**Detailed Build Plan for the GIQ Platform MVP**

|  |  |  |  |
| --- | --- | --- | --- |
| **Core Functionalities** | **Objective** | **Components** | **Implementation Steps** |
| **a. Data Aggregation and Integration** | Aggregate data from multiple external sources and integrate it into the GIQ platform. | - **Data Sources**: Experian, D&B, insurance companies, collection agencies. <br> - **ETL Process**: Extract, Transform, Load (ETL) pipelines for each data source. <br> - **Database Schema**: Design a schema to store the aggregated data efficiently. | - **Set Up Data Connections**: <br> - Establish secure API connections or data pipelines to each data source. <br> - Use tools like Apache NiFi or custom Python scripts for data extraction. <br> - **Data Transformation**: <br> - Standardise data formats and fields from different sources. <br> - Use ETL tools like Apache Spark or Talend to transform the data. <br> - **Data Storage**: <br> - Create a relational database schema (e.g., PostgreSQL) to store transformed data. <br> - Implement indexing and partitioning strategies to optimise query performance. |
| **b. Risk Analysis and Scoring** | Develop algorithms to analyse aggregated data and generate risk scores. | - **Risk Scoring Models**: Algorithms for scoring entities based on payment data, credit ratings, and other factors. <br> - **Data Analysis**: Use statistical and machine learning techniques to analyse trends and patterns. | - **Algorithm Development**: <br> - Define scoring criteria based on industry standards and expert input. <br> - Implement basic models using Python (e.g., scikit-learn for machine learning). <br> - **Integration with Data**: <br> - Link the scoring models to the aggregated data in the database. <br> - Ensure that scores are updated dynamically as new data comes in. <br> - **Validation and Tuning**: <br> - Test the scoring models with historical data to validate accuracy. <br> - Tune the models based on feedback and performance metrics. |
| **c. User Interface and Interaction** | Develop a user-friendly interface for accessing and interacting with data. | - **Dashboard**: A centralised view showing key metrics and insights. <br> - **Customisable Data Views**: Allow users to filter and view data in different formats (e.g., tables, graphs). <br> - **Search and Filter**: Enable users to search for entities and apply filters to data. | - **Frontend Framework**: <br> - Choose a modern frontend framework like React or Angular. <br> - Develop a responsive layout that adapts to different devices. <br> - **Dashboard Design**: <br> - Implement components for displaying key metrics and insights (e.g., charts, tables). <br> - Use libraries like D3.js or Chart.js for data visualisation. <br> - **Customisation Features**: <br> - Allow users to save and load custom views and filters. <br> - Implement dynamic query builders to generate filtered datasets. <br> - **Search and Filter Functionality**: <br> - Develop a robust search engine using Elasticsearch or similar technologies. <br> - Implement filtering options for various data fields (e.g., date range, entity type). |
| **d. Security and Access Control** | Ensure secure access and protect data. | - **User Authentication**: Secure login and user management. <br> - **Role-Based Access Control (RBAC)**: Define user roles and permissions. <br> - **Data Encryption**: Protect data in transit and at rest. | - **Authentication System**: <br> - Use OAuth2 or JWT for secure user authentication. <br> - Implement registration, login, and password management features. <br> - **Access Control**: <br> - Define roles (e.g., admin, user) and associated permissions. <br> - Implement role-based access checks throughout the application. <br> - **Data Security**: <br> - Use HTTPS for secure data transmission. <br> - Encrypt sensitive data in the database using encryption standards (e.g., AES-256). |

