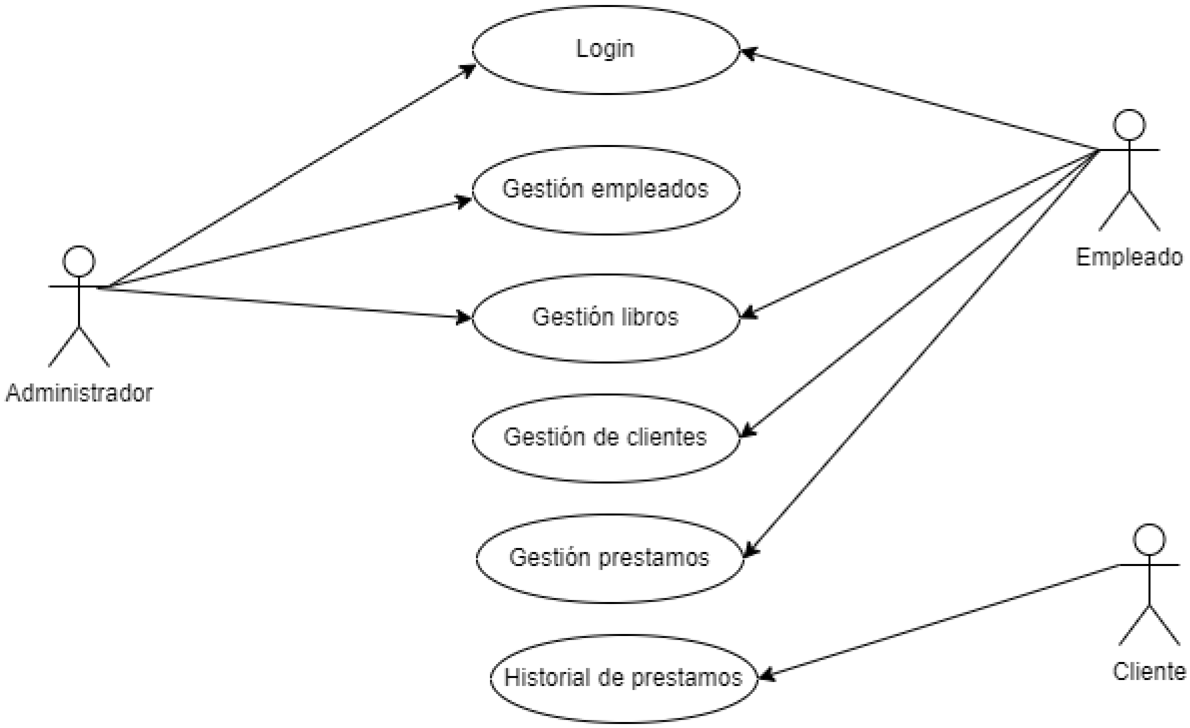
- ### Diagrama de clases

En el siguiente diagrama se pueden observar las clases necesarias para poder implementar el sistema propuesto.

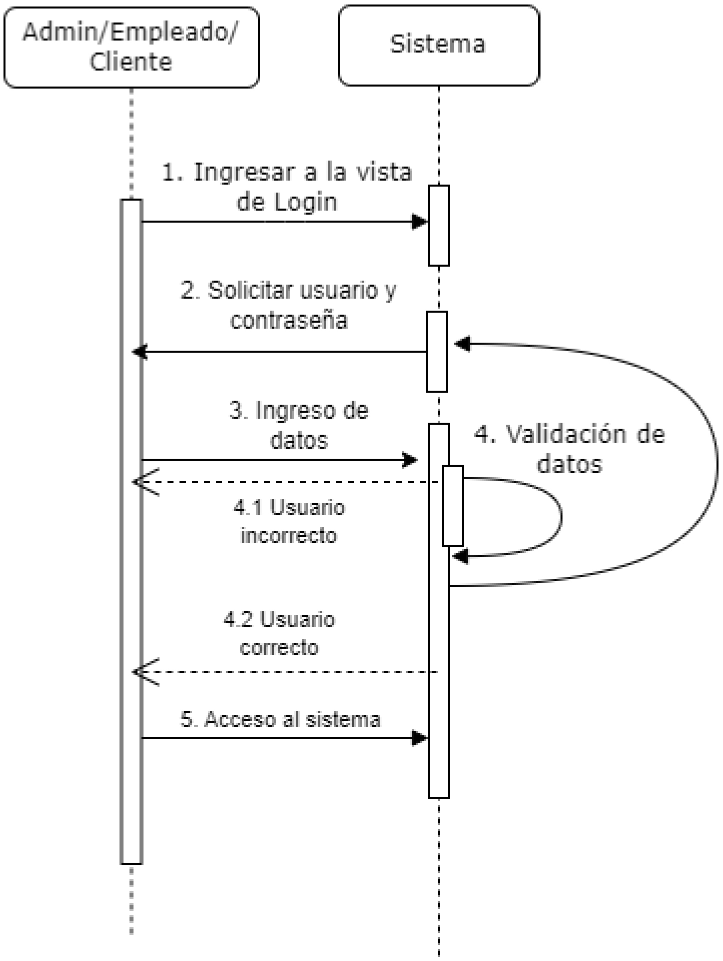


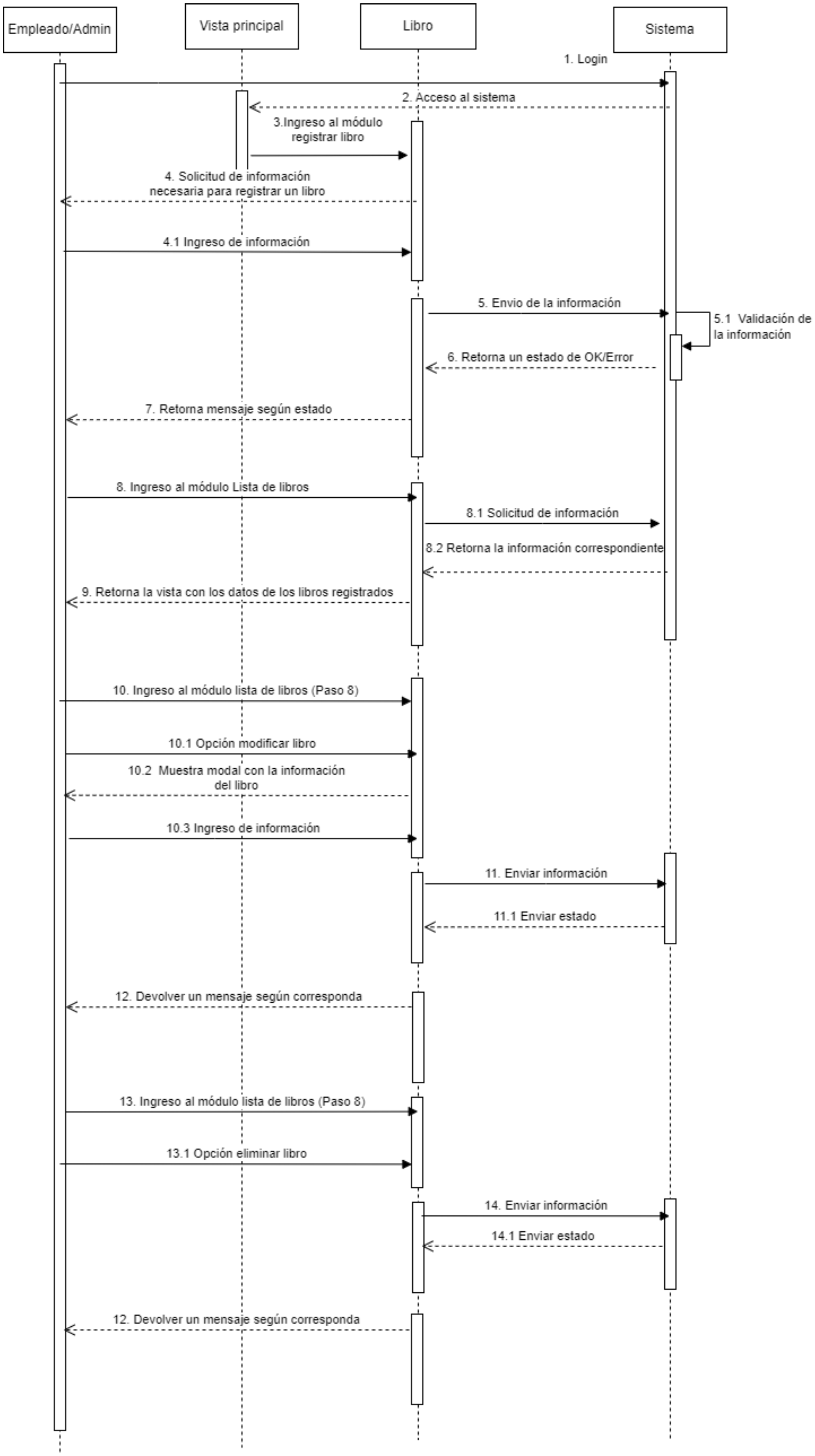
- ### Diagrama de casos de uso

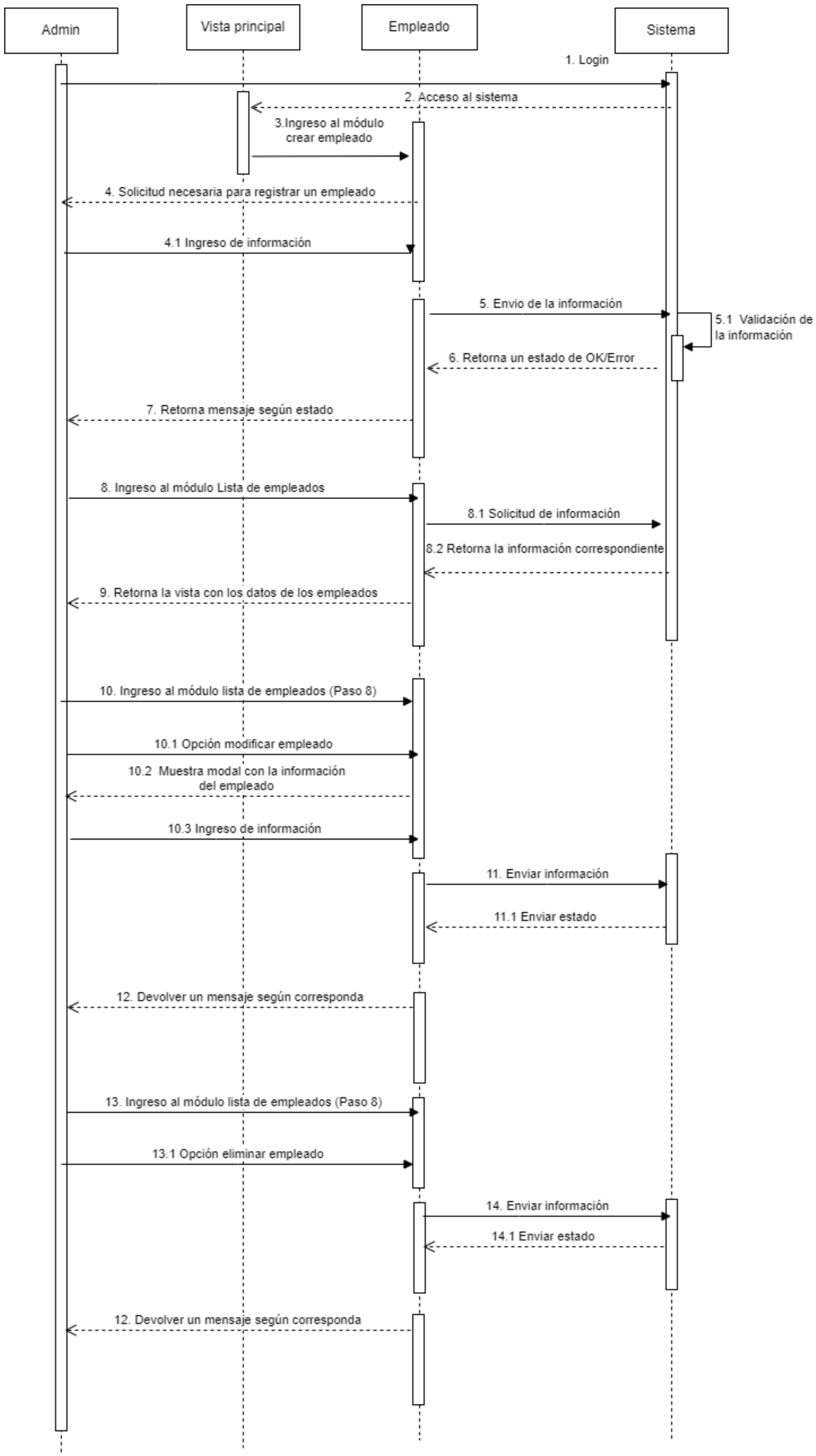
De acuerdo al diagrama de casos de uso que se presenta, se ha llevado a cabo un proceso de análisis con el fin de determinar y entender las principales acciones o funciones que el sistema debe realizar para satisfacer las necesidades de los diferentes usuarios involucrados.

