**UWM – API Components**

**API Components:**

1. Borrower API: Handles operations related to borrowers, such as creating a new borrower, retrieving borrower information, updating borrower details, and deleting a borrower.
2. Loan Application API: Manages loan applications, including creating a new application, retrieving application details, updating application information, and deleting an application.
3. Employment API: Deals with employment details, such as adding employment information for a borrower, retrieving employment history, updating employment details, and removing employment records.
4. Asset API: Handles asset-related operations, such as adding assets for a borrower, retrieving asset information, updating asset details, and deleting assets.
5. Liability API: Manages liabilities, including adding liabilities for a borrower, retrieving liability information, updating liability details, and removing liabilities.
6. Credit Report API: Handles credit reports, including retrieving credit reports for borrowers, updating credit scores, and managing credit history.
7. Rate Lock API: Manages rate lock information, including creating rate locks for loan applications, retrieving rate lock details, updating rate lock information, and removing rate locks.

**Skeletal Structure:** Below is a skeletal structure showcasing the API endpoints and their corresponding functionalities:

* **/api/borrowers**
  + **POST /** - Create a new borrower
  + **GET /{borrowerId}** - Retrieve borrower information
  + **PUT /{borrowerId}** - Update borrower details
  + **DELETE /{borrowerId}** - Delete a borrower
* **/api/loanApplications**
  + **POST /** - Create a new loan application
  + **GET /{applicationId}** - Retrieve application details
  + **PUT /{applicationId}** - Update application information
  + **DELETE /{applicationId}** - Delete an application
* **/api/borrowers/{borrowerId}/employment**
  + **POST /** - Add employment details for a borrower
  + **GET /{employmentId}** - Retrieve employment information
  + **PUT /{employmentId}** - Update employment details
  + **DELETE /{employmentId}** - Delete employment records
* **/api/borrowers/{borrowerId}/assets**
  + **POST /** - Add assets for a borrower
  + **GET /{assetId}** - Retrieve asset information
  + **PUT /{assetId}** - Update asset details
  + **DELETE /{assetId}** - Delete assets
* **/api/borrowers/{borrowerId}/liabilities**
  + **POST /** - Add liabilities for a borrower
  + **GET /{liabilityId}** - Retrieve liability information
  + **PUT /{liabilityId}** - Update liability details
  + **DELETE /{liabilityId}** - Delete liabilities
* **/api/borrowers/{borrowerId}/creditReport**
  + **GET /** - Retrieve credit report for a borrower
  + **PUT /** - Update credit score and history
* **/api/loanApplications/{applicationId}/rateLock**
  + **POST /** - Create a rate lock for a loan application
  + **GET /{rateLockId}** - Retrieve rate lock details
  + **PUT /{rateLockId}** - Update rate lock information
  + **DELETE /{rateLockId}** - Remove a rate lock

**User Story: API Integration and Skeleton Structure**

**Description:** As a developer, I want to integrate APIs into the application, establish the skeletal structure for API, UI, and DB integration, and ensure seamless communication and data flow between the components.

**Acceptance Criteria:**

1. APIs are successfully integrated with the application.
2. The API, UI, and DB are connected and able to exchange data.
3. The API endpoints are properly secured and adhere to authentication and authorization requirements.
4. The UI can fetch data from the API and display it to the user.
5. User actions in the UI trigger the appropriate API calls for data manipulation and storage.
6. Data sent from the UI is correctly processed by the API and stored in the DB.
7. Data retrieved from the DB is sent back to the UI in the expected format.
8. Error handling and appropriate error messages are implemented for failed API requests.
9. Performance and scalability considerations are taken into account for API integration.

**Details of the Skeletal Structure:**

1. API Layer:
   * Define RESTful API endpoints for each functionality, following industry best practices and conventions.
   * Implement API controllers to handle incoming requests, validate input, and invoke necessary services.
   * Integrate authentication and authorization mechanisms to secure the API endpoints.
   * Implement error handling and return appropriate HTTP status codes and error messages.
2. Service Layer:
   * Develop service components responsible for handling business logic and data manipulation.
   * Implement service methods to process requests from the API controllers, interact with the database, and perform necessary validations.
3. Data Access Layer (DB):
   * Design and implement database schema based on the application's data requirements.
   * Create database tables, indexes, and relationships according to the entity model.
   * Implement data access methods (e.g., CRUD operations) to interact with the database using an ORM (Object-Relational Mapping) or raw SQL queries.
4. UI Layer:
   * Design and develop user interface components using the chosen frontend framework (e.g., React, Angular, Vue.js).
   * Configure API endpoints in the UI components to fetch data from the API and display it to the user.
   * Implement UI components to capture user input and trigger the appropriate API calls for data manipulation and storage.
5. Integration and Testing:
   * Establish communication between the UI and API layers using HTTP/HTTPS protocols.
   * Implement unit tests for API endpoints, services, and database operations to ensure proper functionality and data integrity.
   * Perform integration testing to validate the end-to-end flow of data between the API, UI, and DB.
   * Monitor and optimize the performance of the API, considering factors like response times and concurrent user handling.