**2. Technical Architecture**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Objective** | **Components** | **Implementation Steps** |
| **1. System Architecture** | Define the overall architecture to support the MVP functionalities. | - **Backend**: API server to handle business logic and data processing. <br> - **Frontend**: Client application to provide user interface and interaction. <br> - **Database**: Storage solution for aggregated data and application state. <br> - **Infrastructure**: Cloud setup for hosting and scaling the application. | - **Backend Framework**: <br> - Use Node.js with Express.js or Python with Flask/Django for API development. <br> - Implement RESTful or GraphQL APIs to serve data to the frontend. <br> - **Frontend Framework**: <br> - Develop the frontend using React or Angular. <br> - Use Redux or Context API for state management. <br> - **Database**: <br> - Use PostgreSQL for structured data storage. <br> - Implement schema design and ORM (e.g., Sequelize for Node.js or SQLAlchemy for Python). <br> - **Cloud Infrastructure**: <br> - Host the application on AWS, Azure, or Google Cloud. <br> - Set up services for scalability, such as load balancers and auto-scaling groups. |
| **2. Data Flow and Integration** | Establish how data flows between components and integrate them effectively. | - **Data Ingestion Pipelines**: Mechanisms to pull data from external sources. <br> - **Data Transformation**: Processes to standardise and clean the data. <br> - **Data Storage and Retrieval**: Efficient storage and query mechanisms. | - **Data Pipelines**: <br> - Use APIs or ETL tools to fetch data from external sources. <br> - Schedule regular data pulls using cron jobs or cloud-based schedulers. <br> - **Data Transformation**: <br> - Apply cleaning and standardisation rules to incoming data. <br> - Use Pandas in Python or similar tools to transform data into the required format. <br> - **Data Storage**: <br> - Store cleaned data in a relational database. <br> - Implement indexing and optimisation for fast query retrieval. <br> - **APIs for Data Retrieval**: <br> - Develop endpoints to retrieve data based on user queries. <br> - Implement pagination, filtering, and sorting features in the API. |

**3. Detailed Implementation Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Objective** | **Tasks** | **Implementation Steps** |
| **1. Backend Development** | Build the server-side components and logic. | - **Set Up API Server**: <br> - Initialize a Node.js or Python server with necessary packages. <br> - Implement core API endpoints for data access and user management. <br> - **Data Integration**: <br> - Develop data ingestion scripts for each external data source. <br> - Set up cron jobs or event-driven triggers for regular data updates. <br> - **Risk Analysis Logic**: <br> - Implement basic algorithms for risk scoring. <br> - Integrate these algorithms with the data retrieval process. <br> - **Testing and Documentation**: <br> - Write unit tests for API endpoints and data processing logic. <br> - Document API endpoints and data models. | - **Set Up API Server**: <br> - Initialize the backend using Node.js with Express.js or Python with Flask/Django. <br> - Install and configure necessary packages and libraries. <br> - Define and implement core API endpoints for accessing data and managing users. <br> - **Data Integration**: <br> - Write scripts to ingest data from external sources. <br> - Use cron jobs or event-driven triggers to automate regular data updates. <br> - **Risk Analysis Logic**: <br> - Develop algorithms to calculate risk scores based on defined criteria. <br> - Integrate these algorithms into the API to process and retrieve data. <br> - **Testing and Documentation**: <br> - Develop unit tests for each API endpoint and data processing function. <br> - Create detailed documentation for API endpoints and data models. |
| **2. Frontend Development** | Build the client-side application and user interface. | - **Set Up Frontend Framework**: <br> - Initialize a React or Angular project. <br> - Set up routing, state management, and core components. <br> - **Develop UI Components**: <br> - Create reusable components for data display (tables, charts, etc.). <br> - Implement the dashboard and customisable data views. <br> - **Integrate with Backend**: <br> - Connect frontend components to the backend APIs. <br> - Ensure smooth data fetching and state updates. <br> - **User Interaction Features**: <br> - Develop search and filter functionalities. <br> - Implement user settings and customisation options. <br> - **Testing and Optimization**: <br> - Write unit and integration tests for UI components. <br> - Optimize performance for faster load times and smooth interactions. | - **Set Up Frontend Framework**: <br> - Initialize the frontend project using React or Angular. <br> - Configure routing for navigation and set up state management using Redux or Context API. <br> - **Develop UI Components**: <br> - Build reusable components for displaying data, such as tables and charts. <br> - Implement the main dashboard interface and customisable data views. <br> - **Integrate with Backend**: <br> - Connect frontend components to backend APIs to fetch data. <br> - Ensure consistent data flow and handle state updates effectively. <br> - **User Interaction Features**: <br> - Develop features to allow users to search and filter data. <br> - Implement settings for user preferences and customisation. <br> - **Testing and Optimization**: <br> - Create and run unit tests for individual UI components. <br> - Perform integration testing to ensure the frontend and backend work together seamlessly. <br> - Optimize the application for quick load times and responsive interactions. |
| **3. Security and Access Control** | Secure the application and protect user data. | - **Implement Authentication**: <br> - Set up user registration and login with OAuth2 or JWT. <br> - Secure API endpoints to require authentication for access. <br> - **Role-Based Access Control**: <br> - Define user roles and permissions. <br> - Implement middleware to enforce access controls. <br> - **Data Encryption**: <br> - Use HTTPS for all data transmission. <br> - Encrypt sensitive data in the database using industry-standard algorithms. <br> - **Security Testing**: <br> - Perform penetration testing and vulnerability assessments. <br> - Implement measures to fix identified security issues. | - **Implement Authentication**: <br> - Set up authentication using OAuth2 or JWT. <br> - Create endpoints for user registration, login, and password management. <br> - Secure sensitive routes to require authentication tokens. <br> - **Role-Based Access Control**: <br> - Define various user roles (e.g., admin, user) and their permissions. <br> - Use middleware to enforce these roles across different parts of the application. <br> - **Data Encryption**: <br> - Implement HTTPS to secure data transmissions. <br> - Encrypt sensitive data stored in the database using algorithms like AES-256. <br> - **Security Testing**: <br> - Conduct penetration tests to find vulnerabilities. <br> - Address and fix any security issues identified during testing. |
| **4. Deployment and Monitoring** | Deploy the MVP to a live environment and monitor its performance. | - **Set Up Hosting**: <br> - Choose a cloud provider and set up hosting infrastructure. <br> - Configure load balancers, auto-scaling, and other cloud services. <br> - **Continuous Integration/Continuous Deployment (CI/CD)**: <br> - Set up CI/CD pipelines for automated testing and deployment. <br> - Use tools like Jenkins, GitHub Actions, or GitLab CI/CD. <br> - **Monitoring and Logging**: <br> - Implement monitoring tools to track application performance. <br> - Set up logging for error tracking and debugging. <br> - **User Feedback Collection**: <br> - Use analytics tools to gather user interaction data. <br> - Implement feedback forms or surveys to collect user input. | - **Set Up Hosting**: <br> - Select a cloud provider (AWS, Azure, or Google Cloud). <br> - Configure necessary infrastructure components such as virtual machines, databases, and networking. <br> - Set up load balancers and auto-scaling groups to manage traffic and scaling. <br> - **CI/CD Pipelines**: <br> - Configure CI/CD tools like Jenkins, GitHub Actions, or GitLab CI/CD to automate the build, test, and deployment processes. <br> - Ensure automated tests are run before deployment to catch issues early. <br> - **Monitoring and Logging**: <br> - Implement tools like Prometheus, Grafana, or CloudWatch to monitor performance and health metrics. <br> - Set up logging solutions (e.g., ELK Stack, Loggly) to track and debug errors. <br> - **User Feedback Collection**: <br> - Integrate analytics tools to capture user behavior and interaction data. <br> - Create feedback mechanisms such as forms or surveys to gather user insights. |

