SPRINT 1

User Story

As a Developer, I want to document the inception of the project so that the project is well organized and clear from the beginning.

Acceptance Criteria

- All user stories, tasks, and backlog items derived from functional and technical requirements are documented.
- The project scope is clearly defined.
- Potential challenges and risks for the project are identified.
- An MVC architecture sketch is created and included.
- The information is organized to ensure clarity and shared understanding within the team.

Tasks

- Document all the user stories and their tasks and register them in the project backlog.
- Describe the project scope.
- Document the potential challenges for the project.
- Create the MVC architecture sketch.

User Story

As a Developer, I want to have the project structure and the database ready so that I am able to start the project.

Acceptance Criteria

- The project structure is clearly defined and organized to support future development.
- All domain entities are identified and represented in the database schema.
- Relationships between entities are correctly defined in the E/R diagram.
- The SQL database model is created and ready to support code implementation.
- All tables, keys, and constraints are defined to ensure data integrity.

Tasks

- Design and create the E/R diagram.
- Create the SQL database model for future code implementation.

User Story

As a Developer, I want a clear and consistent design for the application's interfaces so that the UI can be implemented in an organized, coherent, and efficient way.

Acceptance Criteria

- The Login Interface is designed with all necessary fields, actions, and visual elements.
- The User Interface is designed with its main screens and interaction elements.
- The Library Staff Interface is designed, including navigation and functional components.
- Navigation flows between all screens are defined and validated.
- Interaction behaviors (button actions, transitions, error states, etc.) are clearly specified.
- The design is reviewed and approved by the team before implementation begins.

Tasks

- Design the Login Interface.
- Design the User Interface.
- Design the Library Staff Interface.
- Document navigation flows and interaction behaviors.
- Review and validate the designs with the team.

SPRINT 2

User Story

As a Developer, I want a secure and fully structured database with tables, relationships, CRUD methods, and planned security measures, so that user and material data is stored efficiently.

Acceptance Criteria

- A SQL-based database is created.
- All tables and relationships are correctly defined and created using DDL.
- Basic DML operations can be performed on each table.
- The application can successfully connect to the database and pass connection tests.
- Errors are returned when invalid or incomplete data is submitted.
- The necessary security measures to be applied in the future (access control, encryption of sensitive data, etc.) are defined.

Tasks

- Write DDL script defining all core tables, relationships, and integrity constraints, and define planned constraints and security policies for future implementation.
- Implement all DAO settings so that the app can connect to the database.
- Implement base DAO methods.
- Implement centralized DAO error handling using DAOException to convert SQLExceptions into application-level errors.
- Create initial seed data for the database to enable development and testing.

User Story

As a Library Staff, I want to register users so they can borrow items.

Acceptance Criteria

- The library staff can add a new user by entering the required information (name, surname, ID, etc.).
- The system validates required fields and returns errors if information is missing or invalid.
- A confirmation message appears after each successful registration.

Tasks

- Design the UserRegistrationView interface.
- Implement the UserRegistrationController with all necessary methods to register a new user.
- Implement the UserService with messages preventing error pass-through.
- Write JUnit tests for UserService.

User Story

As any General User, I want to log in so that the system can identify my role and give me access to the correct interface.

Acceptance Criteria

- The application displays one simple login form requiring a Username/ID and Password for all users.
- Access is granted only when valid credentials (ID/Username and Password) are provided.
- An error message is displayed when invalid credentials are provided (e.g., wrong password or unknown user).
- Upon successful login, the system checks the user's role.
- If the role is 'Admin', the user is redirected to the Admin Management Menu.
- If the role is 'User', the user is redirected to their Personal Dashboard.
- The system prevents unauthorized access to role-specific pages.

Tasks

- Implement LoginService to authenticate users by username and password and return the User object if credentials are valid, including access to the user's role.
- Implement the Login UI.
- Implement the Login Controller to read the returned user role and redirect to the Admin Menu or the Personal Dashboard accordingly.
- Create and implement basic separate landing pages for the Admin Menu and the Personal Dashboard.
- Implement a JUnit test to validate the feature.

User Story

As a Library Staff, I want to add new material so that it is available in the catalog.

Acceptance Criteria

- The system allows the library staff to input all required fields (title, author, year, genre, ISBN, material type).
- The system prevents saving if any required field is missing.
- When the library staff submits the form, the new material is successfully stored in the database.
- The material appears immediately in the catalog search results.
- The system assigns the keyword "available" as the status.
- The system logs the event as "Material updated to catalog".
- If an error occurs during insertion, the system displays an error message and logs it.

Tasks

- Design 'Add Material' UI form.
- Implement MaterialController.addMaterial() to handle input.
- Implement MaterialService for business logic.
- Implement MaterialDAO.create() for JDBC persistence.
- Write JUnit tests.