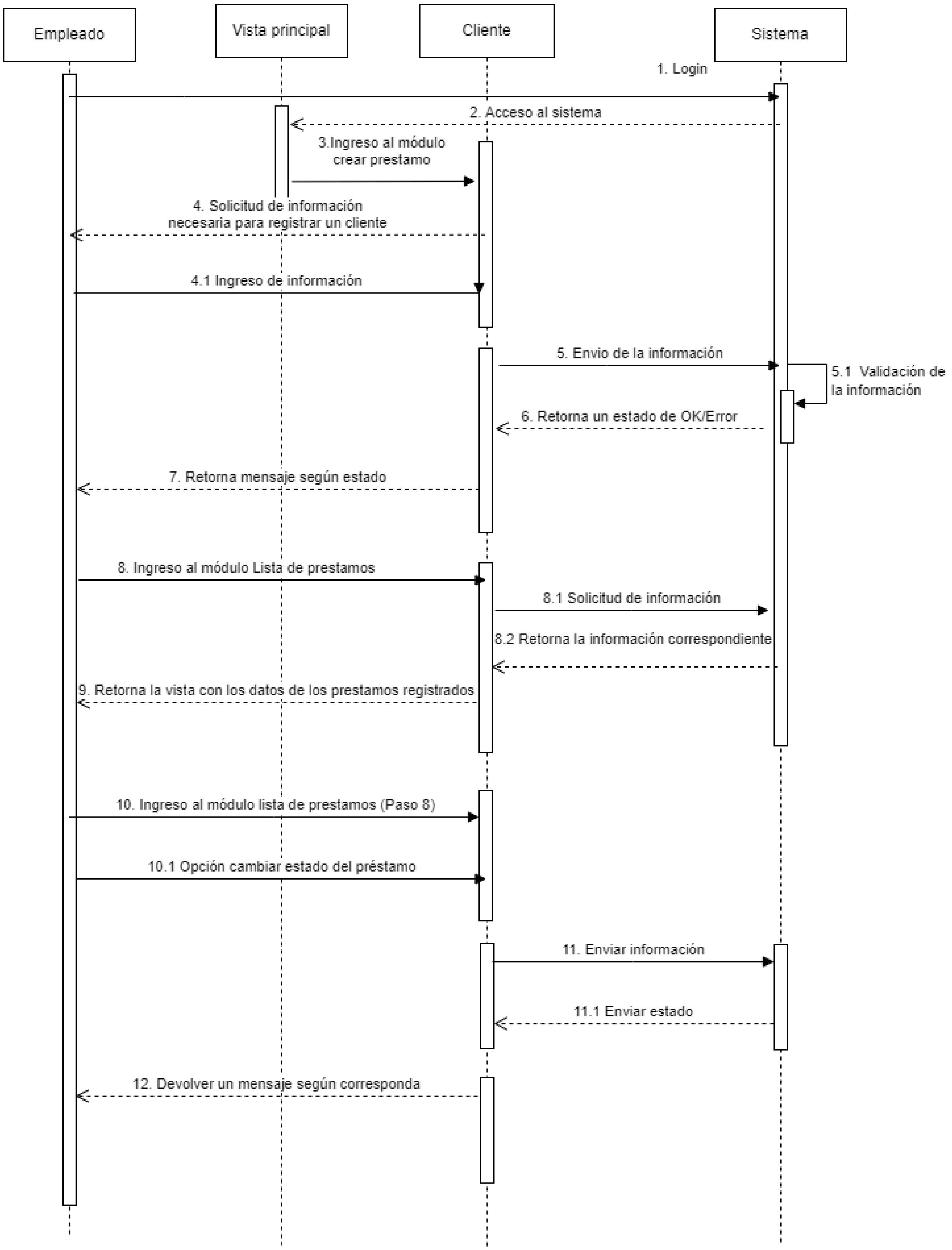

Login
Gestión empleados
Gestión libros
Gestión de clientes
Gestión prestamos
Historial de prestamos
Administrador
Empleado
Cliente

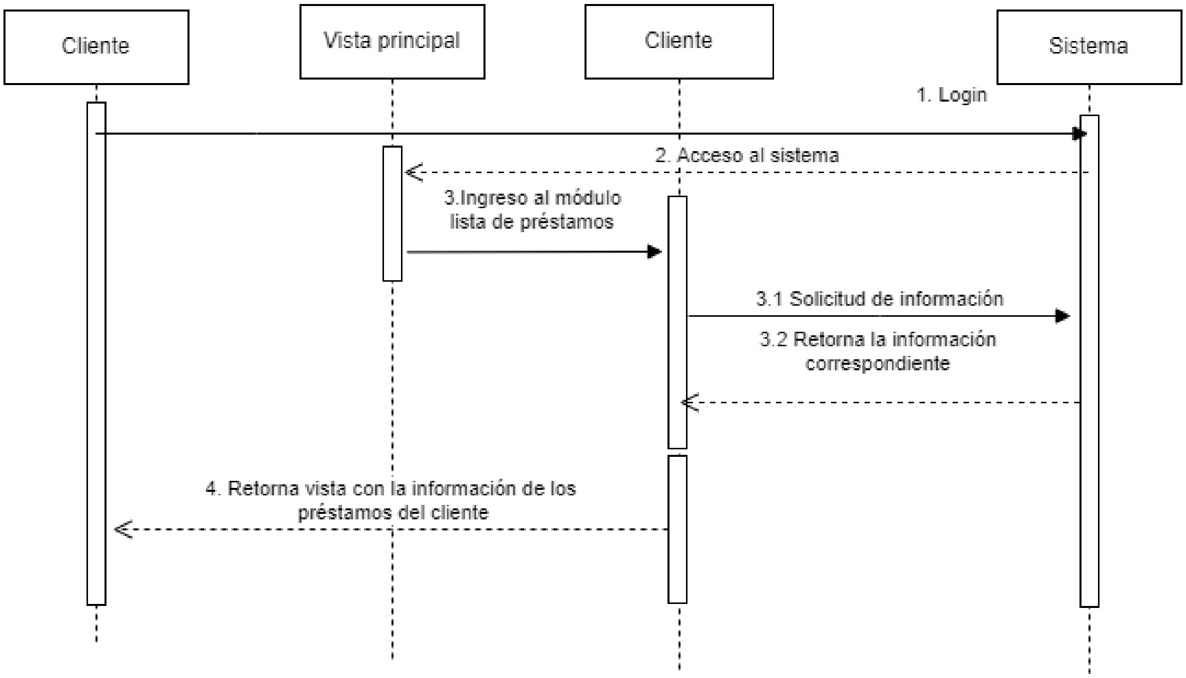
- Vista de secuencia



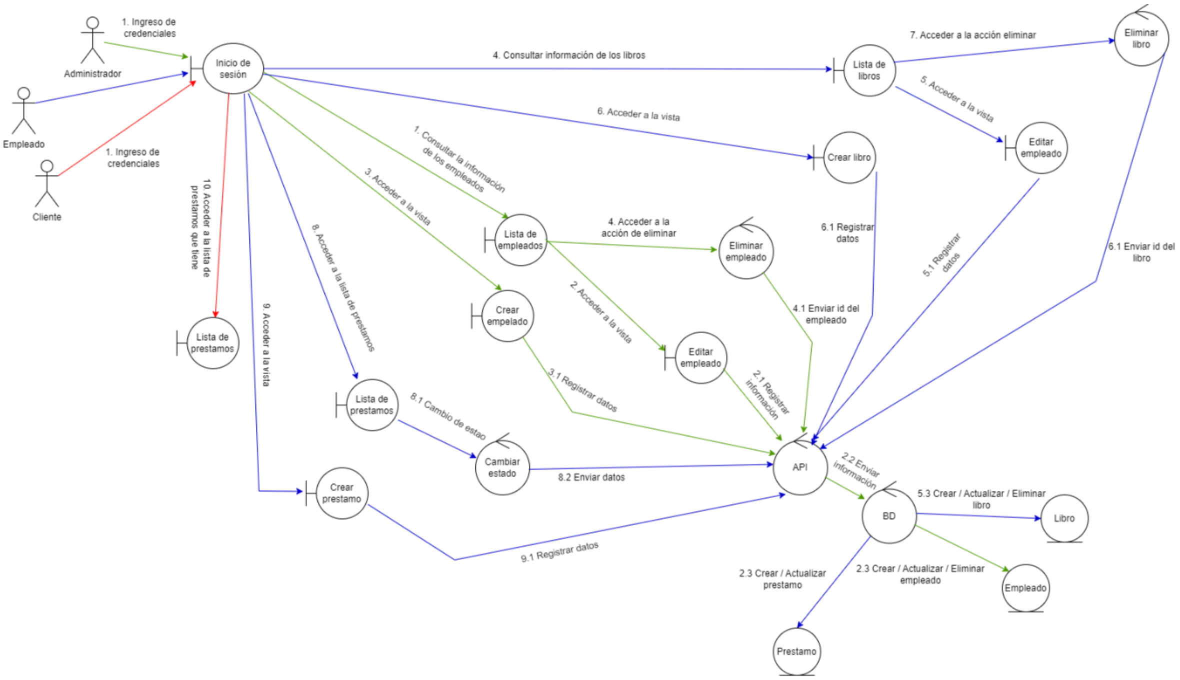

Empleado/Admin
Vista principal
Libro
Sistema
1. Login
2. Acceso al sistema
3.Ingreso al módulo registrar libro
4. Solicitud de información necesaria para registrar un libro
4.1 Ingreso de información
5. Envio de la información
5.1 Validación de la información
6. Retorna un estado de OK/Error
7. Retorna mensaje según estado
8. Ingreso al módulo Lista de libros
8.1 Solicitud de información
8.2 Retorna la información correspondiente
9. Retorna la vista con los datos de los libros registrados
10. Ingreso al módulo lista de libros (Paso 8)
10.1 Opción modificar libro
10.2 Muestra modal con la información del libro
10.3 Ingreso de información
11. Enviar información
11.1 Enviar estado
12. Devolver un mensaje según corresponda
13. Ingreso al módulo lista de libros (Paso 8)
13.1 Opción eliminar libro
14. Enviar información
14.1 Enviar estado
12. Devolver un mensaje según corresponda


Admin
Vista principal
Empleado
Sistema
1. Login
2. Acceso al sistema
3.Ingreso al módulo crear empleado
4. Solicitud necesaria para registrar un empleado
4.1 Ingreso de información
5. Envio de la información
5.1 Validación de la información
6. Retorna un estado de OK/Error
7. Retorna mensaje según estado
8. Ingreso al módulo Lista de empleados
8.1 Solicitud de información
8.2 Retorna la información correspondiente
9. Retorna la vista con los datos de los empleados
10. Ingreso al módulo lista de empleados (Paso 8)
10.1 Opción modificar empleado
10.2 Muestra modal con la información del empleado
10.3 Ingreso de información
11. Enviar información
11.1 Enviar estado
12. Devolver un mensaje según corresponda
13. Ingreso al módulo lista de empleados (Paso 8)
13.1 Opción eliminar empleado
14. Enviar información
14.1 Enviar estado
12. Devolver un mensaje según corresponda

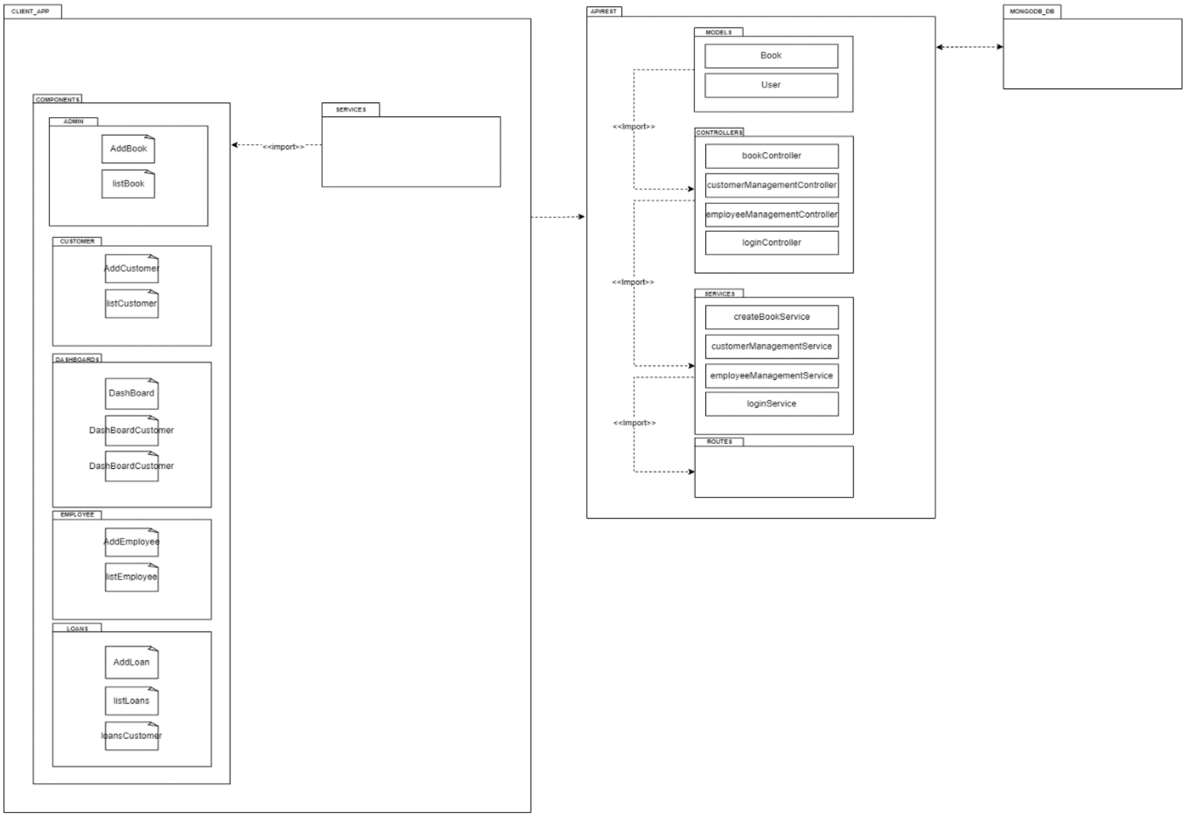

Empleado
Vista principal
Cliente
Sistema
1. Login
2. Acceso al sistema
3.Ingreso al módulo
crear prestamo
4. Solicitud de información
necesaria para registrar un cliente
4.1 Ingreso de información
5. Envio de la información
5.1  Validación de
la información
6. Retorna un estado de OK/Error
7. Retorna mensaje según estado
8. Ingreso al módulo Lista de prestamos
8.1 Solicitud de información
8.2 Retorna la información correspondiente
9. Retorna la vista con los datos de los prestamos registrados
10. Ingreso al módulo lista de prestamos (Paso 8)
10.1 Opción cambiar estado del préstamo
11. Enviar información
11.1 Enviar estado
12. Devolver un mensaje según corresponda


Cliente
Vista principal
Cliente
Sistema
1. Login
2. Acceso al sistema
3.Ingreso al módulo lista de préstamos
3.1 Solicitud de información
3.2 Retorna la información correspondiente
4. Retorna vista con la información de los préstamos del cliente