**4. Deliverables and Milestones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Objective** | **Deliverables** | **Tasks** |
| **1. Planning and Design Completion** | Deliver detailed project plan and design prototypes. | - Detailed project plan. <br> - Design prototypes for the user interface and system architecture. | - Develop a comprehensive project plan outlining scope, objectives, timeline, and resources. <br> - Create design prototypes for review. <br> - Review and approve designs with stakeholders to ensure alignment with project goals. |
| **2. Backend Development Completion** | Deliver working backend with data integration and risk analysis logic. | - Functional backend system. <br> - Integrated data ingestion scripts. <br> - Implemented risk analysis logic. <br> - Tested APIs and documented endpoints. | - Set up the backend server and implement core API endpoints. <br> - Develop and integrate data ingestion scripts. <br> - Implement algorithms for risk scoring. <br> - Test API endpoints for functionality and reliability. <br> - Document API endpoints and data models. |
| **3. Frontend Development Completion** | Deliver functional frontend with key UI components and customisation features. | - Functional frontend application. <br> - Key UI components (dashboard, custom views). <br> - Integrated frontend with backend. | - Initialize frontend project and set up routing and state management. <br> - Develop UI components for data display and user interaction. <br> - Integrate frontend components with backend APIs. <br> - Test UI components for usability and functionality. |
| **4. Security and Access Control Implementation** | Deliver secure user authentication and access control features. | - User authentication system. <br> - Role-based access control (RBAC). <br> - Encrypted data transmission and storage. | - Implement user authentication using OAuth2 or JWT. <br> - Define and implement RBAC with roles and permissions. <br> - Secure all data transmissions using HTTPS. <br> - Encrypt sensitive data stored in the database. <br> - Perform security testing to identify vulnerabilities. <br> - Fix any vulnerabilities found during testing. |
| **5. MVP Deployment** | Deploy the MVP to a live environment. | - Live MVP platform. <br> - Performance monitoring setup. <br> - Initial user feedback collected. | - Choose and set up hosting infrastructure on a cloud provider. <br> - Deploy the MVP to a live environment. <br> - Configure monitoring tools to track performance and health metrics. <br> - Gather initial user feedback for further improvements. |
| **6. Post-Launch Iteration** | Implement iterative improvements based on user feedback. | - Iterative updates to the MVP based on feedback. <br> - Plan for next phase or scaling. | - Collect detailed user feedback through analytics and direct surveys. <br> - Implement improvements and new features based on user feedback. <br> - Prepare a plan for the next phase of development or scaling up the platform. |