SPRINT 3

User Story

As a Library Staff, I want access to a register of all users so I can manage accounts and remove or add Users.

Acceptance Criteria

- The Library Staff can access the User Management menu through a button in the Administration menu.
- The library staff can add a new user by clicking on the “Add User” button.
- The system redirects the Library Staff to the Add User dialog box to complete a registration.
- The library staff can remove an existing user by selecting the user to delete.
- The system prevents removing users with active loans.
- A message appears confirming the deletion.

Tasks

- Develop the User List View to display all existing user accounts.
- Integrate the "Add User" button into the User Management view.
- Implement the redirection to the Add User dialog window.
- Implement the "Remove/Delete" action for selected users in the list.
- Implement the Active Loans logic to check the database for any active loans and prevent deletion if they exist.
- Implement a confirmation message when clicking on "Delete".
- Develop the backend logic to correctly delete the user from the Database.
- Write all necessary JUnit tests to check that the acceptance criteria are met.

User Story

As a Library Staff, I want to see a list of materials so I can search in the catalog.

Acceptance Criteria

- The system displays a list of all catalog materials (Books, CDs, magazines, etc.).
- Each material shows the availability status, title, author, year, and ISBN if applicable.
- The list loads without errors.
- A basic search function allows the library staff to filter materials by their attributes.
- The system responds within an acceptable time (< 2 seconds per query).

Tasks

- Implement MaterialDAO.findAll() and MaterialDAO.findByCriteria() with optimized SQL.
- Implement MaterialCatalogController.showCatalog() to call the service and design the Catalog List View (UI) including the search bar.
- Write JUnit tests for MaterialCatalogService and EditService and verify error-free functioning.

User Story

As a Library Staff, I want to be able to remove and modify materials so I can delete old ones or correct wrong fields.

Acceptance Criteria

- The list of materials displays all saved materials in the library.
- The library staff can select a material and choose to modify or delete it using two separate buttons.
- Clicking "Modify" opens a pop-up dialog pre-filled with the selected material's data.
- Modifications can be saved using a "Save" button, and the database is updated correctly.
- Clicking "Delete" opens a confirmation dialog asking the user to confirm the action.
- Both edit and delete actions update the database correctly.

Tasks

- Add "Edit" and "Delete" buttons in the Material List view if not already implemented.
- Create a Modification Dialog triggered by the "Edit" button, pre-filled with the selected material's data.
- Implement the "Save" button functionality on the Modification Dialog.
- Create a Confirmation Dialog triggered by the "Delete" button to confirm deletion.

User Story

As a Library Staff, I want to loan materials so that Users can borrow items.

Acceptance Criteria

- The library staff can select the material and a user to create a loan record.
- The system prevents loaning a material that is already loaned.
- After a loan is created, the material's availability status updates to "loaned".
- A confirmation message is shown after a successful loan.
- If required information is missing, the system displays an error message.

Tasks

- Design 'New Loan' UI (select user/material, set due date).
- Implement the Loans Controller.
- Implement LoanService.createLoan() logic for availability check and status update.
- Implement transactional LoanDAO.createLoan() (creates loan + updates item status).
- Implement a JUnit test.
- Implement filtering MaterialDAO.selectBy().

User Story

As a Library Staff, I want to return or correct loaned materials so that availability and records are accurate.

Acceptance Criteria

- The library staff can select a loaned material and mark it as returned.
- The library staff can edit a loan if they mistakenly selected the wrong material or need to correct the user's DNI.
- The system updates the material's availability to "available".
- The system records the return date in the loan record.
- If the material is not currently on loan, the system prevents a return or edit action.
- A confirmation message is shown after a successful return or edit.

Tasks

- Design 'Return/Edit Material' UI – allow selecting a loaned item, returning it, or editing the user/material info.
- Implement LoanController.processReturnOrEdit() – handle item selection, return, or edit, and provide feedback.
- Implement transactional LoanDAO.updateReturnOrEdit() – record return date, change item status to "available", and update corrected user info.
- Write JUnit tests for the Loans functionality.

User Story

As a Library Staff, I want to see overdue loans so I can manage delays.

Acceptance Criteria

- The system displays a list of overdue loans automatically based on due dates.
- Each record includes the material type, title, author, ISBN, user, due date, and whether it is delayed.
- The list updates automatically when loans become overdue.
- A search bar allows filtering by keywords such as “delayed”, “delay”, “delays”, “late”, or “overdue”.

Tasks

- Implement DAO methods for Loans to obtain active loans where the due date is past the current date (Delayed Loans).
- Implement LoanController.showOverdueList() to render the overdue loans list.
- Design the 'Overdue Loans' UI to display material type, title, author, ISBN, user, due date, and delayed status.
- Add search-bar functionality to filter the loans list by relevant keywords.

