**Project Implementation Structure: Alt-Credit Scoring Model**

This guide assumes you will follow the technical stack outlined in your document (React, Node.js, Python, MongoDB).

**Phase 1: Data Foundation - Synthetic Data Generation (Python)**

This is the most critical first step. The quality of your entire model depends on generating a realistic and complex dataset.

* **Objective:** Create a Python script to generate a 50,000-record dataset based on the rules in your "Synthetic Data Generation Framework."
* **Key Tasks:**
  1. **Setup:** Initialize a Python environment and install necessary libraries (numpy, pandas, scipy).
  2. **Feature Simulation:**
     + Define all 35+ features.
     + Use statistical distributions (beta, poisson, gamma) to simulate independent risk features like utility\_payment\_ratio, app\_diversity, etc.
  3. **Implement Conditional Logic:**
     + Code the composite income\_tier calculation.
     + Implement the contextual spending risk matrices (OTT, food, ride-hailing) that depend on the income\_tier.
  4. **Risk Score Calculation:**
     + Write a function to calculate the total\_risk score by applying the specified weights to each feature category.
  5. **Default Generation:**
     + Implement the logistic function 1 / (1 + np.exp(-0.25 \* total\_risk + 5)) to generate the default\_prob.
     + Use the probability to assign a binary loan\_default flag (1 or 0) for each borrower.
  6. **Finalize Dataset:** Combine all features and the target variable (loan\_default) into a single Pandas DataFrame and save it as a CSV file (e.g., synthetic\_credit\_data.csv).

**Phase 2: Core Intelligence - ML Modeling & Explainability (Python)**

Now that you have data, you can build the predictive model.

* **Objective:** Train robust classification models and integrate an explainability layer.
* **Key Tasks:**
  1. **Model Training:**
     + Load the synthetic\_credit\_data.csv.
     + Split the data into training and testing sets (e.g., 80/20 split).
     + Train both **XGBoost** and **Random Forest** classifiers on the training data.
  2. **Model Evaluation:**
     + Evaluate the models on the test set using key metrics: **AUC-ROC, F1-Score, Precision, and Recall.**
     + Select the best-performing model to move forward with.
  3. **Implement Explainability with SHAP:**
     + Integrate the SHAP library with your chosen model.
     + Generate **global feature importance** plots to show which features are most influential overall.
     + Create a function that can take a **single borrower's data** and generate a **local SHAP explanation** (e.g., a force plot) to show why that specific prediction was made.
  4. **Host the Model:**
     + Wrap your trained model and the SHAP explanation function in a simple web server using **Flask** or **FastAPI**. This will create API endpoints that your main application can call.

**Phase 3: System Backbone - Backend & API Development (Node.js)**

This phase focuses on building the server-side logic that connects everything.

* **Objective:** Create a secure and scalable backend with RESTful APIs.
* **Key Tasks:**
  1. **Server Setup:**
     + Initialize a **Node.js + Express.js** project.
     + Connect to a **MongoDB** database. Design schemas for storing user data (anonymized), feature vectors, and generated tokens.
  2. **API Endpoint Development:**
     + **User Management:** Create endpoints for user interactions, ensuring user identities are hashed.
     + **Scoring API:** An endpoint that receives a user ID, retrieves their feature vector, and calls your Python ML service to get a risk score and explanation.
     + **Tokenization Layer:** An endpoint to issue composite tokens (e.g., { "token": "Disciplined\_Spender", "value": 88, ... }) after a score is generated.
     + **Verifier API:** A secure endpoint for lenders to submit a token and validate its authenticity.
     + **Data Privacy:** Implement the logic for the "Right to Opt-Out" to delete user data and tokens upon request.

**Phase 4: User Experience - Frontend Development (React.js)**

With the backend in place, you can build the user-facing interfaces.

* **Objective:** Develop two distinct web applications for borrowers and lenders.
* **Key Tasks:**
  1. **Borrower-Facing System (The "Alt-Credit Wallet"):**
     + Create a React app that functions like a digital wallet.
     + **Components to Build:**
       - A visual risk dashboard (e.g., a "credit score meter").
       - A simple loan application form.
       - A "Token Wallet" to view issued tokens and their risk flags.
       - Privacy controls for the opt-out feature.
  2. **Lender/Verifier Portal:**
     + Create a second React app for administrative/lender use.
     + **Components to Build:**
       - An admin dashboard to view and filter all borrowers.
       - A detailed view for each borrower that displays the **SHAP-based explainability chart**, showing exactly why a decision was made.
       - An interface to interact with the Verifier API to check token integrity.

**Phase 5: Finalization & Presentation**

The final stretch is about integrating all parts and preparing to showcase your work.

* **Objective:** Ensure the system works end-to-end and create a compelling presentation.
* **Key Tasks:**
  1. **Integration Testing:** Test the full flow: a user applies, the backend calls the ML model, a score is generated, a token is issued, and the lender can view the explanation and verify the token.
  2. **Containerize (Recommended):** Use **Docker** and **Docker Compose** to package your Frontend, Backend, and Python ML service. This ensures it's easy for judges to run your project with a single command.
  3. **Prepare Your Pitch:**
     + Create a presentation that starts with the problem statement (underbanked population).
     + Walk through your "End-to-End Solution Flow" diagram.
     + **Perform a live demo:** Show the borrower app and the lender portal in action. Emphasize the SHAP explainability and the privacy-preserving tokenization as your key innovations.
     + Conclude with your results, the technical stack, and future improvements.