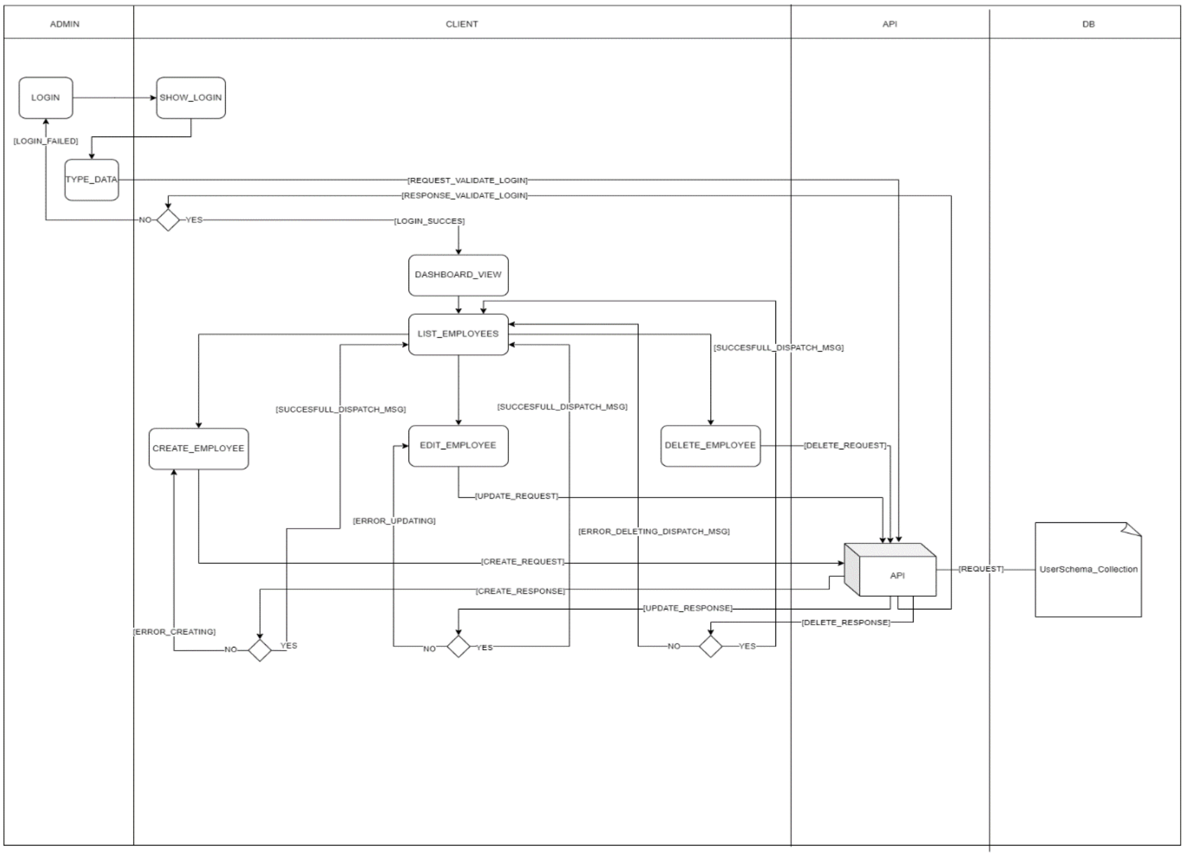
- Vista de comunicación

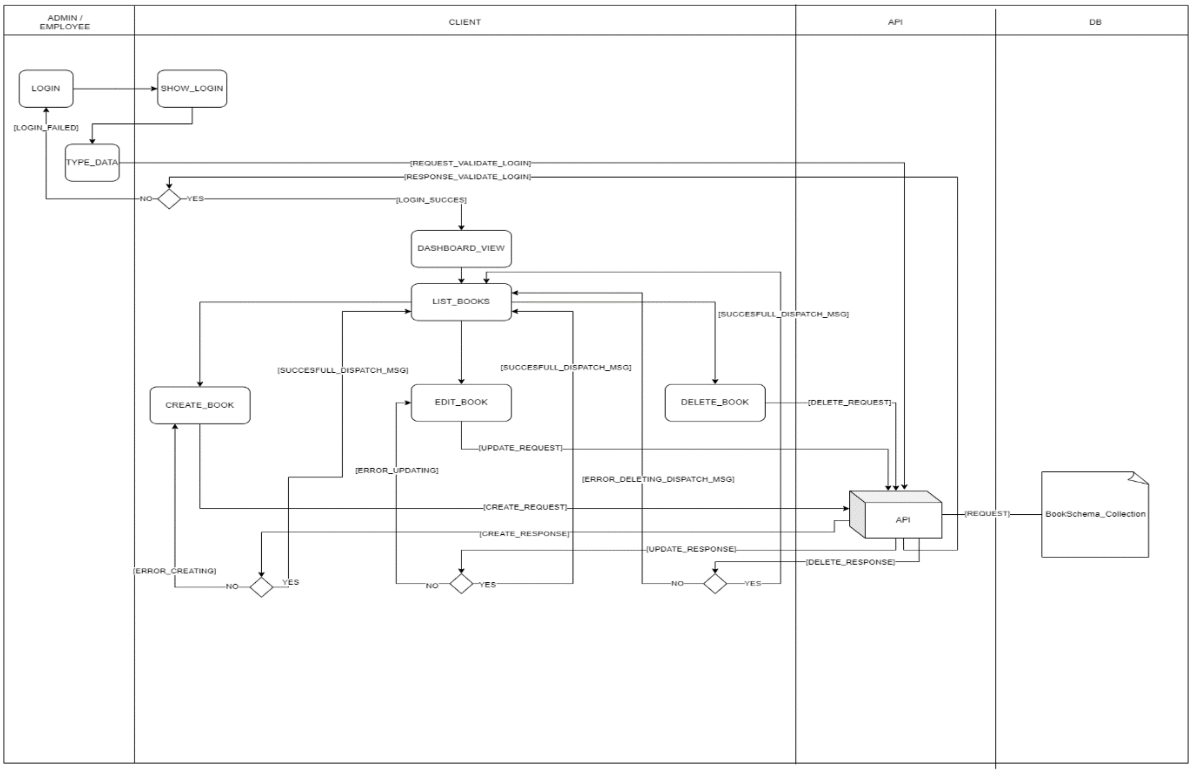


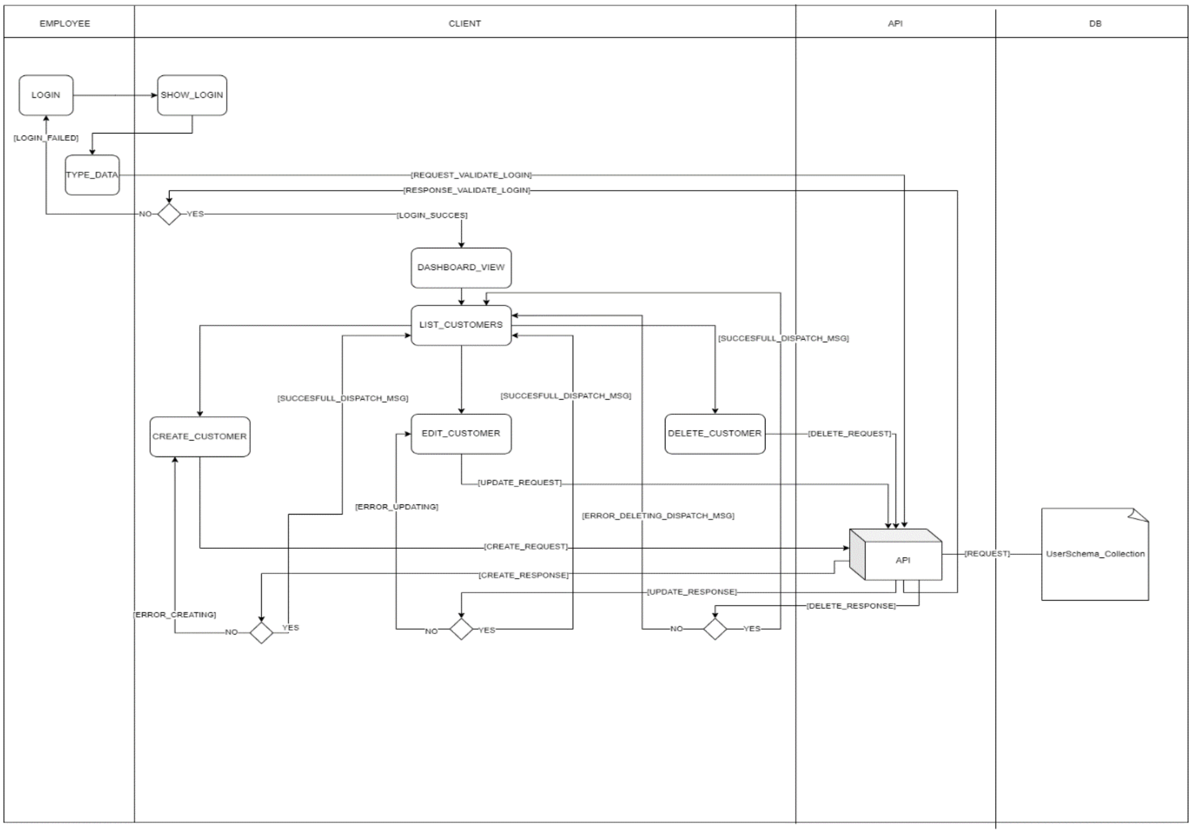
- Vista de despliegue

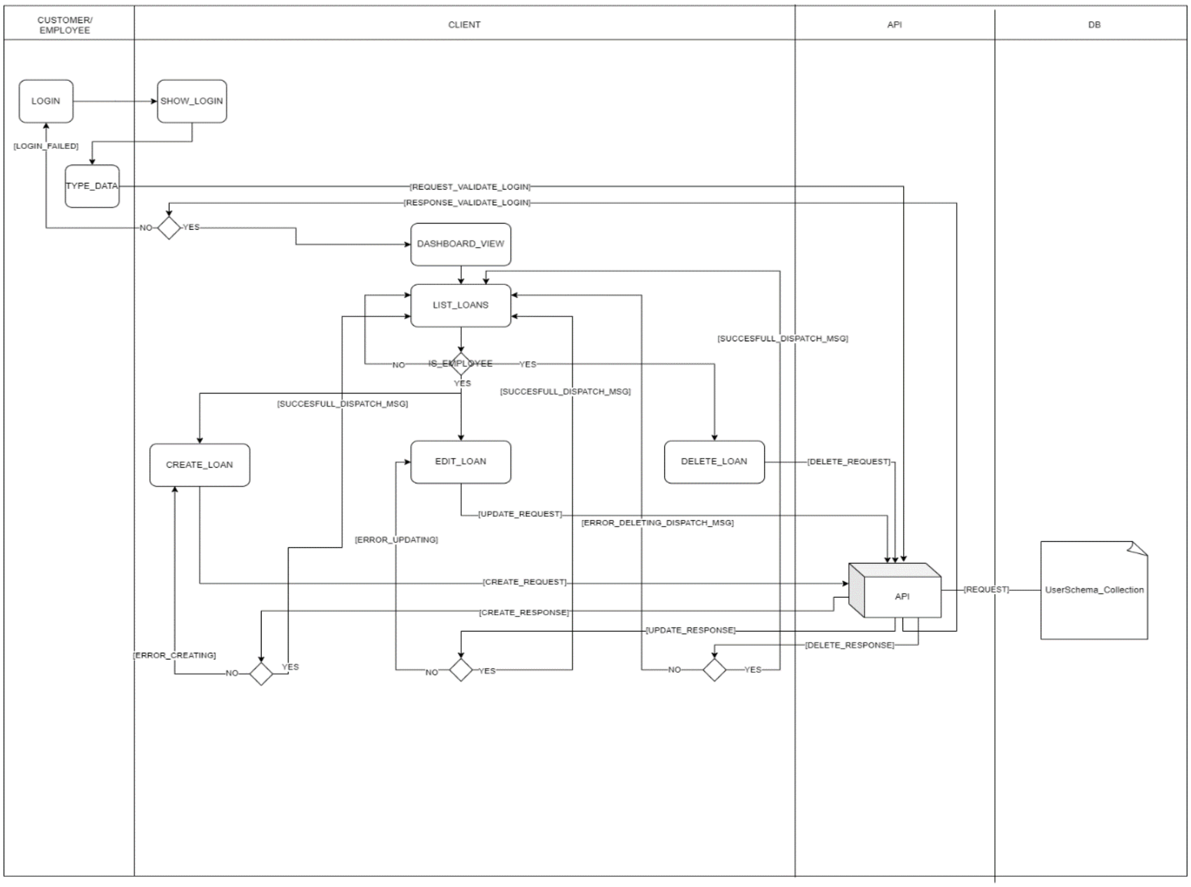


- Vista de Procesos

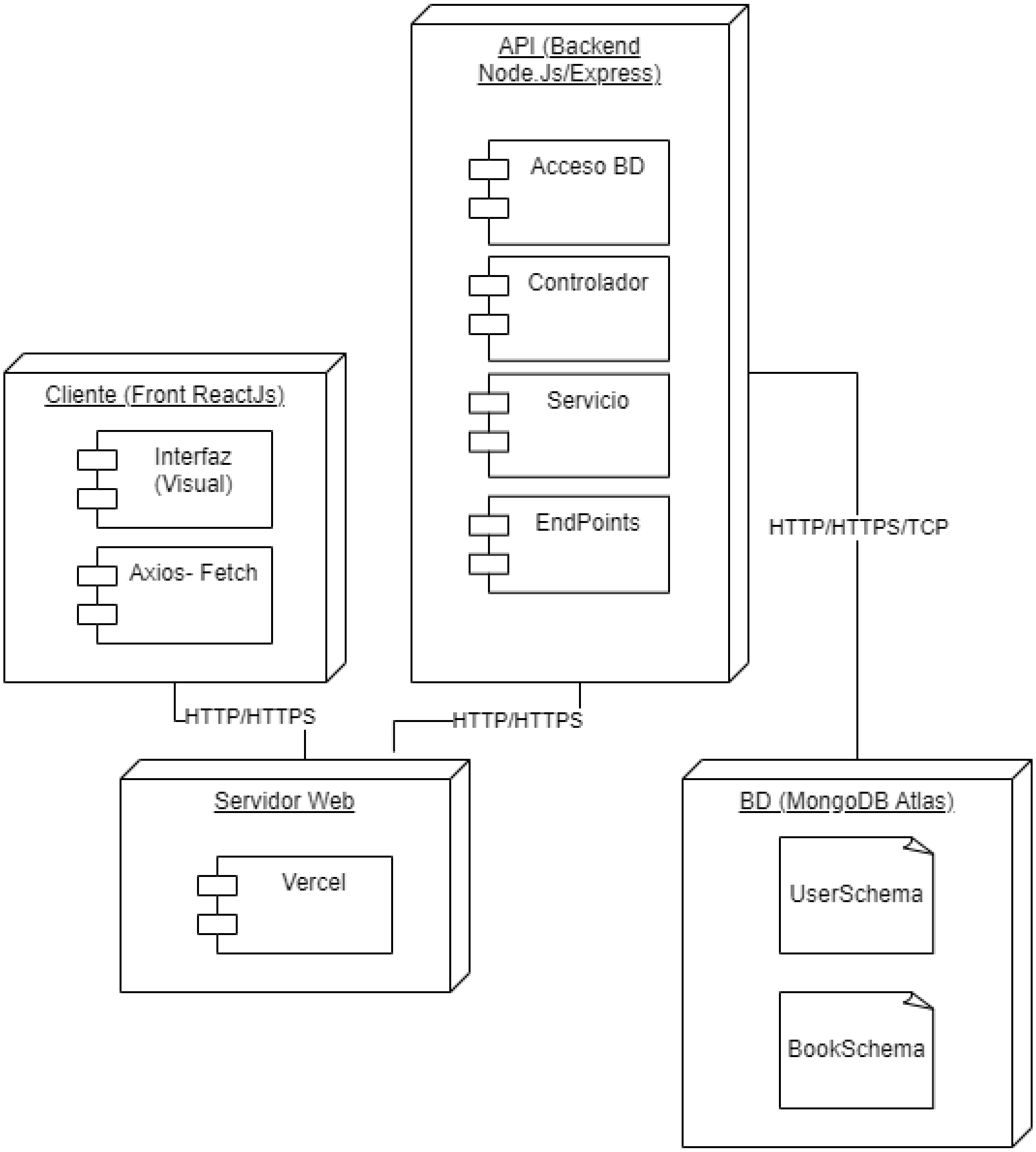





EMPLOYEE
CLIENT
API
DB
LOGIN
SHOW_LOGIN
[LOGIN_FAILED]
TYPE_DATA
[REQUEST_VALIDATE_LOGIN]
[RESPONSE_VALIDATE_LOGIN]
NO
YES
[LOGIN_SUCCES]
DASHBOARD_VIEW
LIST_CUSTOMERS
[SUCCESFULL_DISPATCH_MSG]
[SUCCESFULL_DISPATCH_MSG]
[SUCCESFULL_DISPATCH_MSG]
CREATE_CUSTOMER
EDIT_CUSTOMER
DELETE_CUSTOMER
[DELETE_REQUEST]
[UPDATE_REQUEST]
[ERROR_UPDATING]
[ERROR_DELETING_DISPATCH_MSG]
[CREATE_REQUEST]
API
[REQUEST]
UserSchema_Collection
[CREATE_RESPONSE]
[UPDATE_RESPONSE]
[DELETE_RESPONSE]
[ERROR_CREATING]
NO
YES
NO
YES
NO
YES


CUSTOMER/
EMPLOYEE
CLIENT
API
DB
LOGIN
SHOW_LOGIN
[LOGIN_FAILED]
TYPE_DATA
[REQUEST_VALIDATE_LOGIN]
[RESPONSE_VALIDATE_LOGIN]
NO
YES
DASHBOARD_VIEW
LIST_LOANS
[SUCCESFULL_DISPATCH_MSG]
NO
IS_EMPLOYEE
YES
YES
[SUCCESFULL_DISPATCH_MSG]
[SUCCESFULL_DISPATCH_MSG]
CREATE_LOAN
EDIT_LOAN
DELETE_LOAN
[DELETE_REQUEST]
[UPDATE_REQUEST]
[ERROR_DELETING_DISPATCH_MSG]
[ERROR_UPDATING]
[CREATE_REQUEST]
API
[REQUEST]
UserSchema_Collection
[CREATE_RESPONSE]
[UPDATE_RESPONSE]
[DELETE_RESPONSE]
[ERROR_CREATING]
NO
YES
NO
YES
NO
YES

- Vista física



c. Requisitos funcionales

| Requisito: RF-01 Persistencia de datos | |
|---|---|
| **Descripción** | El sistema debe persistir los datos con el fin de poder acceder a los mismos en cualquier momento que se ejecute la aplicación |
| **Prioridad** | Alta |
| **Alcance** | La implementación de este requisito asegura que el sistema se desempeñe de la mejor manera. |

| **Requisito: RF-01 Inicio de sesión** | |
|---|---|
| **Descripción** | El sistema debe permitir diligenciar los datos necesarios para el acceso de los usuarios al sistema, sabiendo que, existen clientes y empleados. Los datos necesarios para el acceso de los usuarios son:<br>- Usuario (Alfanumérico)<br>- Contraseña (Alfanumérico) |
| **Prioridad** | Alta |
| **Alcance** | Este apartado es necesario para el acceso de los usuarios a sus respectivos módulos, esto con el fin de que cada uno realice sus actividades correspondientes. |

| **Requisito: RF-02 Gestión de empleados** | |
|---|---|
| **Descripción** | El sistema debe permitir realizar la gestión de los empleados. Cada uno de estos puede estar sujeto a cambios tales como, creación, visualización, actualización y eliminación. Los datos necesarios para la correcta gestión de los empleados son:<br>- Nombre<br>- Apellido<br>- Fecha nacimiento<br>- Teléfono<br>- Usuario<br>- Contraseña |
| **Prioridad** | Alta |
| **Alcance** | Este apartado es muy importante, ya que estos son los encargados de realizar la gestión de los libros dentro de la biblioteca. |

| **Requisito: RF-03 Gestión de libros** | |
|---|---|
| **Descripción** | El sistema debe permitir realizar la gestión de libros. Cada uno de estos puede estar sujeto a cambios tales como, creación, visualización, actualización y eliminación. Los datos necesarios para la correcta gestión de los libros son:<br>- ISBN<br>- Nombre<br>- Autor<br>- Genero<br>- Número de copias |

| | |
|---|---|
| | - Fecha publicación<br>- Valor multa diaria |
| **Prioridad** | Alta |
| **Alcance** | Este apartado es muy importante, ya que es la parte central de la biblioteca. |

| **Requisito: RF-04 Gestión de usuarios** | |
|---|---|
| **Descripción** | El sistema debe permitir realizar la gestión de usuarios, estos solo podrán ser registrados por alguno de los empleados que se encuentran dentro del sistema. Cada uno de estos puede estar sujeto a cambios tales como, creación, visualización, actualización y eliminación. Los datos necesarios para la correcta gestión de los usuarios son:<br>- Nombre<br>- Apellido<br>- Teléfono<br>- Dirección<br>- Usuario<br>- Contraseña |