SPRINT 4

User Story

As a User, I want to reserve a material so I can pick it up later.

Acceptance Criteria

- The user can select a material and submit a reservation request.
- The system prevents reservations for materials that are already holded or loaned.
- Reservations are stored in the database and linked to the user.
- The material's availability status is updated to "holded".
- The user can cancel a holded material.

Tasks

- Display a dynamic “Hold / Holded / Unavailable” button for each catalog entry based on material availability and the current user’s hold state.
- Implement a service-layer method that atomically updates material availability and stores reservation records.
- Implement controller logic to handle user interactions for placing and releasing holds using a service-layer abstraction.
- Prevent users from attempting to reserve materials that are already reserved or loaned.

User Story

As a User, I want to search by filters so I can find materials faster.

Acceptance Criteria

- Users can apply filters such as title, author, year, genre, or category.
- The system returns only materials that match the selected filters.
- Results load without errors.
- Filters can be reset to view the full catalog.

Tasks

- Design the Advanced Filter UI (View) supporting Title, Author, Year, Genre, and Category toggles.
- Implement client-side filtering logic inside UserCatalogController for grouped materials, using UI-driven filter state.
- Implement a similar functionality as the MaterialCatalogService but for grouped materials.
- Implement a reset mechanism to clear all active filters and restore the full catalog view.

User Story

As a User, I want to see my consolidated history of loans and reservations, and be able to cancel any of my active reservations.

Acceptance Criteria

- The page displays two lists containing both Loan records and Reservation records.
- Each entry clearly shows the associated Material (title), relevant Dates (Due Date), and the current Status.
- Overdue loan items are calculated as "Delayed" and visually marked for quick identification.
- Only the authenticated user's historical information is displayed.
- The user can cancel any active reservation directly from this page.

Tasks

- Design and implement the 'History' View to display the merged and sorted records with visual marking for delayed items. Add a "Cancel Reservation" button and its handler (handleCancelReservation()) to the UI.
- Implement LoanDAO.getHistory(userId) and ReservationDAO.getHistory(userId) to fetch all user loans and reservations as LoanRow and HoldRow objects. Implement ReservationDAO.delete(reservationId) to allow cancellation of active reservations.
- Implement UserController.showHistory() to securely pass the userId and render the view. Ensure the controller handles user interaction (cancellation) and triggers a view refresh afterward.
- Write JUnit tests for LoanService.compileHistory() (ensuring accurate status calculation and merging) and for ReservationService.cancelReservation() (ensuring deletion success and proper error handling).

User Story

As a User, I want to be notified of overdue loans and active holds so that I can manage my borrowed and reserved materials on time.

Acceptance Criteria

- The application identifies overdue loans and active holds automatically.
- A notification is generated for each overdue loan or active hold each time the user logs in.
- Notifications are visible to the user in the application.
- Notifications include the details of the delayed or holded material.
- Notifications are not duplicated.
- The "Notifications" button visually indicates when there are active notifications.

Tasks

- Implement the SQL method to obtain all overdue loans and active holds for the logged-in user and store the result fields (Loan and Hold details) to be displayed in the "Notifications" section.
- Connect this method to the initialization of the User Landing Interface so the notifications query runs each time the interface is initialized.
- Implement backend logic that makes the "Notifications" button turn red if the query returns a non-empty list of overdue loans or active holds.
- Create the Notifications interface.
- Generate the controller for the Notifications interface.
- Generate a NotificationsService and a JUnit test that checks there are no duplicates and verifies all notification-related scenarios.

User Story

As a Developer, I want to check that the MVC pattern is followed thanks to the use of independent controllers, so that the codebase remains modular and maintainable.

Acceptance Criteria

- Each interface in the application is controlled by its own Controller file.
- Each Controller is properly linked to its FXML view.
- All views and controllers are integrated coherently in the Main App.
- The codebase remains modular, maintainable, and consistent with the MVC pattern.

Tasks

- Check that the MVC pattern is correctly followed across all interfaces.

User Story

As a Developer, I want to make sure that all aspects of the app are consistent and coherent from both the functionality and graphical point of view.

Acceptance Criteria

- All interfaces have the same design format, window aspect, and size.
- All menus include a "Back to home" button.
- Search functions follow a consistent pattern across all views.
- All other functionalities are implemented according to the requirements.
- Any unnecessary or garbage files generated during development are removed.
- The application is visually coherent and provides a consistent user experience.

Tasks

- Check that all window sizes are consistent.
- Check the graphic consistency of the entire application.
- Filter out garbage files that may have been created during development.
- Run through the list of requirements to validate all functionalities are met.
- Check all methods and classes are well documented and the comments of the code are in english.