**More Detail:**

**Milestone 1: Planning and Design Completion**

|  |  |
| --- | --- |
| **Objective** | **Deliver detailed project plan and design prototypes.** |
| **Deliverables** | - Detailed project plan. <br> - Design prototypes for the user interface and system architecture. |
| **Tasks** | - **Develop Project Plan**: <br> - Define scope, objectives, timeline, and resources. <br> - Identify project risks and mitigation strategies. <br> - **Create Design Prototypes**: <br> - Develop wireframes and mockups for key UI components (dashboard, data views). <br> - Design system architecture diagrams showing data flow, backend, and frontend integration. <br> - **Stakeholder Review**: <br> - Present project plan and design prototypes to stakeholders. <br> - Gather feedback and make necessary adjustments. |
| **Timeline** | 2-3 weeks |
| **Tools** | - Project Management: Jira, Trello. <br> - Design Tools: Figma, Adobe XD. <br> - Documentation: Confluence, Google Docs. |

**Milestone 2: Backend Development Completion**

|  |  |
| --- | --- |
| **Objective** | **Deliver working backend with data integration and risk analysis logic.** |
| **Deliverables** | - Functional backend system. <br> - Integrated data ingestion scripts. <br> - Implemented risk analysis logic. <br> - Tested APIs and documented endpoints. |
| **Tasks** | - **Set Up Backend Server**: <br> - Initialize backend environment using Node.js/Express or Python/Flask. <br> - Configure development tools and frameworks. <br> - **Implement Core API Endpoints**: <br> - Develop API endpoints for data access and user management. <br> - Document API specifications and test endpoints with Postman. <br> - **Data Ingestion Scripts**: <br> - Write scripts to connect and extract data from external sources (Experian, D&B, etc.). <br> - Use tools like Apache NiFi for automating data extraction. <br> - **Risk Analysis Logic**: <br> - Implement basic algorithms for risk scoring using Python. <br> - Integrate risk scoring algorithms with data retrieval processes. <br> - **Testing and Documentation**: <br> - Write unit tests for API endpoints and data processing logic. <br> - Document API endpoints, data models, and risk scoring logic. |
| **Timeline** | 8-10 weeks |
| **Tools** | - Backend Frameworks: Node.js/Express, Python/Flask. <br> - Data Tools: Apache NiFi, Python scripts. <br> - Testing: Postman, Jest (Node.js), Pytest (Python). <br> - Documentation: Swagger, Postman documentation. |