| **Prioridad** | Alta |
| **Alcance** | Serán los encargados de realizar la solicitud de préstamos. |

| **Requisito: RF-05 Gestión de préstamos** | |
|---|---|
| **Descripción** | El sistema debe permitir realizar la gestión de préstamos. Cada uno de estos puede estar sujeto a cambios tales como, creación, visualización y actualización. Los datos necesarios para la correcta gestión de los préstamos son:<br>- Fecha del préstamo<br>- Fecha de regreso<br>- Estado |
| **Prioridad** | Alta |
| **Alcance** | Es una parte importante del sistema, ya que, se llevará el histórico de todos los prestamos que se han hecho dentro de la biblioteca. |

| **Requisito: RF-06 Visualización de préstamos de cada usuario** | |
|---|---|
| **Descripción** | El sistema debe permitirle al usuario visualizar los préstamos que tiene pendientes, así mismo como un historial de todos los préstamos que ha solicitado. |
| **Prioridad** | Alta |

| | |
|---|---|
| **Alcance** | Es importante para cada uno de los usuarios, porque allí podrán visualizar si tiene algún préstamo pendiente por entregar. |

d. Requisitos no funcionales

| **Requisito: RNF-01 Sistema web responsive** | |
|---|---|
| **Descripción** | El sistema tendrá un diseño responsivo, de modo que, cada una de sus funcionalidades se podrán utilizar en cualquier dispositivo. |
| **Prioridad** | Alta |
| **Alcance** | Dicha funcionalidad es sumamente importante, debido a que el sitio web será funcional en diversas arquitecturas brindando una experiencia de usuario óptima para los clientes del establecimiento. |

| **Requisito: RNF-02 Usabilidad** | |
|---|---|
| **Descripción** | El sistema contará con una interfaz intuitiva para el usuario, se hace uso de paneles y menús de fácil entendimiento de acuerdo a cada uno de los roles establecidos en la problemática. |
| **Prioridad** | Alta |
| **Alcance** | Dicha funcionalidad busca facilitar la comprensión de todos los apartados del sitio web. |

## II. Implementación del sistema

En cuanto a la implementación del sistema, cabe mencionar que se llevó a cabo un proceso de desarrollo flexible con la implementación de varios paquetes para un fácil mantenimiento posterior. Se ejecutaron diversas pruebas para hacer un uso adecuado de las tecnologías con las cuales se realizó el proyecto, tratando siempre de aplicar las buenas prácticas de programación.

Para poder acceder al sistema se debe acceder al siguiente link: Biblioteca (front-end-biblioteca-zg8u-lgzzp8p0p-stevenrincon24.vercel.app)

Credenciales de acceso:

**Perfil administrador**

Usuario: admin@gmail.com

Contraseña: admin

**Perfil empleado**

Usuario: employee@gmail.com

Contraseña: employee

**Perfil cliente**

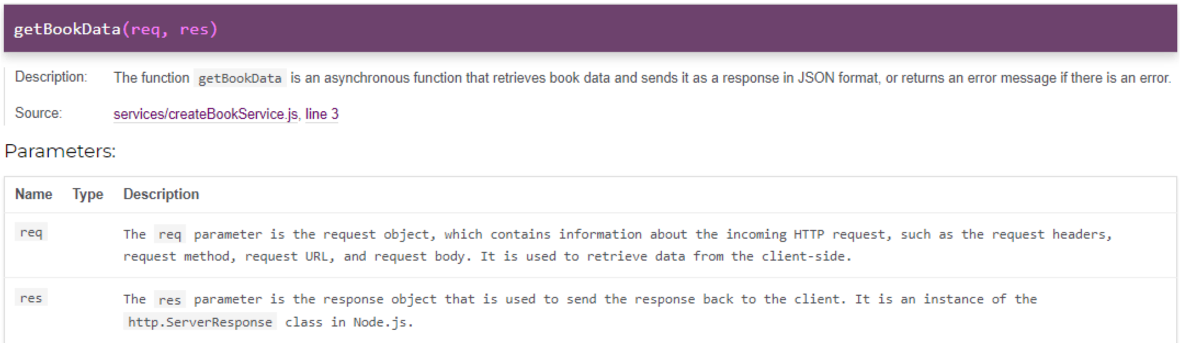
Usuario: customer@gmail.com

Contraseña:1234

a. Documentación del sistema

1. Servicios

- **Book Service**



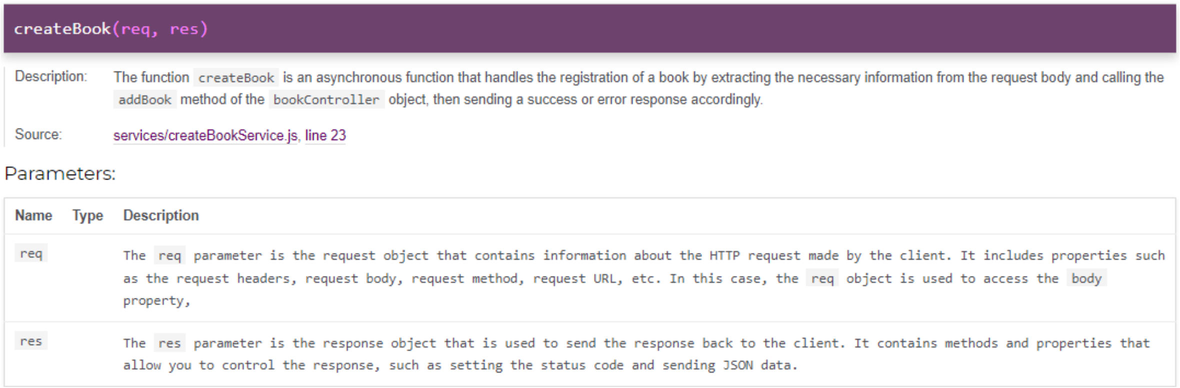
### getBookData(req, res)

Description: The function `getBookData` is an asynchronous function that retrieves book data and sends it as a response in JSON format, or returns an error message if there is an error.

Source: services/createBookService.js, line 3

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object, which contains information about the incoming HTTP request, such as the request headers, request method, request URL, and request body. It is used to retrieve data from the client-side. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an instance of the `http.ServerResponse` class in Node.js. |



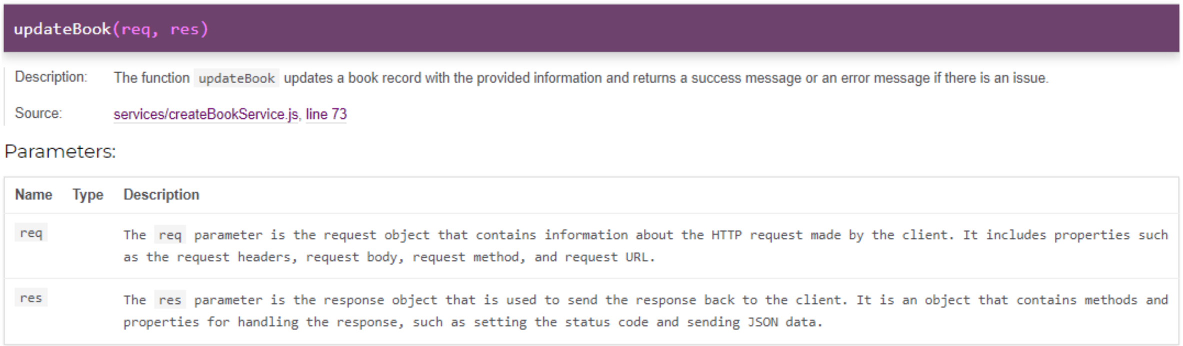
### createBook(req, res)

Description: The function `createBook` is an asynchronous function that handles the registration of a book by extracting the necessary information from the request body and calling the `addBook` method of the `bookController` object, then sending a success or error response accordingly.

Source: services/createBookService.js, line 23

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as the request headers, request body, request method, request URL, etc. In this case, the `req` object is used to access the `body` property, |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to control the response, such as setting the status code and sending JSON data. |



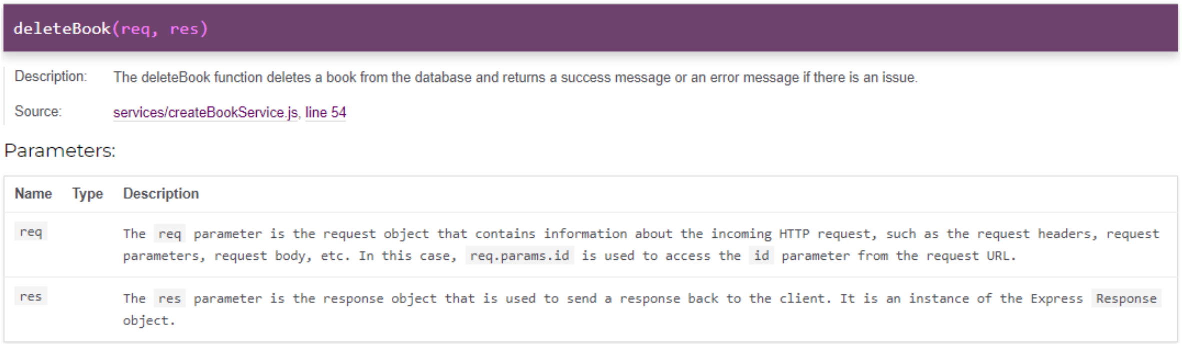
### updateBook(req, res)

Description: The function `updateBook` updates a book record with the provided information and returns a success message or an error message if there is an issue.

Source: services/createBookService.js, line 73

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as the request headers, request body, request method, and request URL. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an object that contains methods and properties for handling the response, such as setting the status code and sending JSON data. |



### deleteBook(req, res)

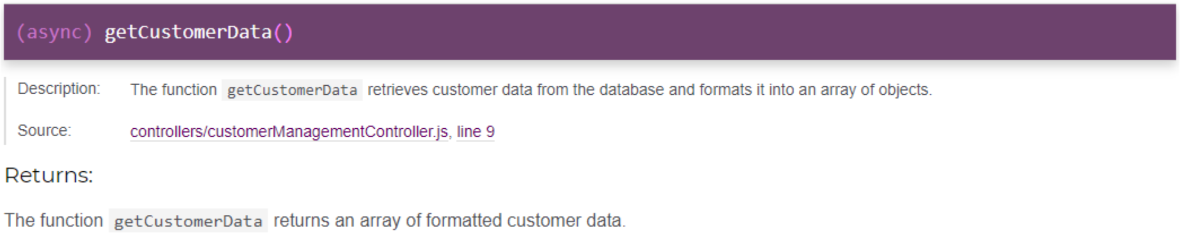
Description: The deleteBook function deletes a book from the database and returns a success message or an error message if there is an issue.

Source: services/createBookService.js, line 54

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the incoming HTTP request, such as the request headers, request parameters, request body, etc. In this case, `req.params.id` is used to access the `id` parameter from the request URL. |
| `res` | | The `res` parameter is the response object that is used to send a response back to the client. It is an instance of the Express `Response` object. |

- **Customer Service**



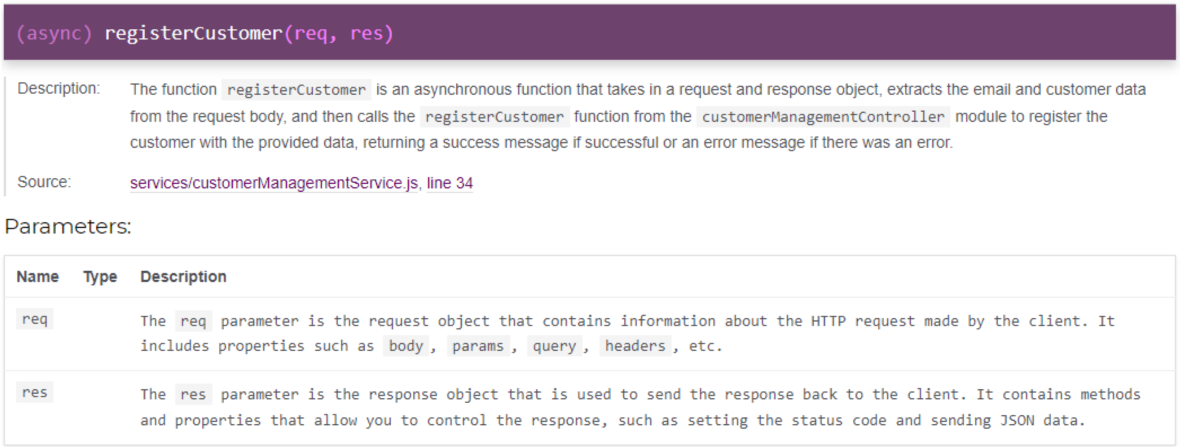
### (async) getCustomerData()

Description: The function `getCustomerData` retrieves customer data from the database and formats it into an array of objects.

Source: controllers/customerManagementController.js, line 9

#### Returns:

The function `getCustomerData` returns an array of formatted customer data.



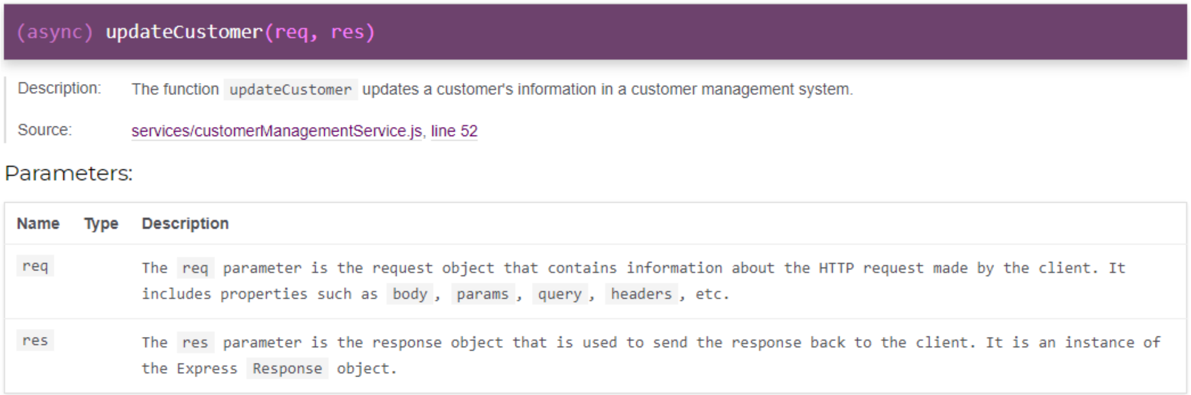
### (async) registerCustomer(req, res)

Description: The function `registerCustomer` is an asynchronous function that takes in a request and response object, extracts the email and customer data from the request body, and then calls the `registerCustomer` function from the `customerManagementController` module to register the customer with the provided data, returning a success message if successful or an error message if there was an error.

Source: services/customerManagementService.js, line 34

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as `body`, `params`, `query`, `headers`, etc. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to control the response, such as setting the status code and sending JSON data. |



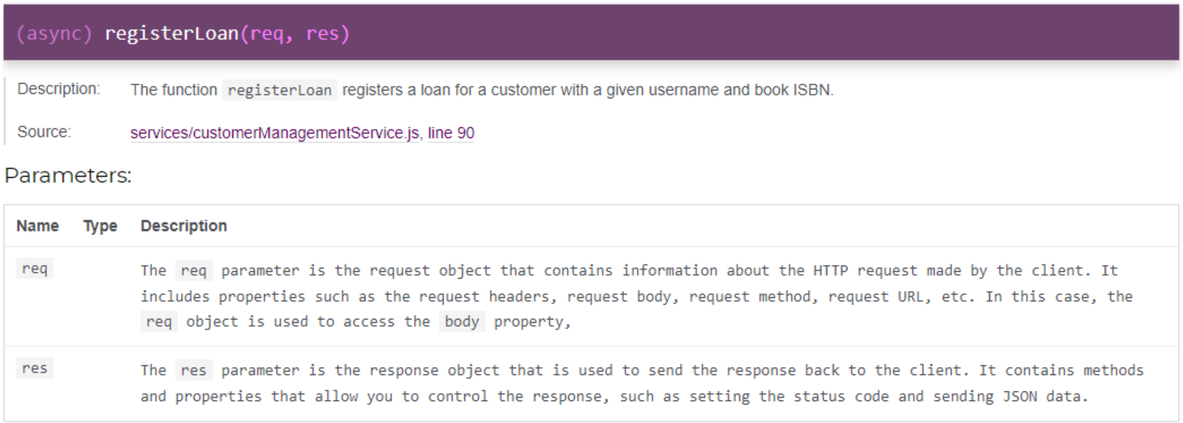
### (async) updateCustomer(req, res)

Description: The function `updateCustomer` updates a customer's information in a customer management system.

Source: services/customerManagementService.js, line 52

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as `body`, `params`, `query`, `headers`, etc. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an instance of the Express `Response` object. |



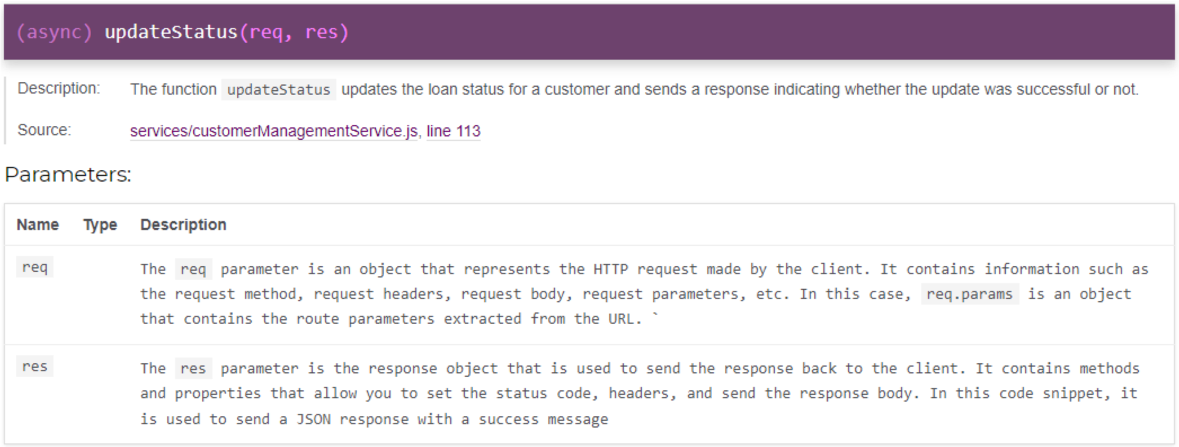
### (async) registerLoan(req, res)

Description: The function `registerLoan` registers a loan for a customer with a given username and book ISBN.

Source: services/customerManagementService.js, line 90

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as the request headers, request body, request method, request URL, etc. In this case, the `req` object is used to access the `body` property, |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to control the response, such as setting the status code and sending JSON data. |



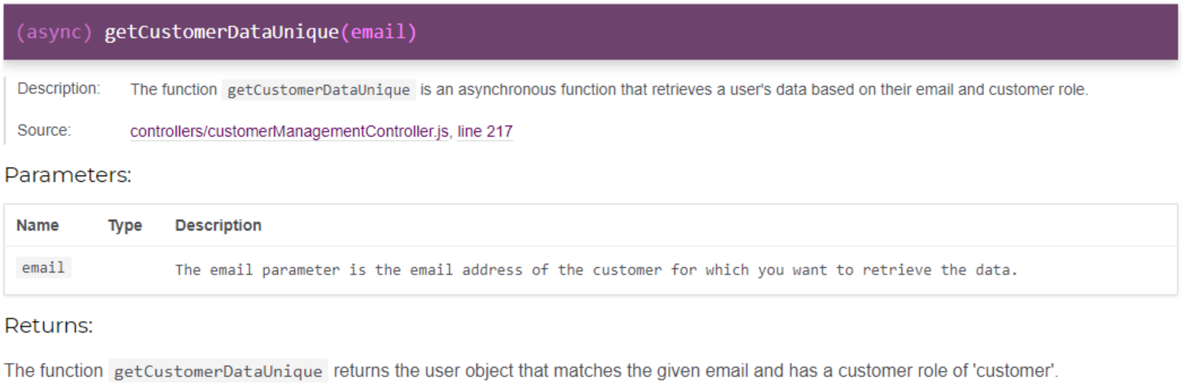
### (async) updateStatus(req, res)

Description: The function `updateStatus` updates the loan status for a customer and sends a response indicating whether the update was successful or not.

Source: services/customerManagementService.js, line 113

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is an object that represents the HTTP request made by the client. It contains information such as the request method, request headers, request body, request parameters, etc. In this case, `req.params` is an object that contains the route parameters extracted from the URL. ` |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to set the status code, headers, and send the response body. In this code snippet, it is used to send a JSON response with a success message |



### (async) getCustomerDataUnique(email)

Description: The function `getCustomerDataUnique` is an asynchronous function that retrieves a user's data based on their email and customer role.

Source: controllers/customerManagementController.js, line 217

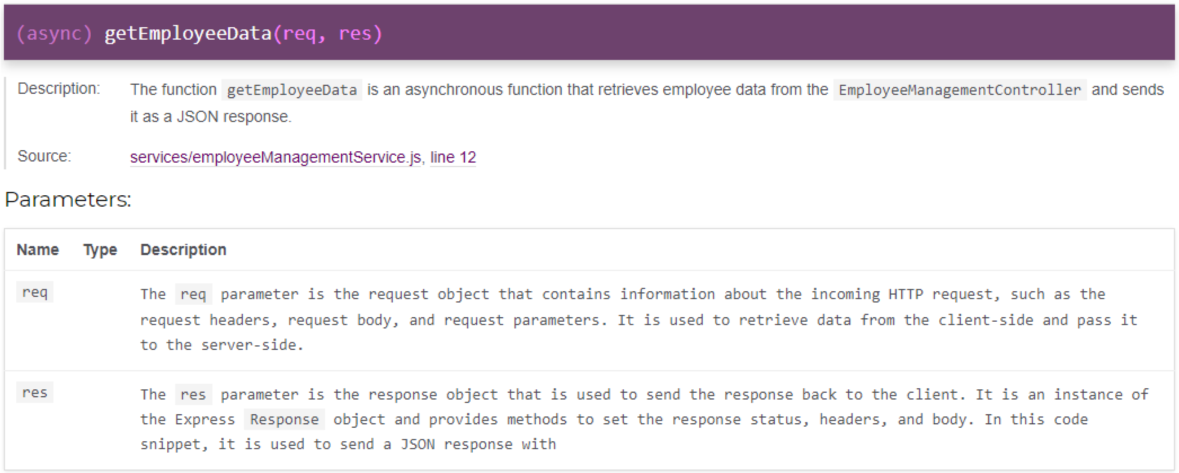
Parameters:

| Name | Type | Description |
|---|---|---|
| `email` | | The email parameter is the email address of the customer for which you want to retrieve the data. |

Returns:

The function `getCustomerDataUnique` returns the user object that matches the given email and has a customer role of 'customer'.