**Milestone 3: Frontend Development Completion**

|  |  |
| --- | --- |
| **Objective** | **Deliver functional frontend with key UI components and customisation features.** |
| **Deliverables** | - Functional frontend application. <br> - Key UI components (dashboard, custom views). <br> - Integrated frontend with backend. |
| **Tasks** | - **Initialize Frontend Project**: <br> - Set up project using React or Angular. <br> - Configure routing and state management (Redux or Context API). <br> - **Develop UI Components**: <br> - Build reusable components for data display (tables, charts). <br> - Implement the main dashboard interface and customisable data views. <br> - **Integrate with Backend**: <br> - Connect frontend components to backend APIs to fetch and display data. <br> - Handle state management and data updates efficiently. <br> - **User Interaction Features**: <br> - Develop features for search, filtering, and user customisation. <br> - Implement user settings and preferences management. <br> - **Testing and Optimization**: <br> - Perform unit and integration testing of UI components. <br> - Optimize for performance and smooth user experience. |
| **Timeline** | 8-10 weeks |
| **Tools** | - Frontend Frameworks: React, Angular. <br> - State Management: Redux, Context API. <br> - UI Libraries: D3.js, Chart.js. <br> - Testing: Jest, Cypress. <br> - Design: Figma, Adobe XD. |

**Milestone 4: Security and Access Control Implementation**

|  |  |
| --- | --- |
| **Objective** | **Deliver secure user authentication and access control features.** |
| **Deliverables** | - User authentication system. <br> - Role-based access control (RBAC). <br> - Encrypted data transmission and storage. |
| **Tasks** | - **Implement Authentication**: <br> - Set up user registration and login using OAuth2 or JWT. <br> - Secure API endpoints to require authentication tokens. <br> - **Role-Based Access Control**: <br> - Define roles and permissions for different user types. <br> - Implement middleware to enforce access controls based on roles. <br> - **Data Encryption**: <br> - Use HTTPS for securing all data transmissions. <br> - Encrypt sensitive data stored in the database using industry-standard algorithms (e.g., AES-256). <br> - **Security Testing**: <br> - Perform penetration testing to identify vulnerabilities. <br> - Use tools like OWASP ZAP for automated security assessments. <br> - Fix identified security issues and re-test. |
| **Timeline** | 2-3 weeks |
| **Tools** | - Authentication: OAuth2, JWT. <br> - Security Testing: OWASP ZAP, Burp Suite. <br> - Encryption: SSL/TLS for HTTPS, database encryption. <br> - Middleware: Express middleware (Node.js), Flask extensions (Python). |

**Milestone 5: MVP Deployment**

|  |  |
| --- | --- |
| **Objective** | **Deploy the MVP to a live environment.** |
| **Deliverables** | - Live MVP platform. <br> - Performance monitoring setup. <br> - Initial user feedback collected. |
| **Tasks** | - **Set Up Hosting Infrastructure**: <br> - Choose a cloud provider (AWS, Azure, Google Cloud) and set up necessary infrastructure. <br> - Configure load balancers, auto-scaling groups, and virtual machines. <br> - **Deploy MVP**: <br> - Deploy the application to the live environment. <br> - Ensure all components (backend, frontend, database) are correctly integrated. <br> - **Performance Monitoring**: <br> - Set up monitoring tools to track application performance and health (e.g., Prometheus, Grafana). <br> - Establish logging for error tracking and debugging (e.g., ELK Stack, Loggly). <br> - **User Feedback Collection**: <br> - Use analytics tools to gather user interaction data. <br> - Implement feedback forms or surveys to collect user input on the MVP. |
| **Timeline** | 2-3 weeks |
| **Tools** | - Cloud Providers: AWS, Azure, Google Cloud. <br> - Monitoring: Prometheus, Grafana. <br> - Logging: ELK Stack, Loggly. <br> - Analytics: Google Analytics, Mixpanel. |

**Milestone 6: Post-Launch Iteration**

|  |  |
| --- | --- |
| **Objective** | **Implement iterative improvements based on user feedback.** |
| **Deliverables** | - Iterative updates to the MVP based on feedback. <br> - Plan for next phase or scaling. |
| **Tasks** | - **Collect User Feedback**: <br> - Use analytics and direct surveys to gather detailed feedback from users. <br> - Identify common issues, requested features, and areas for improvement. <br> - **Implement Improvements**: <br> - Prioritise and implement updates and new features based on feedback. <br> - Continue to test and validate changes to ensure they meet user needs. <br> - **Prepare for Next Phase**: <br> - Develop a plan for the next phase of development or scaling up the platform. <br> - Consider enhancements, new features, or broader deployment strategies. |
| **Timeline** | Ongoing |
| **Tools** | - Feedback Collection: Surveys, User Interviews. <br> - Analytics: Google Analytics, Mixpanel. <br> - Planning: Jira, Confluence. |