- **Employee Service**



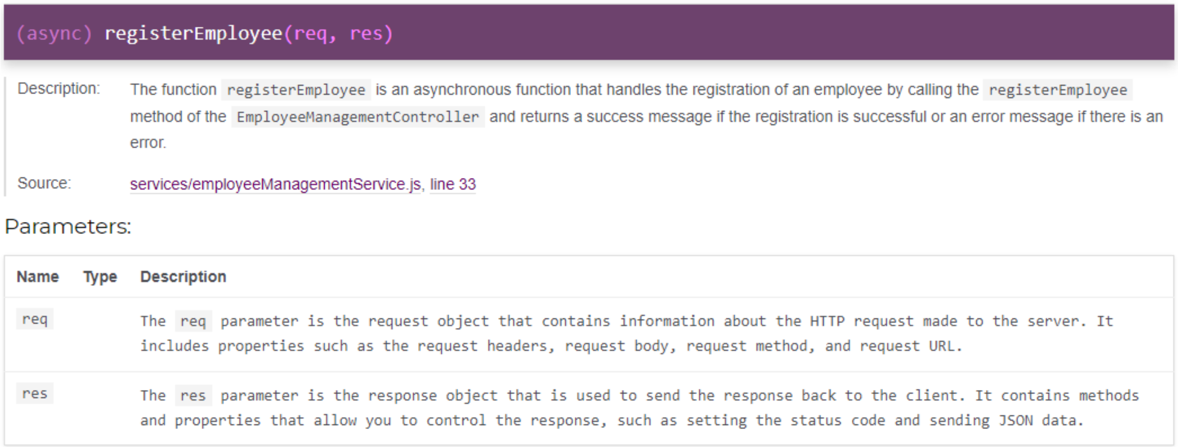
### (async) getEmployeeData(req, res)

Description: The function `getEmployeeData` is an asynchronous function that retrieves employee data from the `EmployeeManagementController` and sends it as a JSON response.

Source: services/employeeManagementService.js, line 12

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the incoming HTTP request, such as the request headers, request body, and request parameters. It is used to retrieve data from the client-side and pass it to the server-side. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an instance of the Express `Response` object and provides methods to set the response status, headers, and body. In this code snippet, it is used to send a JSON response with |



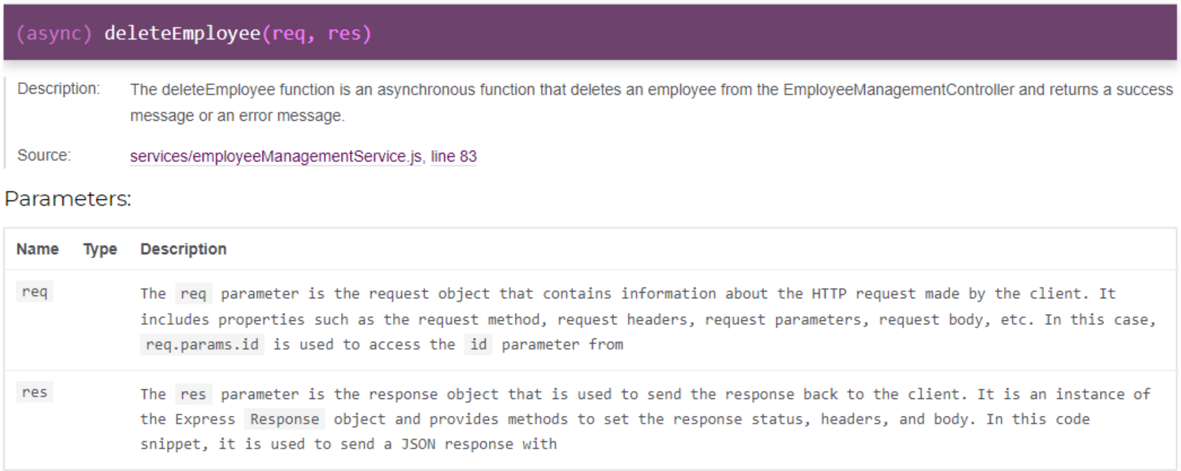
### (async) registerEmployee(req, res)

Description: The function `registerEmployee` is an asynchronous function that handles the registration of an employee by calling the `registerEmployee` method of the `EmployeeManagementController` and returns a success message if the registration is successful or an error message if there is an error.

Source: services/employeeManagementService.js, line 33

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made to the server. It includes properties such as the request headers, request body, request method, and request URL. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to control the response, such as setting the status code and sending JSON data. |



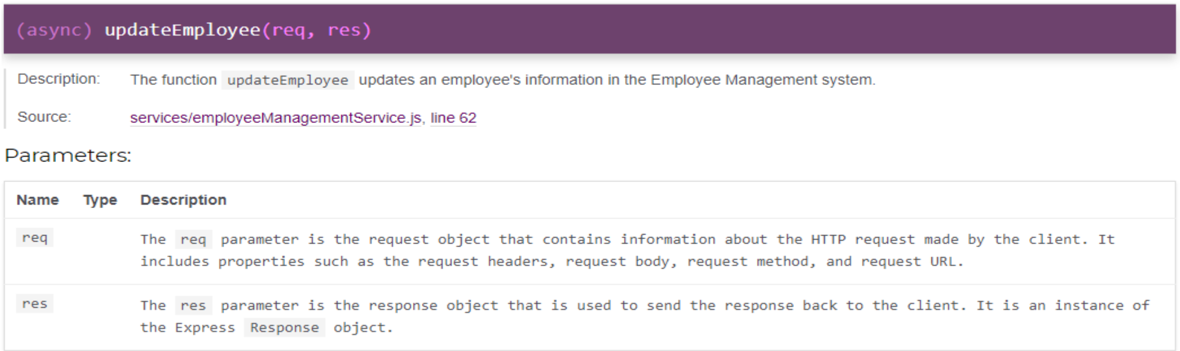
### (async) deleteEmployee(req, res)

Description: The deleteEmployee function is an asynchronous function that deletes an employee from the EmployeeManagementController and returns a success message or an error message.

Source: services/employeeManagementService.js, line 83

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as the request method, request headers, request parameters, request body, etc. In this case, `req.params.id` is used to access the `id` parameter from |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an instance of the Express `Response` object and provides methods to set the response status, headers, and body. In this code snippet, it is used to send a JSON response with |



### (async) updateEmployee(req, res)

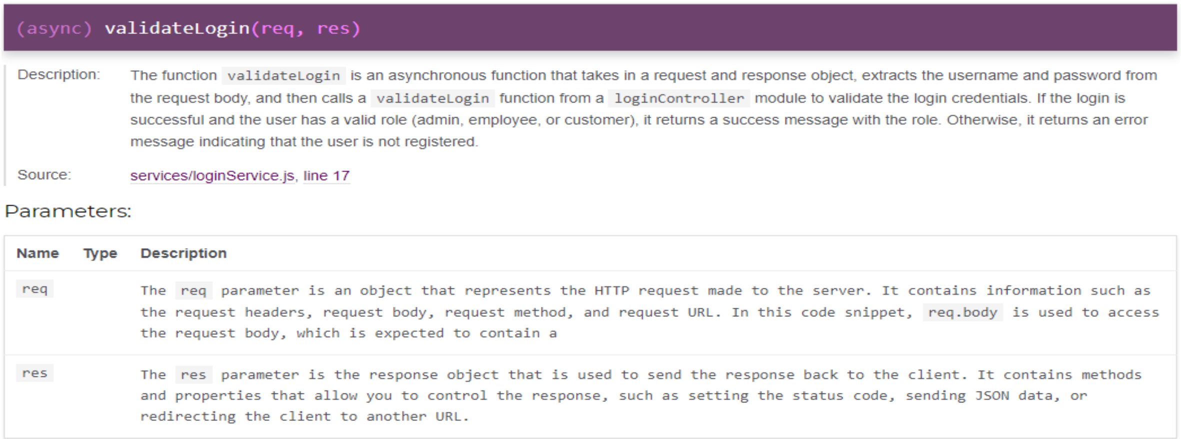
Description: The function `updateEmployee` updates an employee's information in the Employee Management system.

Source: services/employeeManagementService.js, line 62

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is the request object that contains information about the HTTP request made by the client. It includes properties such as the request headers, request body, request method, and request URL. |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It is an instance of the Express `Response` object. |

- **Login Service**



(async) validateLogin(req, res)

Description: The function `validateLogin` is an asynchronous function that takes in a request and response object, extracts the username and password from the request body, and then calls a `validateLogin` function from a `loginController` module to validate the login credentials. If the login is successful and the user has a valid role (admin, employee, or customer), it returns a success message with the role. Otherwise, it returns an error message indicating that the user is not registered.

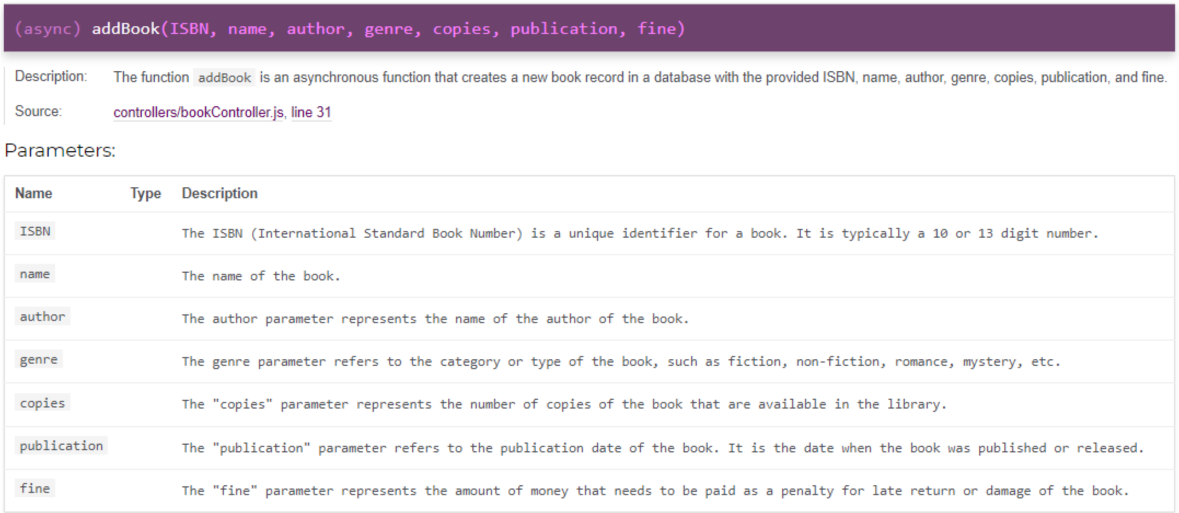
Source: services/loginService.js, line 17

Parameters:

| Name | Type | Description |
|---|---|---|
| `req` | | The `req` parameter is an object that represents the HTTP request made to the server. It contains information such as the request headers, request body, request method, and request URL. In this code snippet, `req.body` is used to access the request body, which is expected to contain a |
| `res` | | The `res` parameter is the response object that is used to send the response back to the client. It contains methods and properties that allow you to control the response, such as setting the status code, sending JSON data, or redirecting the client to another URL. |

## 2. Controladores

- **Book Controller**



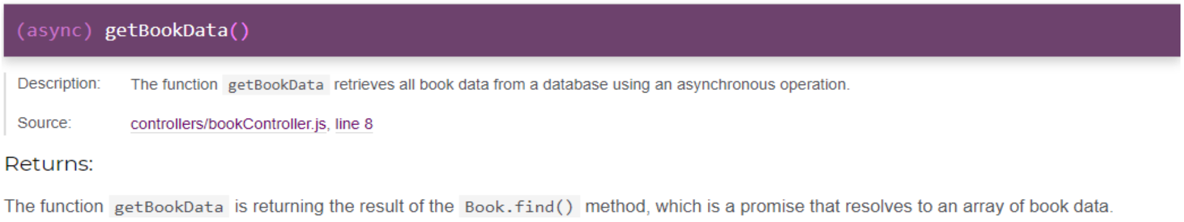
(async) addBook(ISBN, name, author, genre, copies, publication, fine)

Description: The function `addBook` is an asynchronous function that creates a new book record in a database with the provided ISBN, name, author, genre, copies, publication, and fine.

Source: controllers/bookController.js, line 31

Parameters:

| Name | Type | Description |
|---|---|---|
| `ISBN` | | The ISBN (International Standard Book Number) is a unique identifier for a book. It is typically a 10 or 13 digit number. |
| `name` | | The name of the book. |
| `author` | | The author parameter represents the name of the author of the book. |
| `genre` | | The genre parameter refers to the category or type of the book, such as fiction, non-fiction, romance, mystery, etc. |
| `copies` | | The "copies" parameter represents the number of copies of the book that are available in the library. |
| `publication` | | The "publication" parameter refers to the publication date of the book. It is the date when the book was published or released. |
| `fine` | | The "fine" parameter represents the amount of money that needs to be paid as a penalty for late return or damage of the book. |



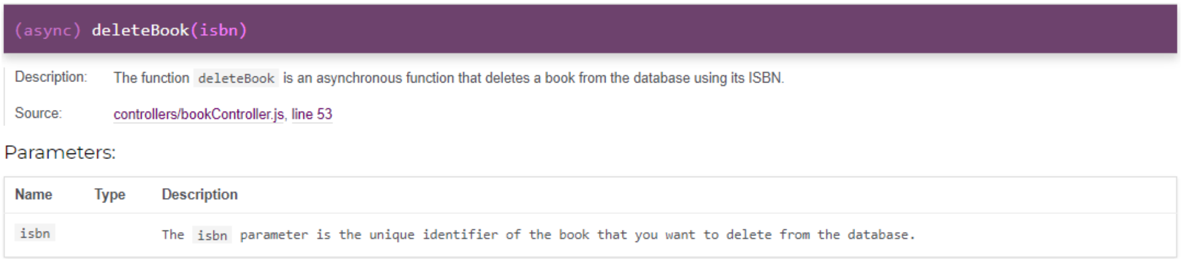
(async) getBookData()

Description: The function `getBookData` retrieves all book data from a database using an asynchronous operation.

Source: controllers/bookController.js, line 8

Returns:

The function `getBookData` is returning the result of the `Book.find()` method, which is a promise that resolves to an array of book data.



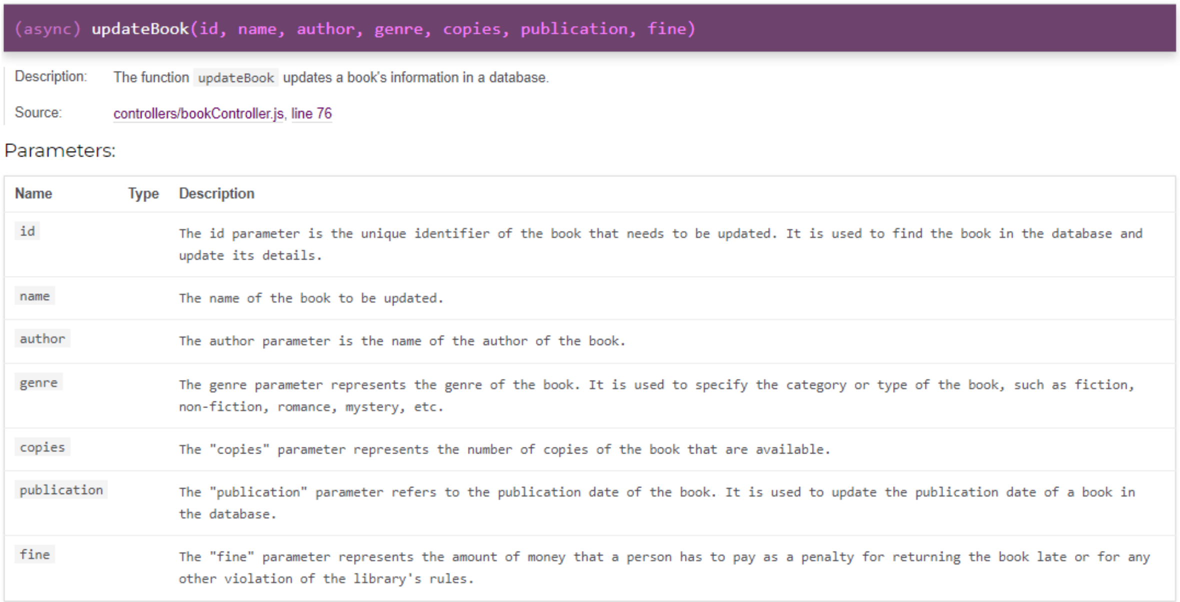
### (async) deleteBook(isbn)

Description: The function `deleteBook` is an asynchronous function that deletes a book from the database using its ISBN.

Source: controllers/bookController.js, line 53

Parameters:

| Name | Type | Description |
|---|---|---|
| `isbn` | | The `isbn` parameter is the unique identifier of the book that you want to delete from the database. |



### (async) updateBook(id, name, author, genre, copies, publication, fine)

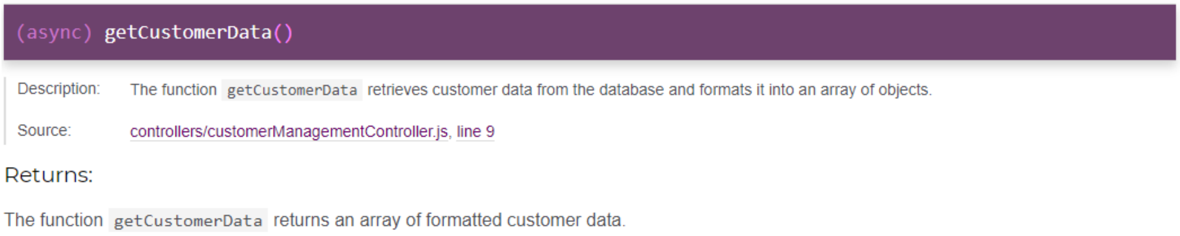
Description: The function `updateBook` updates a book's information in a database.

Source: controllers/bookController.js, line 76

Parameters:

| Name | Type | Description |
|---|---|---|
| `id` | | The id parameter is the unique identifier of the book that needs to be updated. It is used to find the book in the database and update its details. |
| `name` | | The name of the book to be updated. |
| `author` | | The author parameter is the name of the author of the book. |
| `genre` | | The genre parameter represents the genre of the book. It is used to specify the category or type of the book, such as fiction, non-fiction, romance, mystery, etc. |
| `copies` | | The "copies" parameter represents the number of copies of the book that are available. |
| `publication` | | The "publication" parameter refers to the publication date of the book. It is used to update the publication date of a book in the database. |
| `fine` | | The "fine" parameter represents the amount of money that a person has to pay as a penalty for returning the book late or for any other violation of the library's rules. |

- **Customer controller**



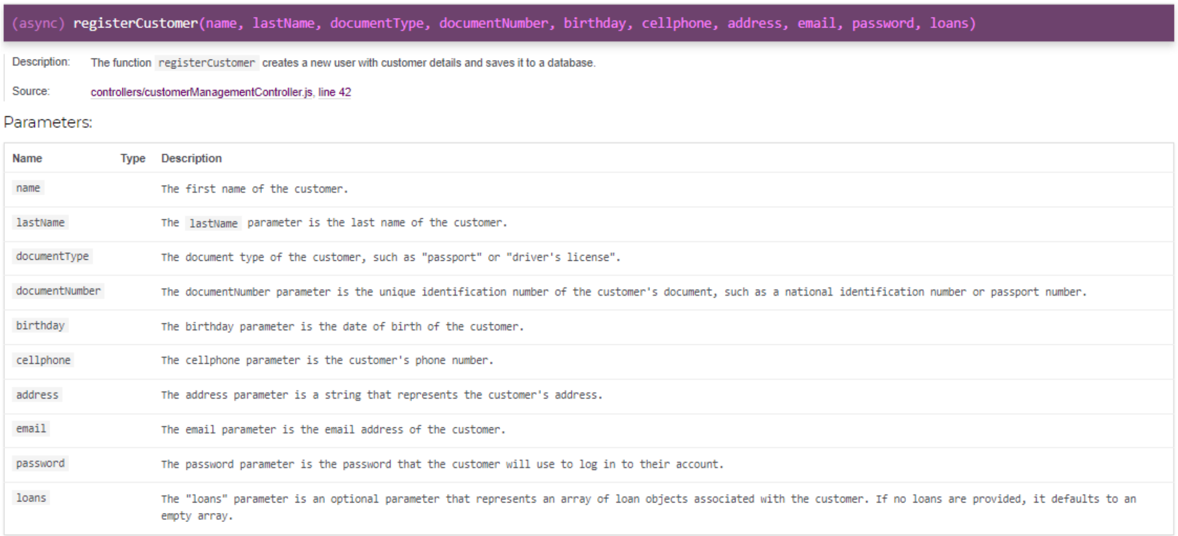
### (async) getCustomerData()

Description: The function `getCustomerData` retrieves customer data from the database and formats it into an array of objects.

Source: controllers/customerManagementController.js, line 9

Returns:

The function `getCustomerData` returns an array of formatted customer data.



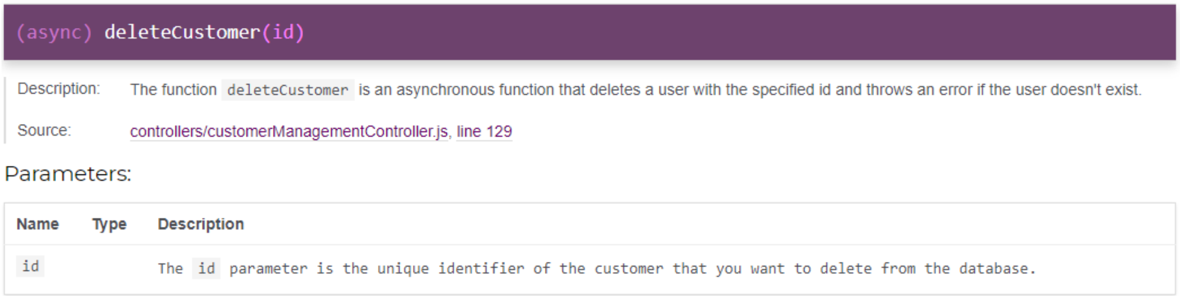
### (async) registerCustomer(name, lastName, documentType, documentNumber, birthday, cellphone, address, email, password, loans)

Description: The function `registerCustomer` creates a new user with customer details and saves it to a database.

Source: controllers/customerManagementController.js, line 42

Parameters:

| Name | Type | Description |
|---|---|---|
| `name` | | The first name of the customer. |
| `lastName` | | The `lastName` parameter is the last name of the customer. |
| `documentType` | | The document type of the customer, such as "passport" or "driver's license". |
| `documentNumber` | | The documentNumber parameter is the unique identification number of the customer's document, such as a national identification number or passport number. |
| `birthday` | | The birthday parameter is the date of birth of the customer. |
| `cellphone` | | The cellphone parameter is the customer's phone number. |
| `address` | | The address parameter is a string that represents the customer's address. |
| `email` | | The email parameter is the email address of the customer. |
| `password` | | The password parameter is the password that the customer will use to log in to their account. |
| `loans` | | The "loans" parameter is an optional parameter that represents an array of loan objects associated with the customer. If no loans are provided, it defaults to an empty array. |



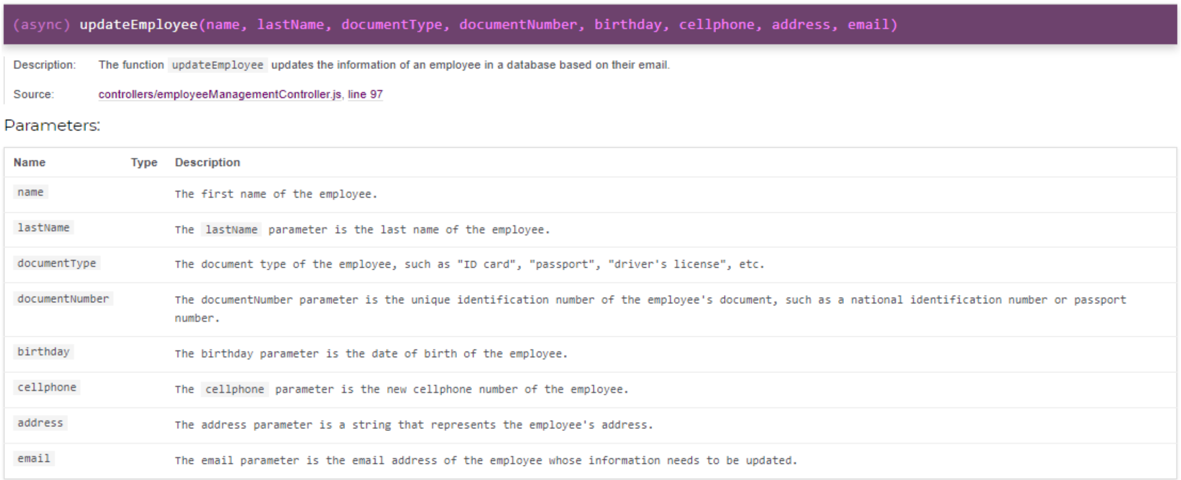
### (async) deleteCustomer(id)

Description: The function `deleteCustomer` is an asynchronous function that deletes a user with the specified id and throws an error if the user doesn't exist.

Source: controllers/customerManagementController.js, line 129

Parameters:

| Name | Type | Description |
|---|---|---|
| `id` | | The `id` parameter is the unique identifier of the customer that you want to delete from the database. |



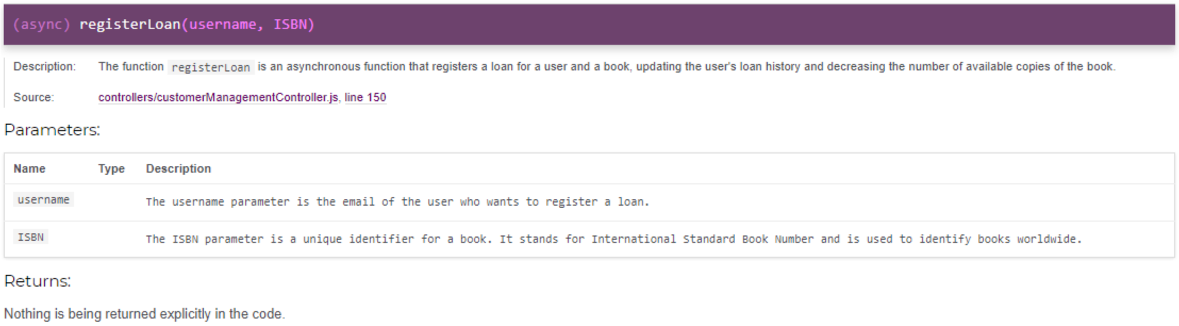
### (async) updateEmployee(name, lastName, documentType, documentNumber, birthday, cellphone, address, email)

Description: The function `updateEmployee` updates the information of an employee in a database based on their email.

Source: controllers/employeeManagementController.js, line 97

Parameters:

| Name | Type | Description |
|---|---|---|
| `name` | | The first name of the employee. |
| `lastName` | | The `lastName` parameter is the last name of the employee. |
| `documentType` | | The document type of the employee, such as "ID card", "passport", "driver's license", etc. |
| `documentNumber` | | The documentNumber parameter is the unique identification number of the employee's document, such as a national identification number or passport number. |
| `birthday` | | The birthday parameter is the date of birth of the employee. |
| `cellphone` | | The `cellphone` parameter is the new cellphone number of the employee. |
| `address` | | The address parameter is a string that represents the employee's address. |
| `email` | | The email parameter is the email address of the employee whose information needs to be updated. |



### (async) registerLoan(username, ISBN)

Description: The function `registerLoan` is an asynchronous function that registers a loan for a user and a book, updating the user's loan history and decreasing the number of available copies of the book.

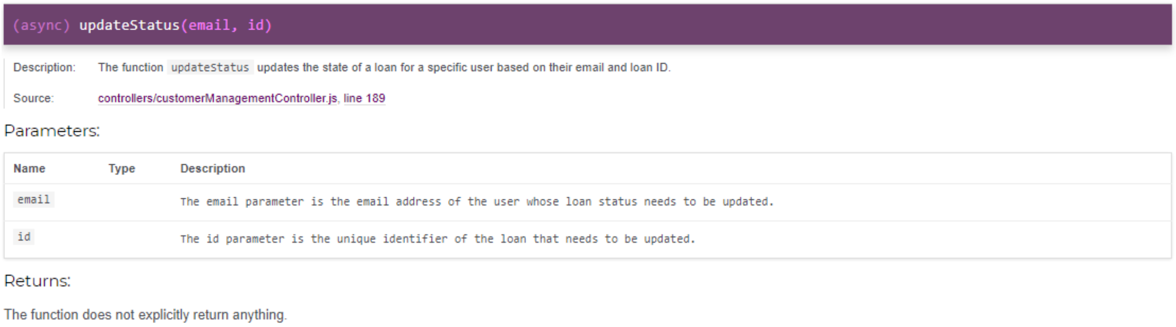
Source: controllers/customerManagementController.js, line 150

Parameters:

| Name | Type | Description |
|---|---|---|
| `username` | | The username parameter is the email of the user who wants to register a loan. |
| `ISBN` | | The ISBN parameter is a unique identifier for a book. It stands for International Standard Book Number and is used to identify books worldwide. |

Returns:

Nothing is being returned explicitly in the code.



### (async) updateStatus(email, id)

Description: The function `updateStatus` updates the state of a loan for a specific user based on their email and loan ID.

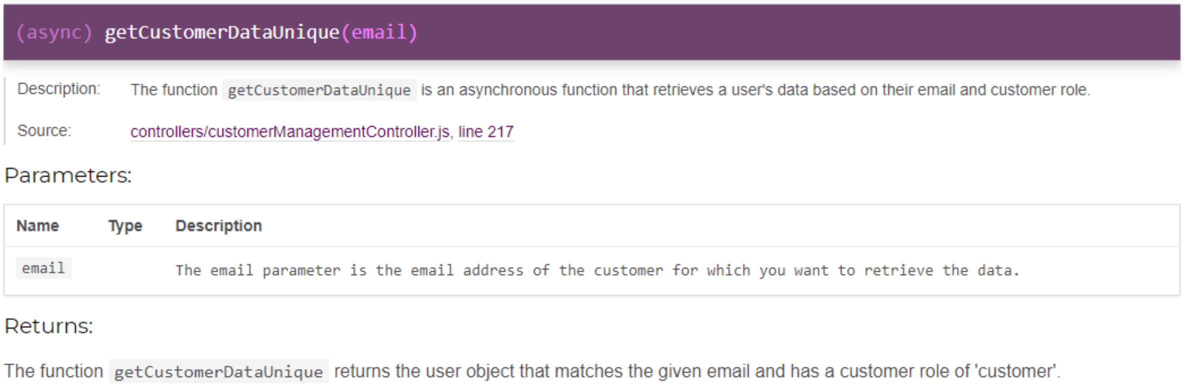
Source: controllers/customerManagementController.js, line 189

Parameters:

| Name | Type | Description |
|---|---|---|
| `email` | | The email parameter is the email address of the user whose loan status needs to be updated. |
| `id` | | The id parameter is the unique identifier of the loan that needs to be updated. |

Returns:

The function does not explicitly return anything.



### (async) getCustomerDataUnique(email)

Description: The function `getCustomerDataUnique` is an asynchronous function that retrieves a user's data based on their email and customer role.

Source: controllers/customerManagementController.js, line 217

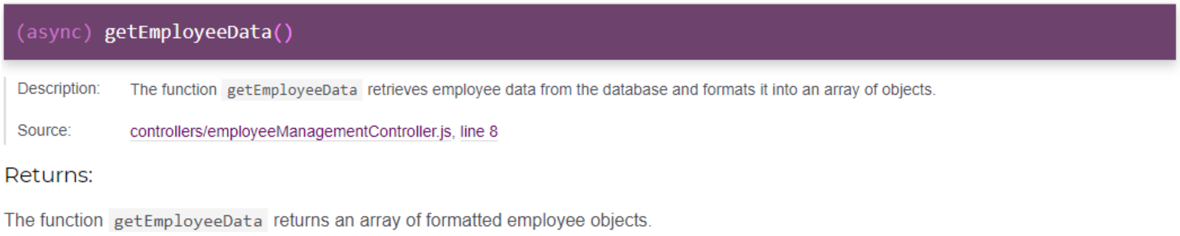
Parameters:

| Name | Type | Description |
|---|---|---|
| `email` | | The email parameter is the email address of the customer for which you want to retrieve the data. |

Returns:

The function `getCustomerDataUnique` returns the user object that matches the given email and has a customer role of 'customer'.

- **Employee Controller**



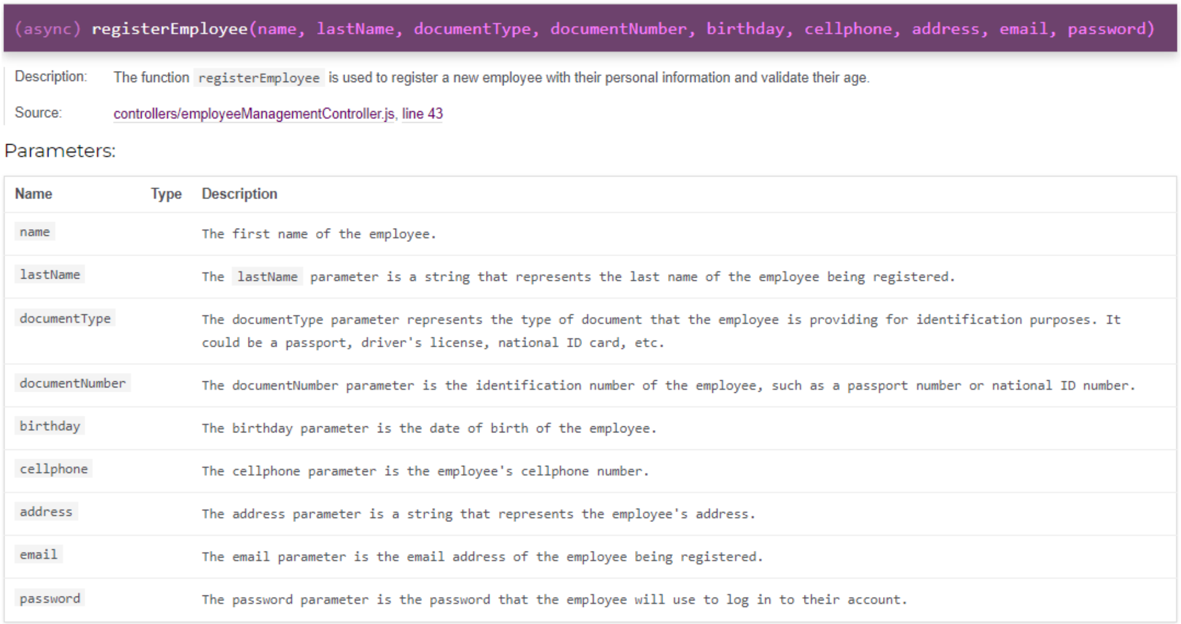
### (async) getEmployeeData()

Description: The function `getEmployeeData` retrieves employee data from the database and formats it into an array of objects.

Source: controllers/employeeManagementController.js, line 8

Returns:

The function `getEmployeeData` returns an array of formatted employee objects.



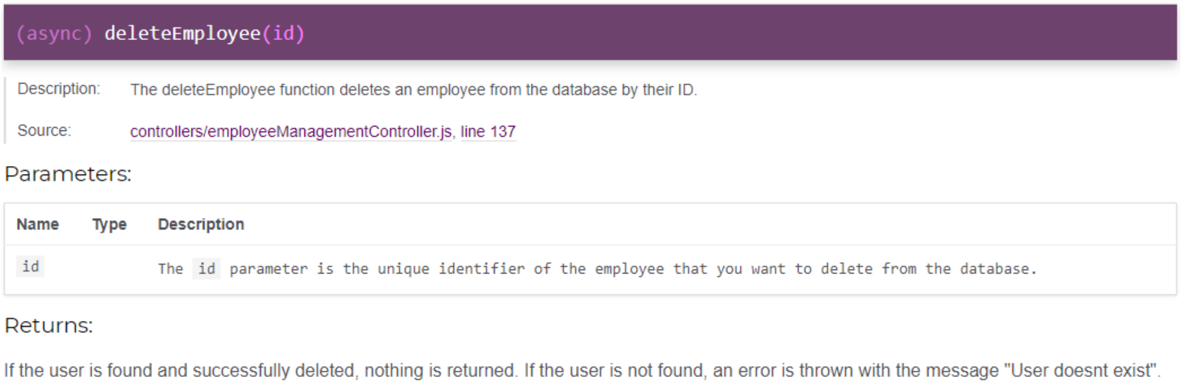
### (async) registerEmployee(name, lastName, documentType, documentNumber, birthday, cellphone, address, email, password)

Description: The function `registerEmployee` is used to register a new employee with their personal information and validate their age.

Source: controllers/employeeManagementController.js, line 43

Parameters:

| Name | Type | Description |
|---|---|---|
| `name` | | The first name of the employee. |
| `lastName` | | The `lastName` parameter is a string that represents the last name of the employee being registered. |
| `documentType` | | The documentType parameter represents the type of document that the employee is providing for identification purposes. It could be a passport, driver's license, national ID card, etc. |
| `documentNumber` | | The documentNumber parameter is the identification number of the employee, such as a passport number or national ID number. |
| `birthday` | | The birthday parameter is the date of birth of the employee. |
| `cellphone` | | The cellphone parameter is the employee's cellphone number. |
| `address` | | The address parameter is a string that represents the employee's address. |
| `email` | | The email parameter is the email address of the employee being registered. |
| `password` | | The password parameter is the password that the employee will use to log in to their account. |



### (async) deleteEmployee(id)

Description: The deleteEmployee function deletes an employee from the database by their ID.

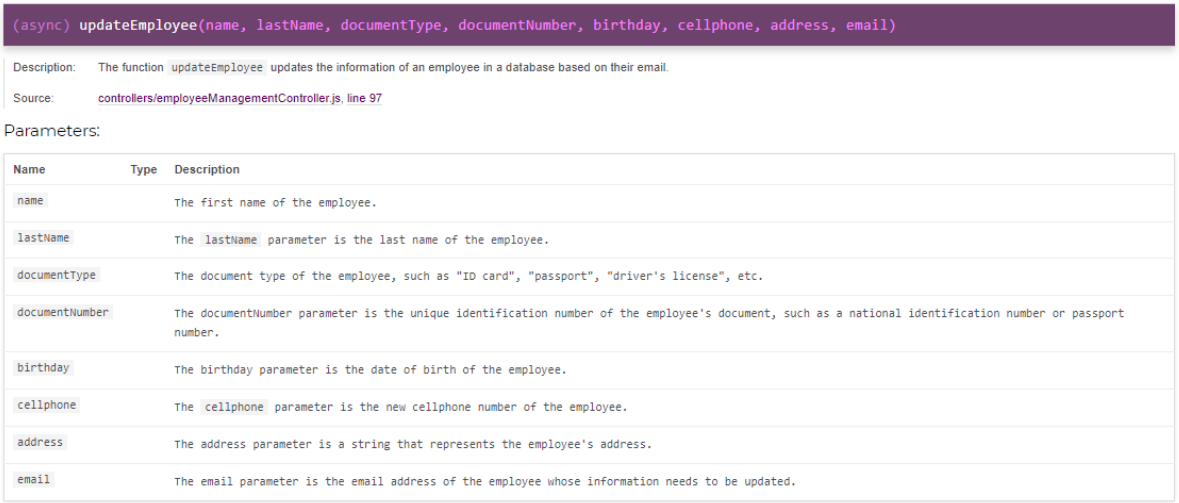
Source: controllers/employeeManagementController.js, line 137

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `id` | | The `id` parameter is the unique identifier of the employee that you want to delete from the database. |

#### Returns:

If the user is found and successfully deleted, nothing is returned. If the user is not found, an error is thrown with the message "User doesnt exist".



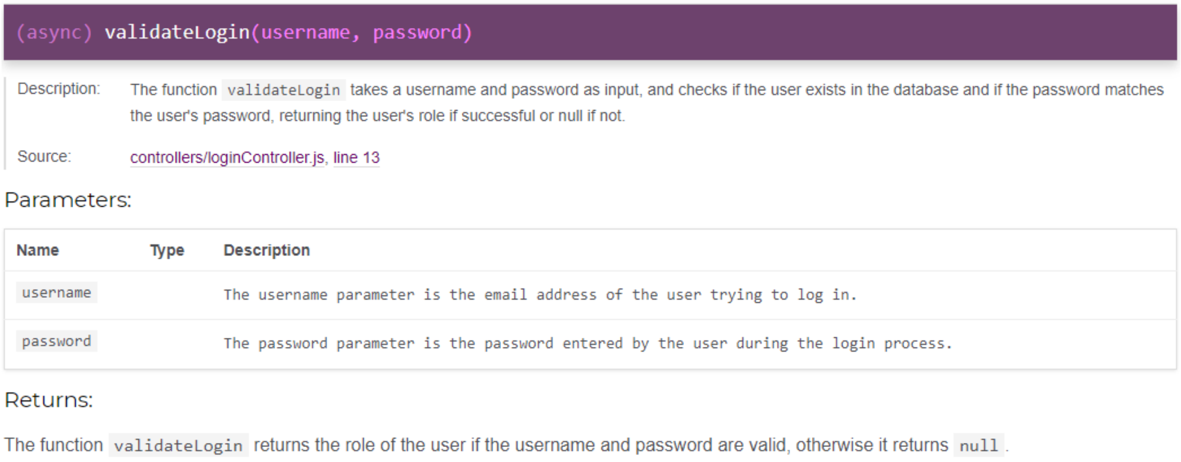
### (async) updateEmployee(name, lastName, documentType, documentNumber, birthday, cellphone, address, email)

Description: The function `updateEmployee` updates the information of an employee in a database based on their email.

Source: controllers/employeeManagementController.js, line 97

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `name` | | The first name of the employee. |
| `lastName` | | The `lastName` parameter is the last name of the employee. |
| `documentType` | | The document type of the employee, such as "ID card", "passport", "driver's license", etc. |
| `documentNumber` | | The documentNumber parameter is the unique identification number of the employee's document, such as a national identification number or passport number. |
| `birthday` | | The birthday parameter is the date of birth of the employee. |
| `cellphone` | | The `cellphone` parameter is the new cellphone number of the employee. |
| `address` | | The address parameter is a string that represents the employee's address. |
| `email` | | The email parameter is the email address of the employee whose information needs to be updated. |



### (async) validateLogin(username, password)

Description: The function `validateLogin` takes a username and password as input, and checks if the user exists in the database and if the password matches the user's password, returning the user's role if successful or null if not.

Source: controllers/loginController.js, line 13

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `username` | | The username parameter is the email address of the user trying to log in. |
| `password` | | The password parameter is the password entered by the user during the login process. |

#### Returns:

The function `validateLogin` returns the role of the user if the username and password are valid, otherwise it returns `null`.