Project Report on

Micro-Task and Earning Platform

INDEX:

1. Introduction:

The Micro-Task and Earning Platform is a web-based application that enables users to earn virtual currency (coins) by completing small online tasks such as reviewing websites, liking content, or following social media pages. The platform is designed to support three types of users: Workers, who complete tasks for rewards; Task-Creators, who publish tasks and pay coins for their completion; and Admins, who oversee the functionality and integrity of the platform.

This application addresses the growing demand for flexible digital earning opportunities by providing a secure, scalable, and easy-to-use interface where tasks can be created, completed, verified, and rewarded seamlessly. Users can purchase coins via Stripe payment integration, and Workers can request withdrawals through local payment services such as Bkash, Rocket, and Nagad.

The platform has been developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), ensuring robust performance and a modern development framework. Authentication is handled using Firebase and JSON Web Tokens (JWT) to ensure secure access across user roles. Additional integrations include ImageBB for image hosting and Stripe for payment processing.

This report documents the project’s planning, design decisions, implementation details, and potential enhancements to guide future development and maintenance.

2. Objectives:

The main objectives of the Micro-Task and Earning Platform are as follows:

- To design a secure and scalable platform that supports seamless interaction among Workers, Task-Creators, and Admins.
- To allow users to easily register and authenticate via Firebase and JWT, ensuring secure access based on user roles.
- To enable Task-Creators to efficiently create, manage, and fund micro-tasks using a dedicated dashboard.
- To implement a Worker dashboard where tasks can be accessed, completed, and submitted with tracking of submission status.
- To facilitate the submission approval process, allowing Task-Creators to validate and reward task completions.
- To implement a coin-based incentive system with integrated Stripe payments for purchasing and managing coin balances.
- To support withdrawal mechanisms via regional mobile financial services (Bkash, Rocket, Nagad) for Workers to claim earnings.
- To equip Admins with tools to monitor platform activity, manage users, and process payments and withdrawals.
- To ensure a responsive, intuitive, and accessible user interface across mobile, tablet, and desktop platforms.
- To maintain data integrity and scalability using a normalized MongoDB schema with efficient querying and indexing.

3. System Architecture:

The system architecture is based on the MERN stack and is structured into three layers: Client Layer, Application Layer, and Data Layer. It leverages modern web technologies to ensure modularity, scalability, and performance.

3.1. Client Layer (Frontend):

- Developed using React.js with React Router for navigation and Axios for HTTP requests.
- Provides dedicated dashboards for Workers, Task-Creators, and Admins.
- Utilizes Tailwind CSS for responsive, mobile-friendly design.
- Communicates with the backend via RESTful API calls.
- Application Layer (Backend):
- Built with Node.js and Express.js.
- Implements REST API endpoints for authentication, task management, submission handling, payment processing, and withdrawals.
- Middleware is used to enforce JWT-based role-based access control.
- Integrates with external APIs for services such as image uploading (ImageBB) and payments (Stripe).

3.2. Authentication:

- Uses Firebase Authentication for initial user registration and login.

- JWT tokens are issued post-login and used to maintain secure sessions.
- Role-based access control (RBAC) ensures each user type (Worker, Task-Creator, Admin) can only access authorized resources.

3.3. Data Layer (Database):

- MongoDB serves as the NoSQL database, storing all key collections: Users, Tasks, Submissions, Payments, Withdrawals, and Notifications.
- Collections are fully normalized to 3NF to eliminate redundancy and ensure data integrity.
- MongoDB indexing and schema design are optimized for read/write performance across user roles.

3.4. Third-party Integrations:

- ImageBB API is used for uploading and storing images (e.g., user profile images, task proofs).
- Stripe API handles secure online payments for purchasing coin balances.
- This architectural design ensures scalability, modular development, and seamless user experience across all supported functionalities.

4. User Roles and Functionalities:

4.1 Worker:

Workers are end-users who complete micro-tasks in exchange for coins, which can later be withdrawn. Their functionalities include:

- Browsing and viewing a list of open tasks.
- Viewing task details including instructions, deadlines, and rewards.
- Submitting task proofs such as text responses or image uploads.
- Tracking submission status: pending, approved, or rejected.
- Viewing accumulated coin balance and submission history.
- Requesting withdrawals using regional mobile banking services like Bkash, Rocket, or Nagad.
- Receiving in-platform notifications regarding task updates or admin messages.

4.2 Task-Creator:

Task-Creators are registered users who create tasks for Workers. Their dashboard provides tools to:

- Create new tasks with descriptions, requirements, deadlines, and coin rewards.
- Upload related media via ImageBB and define the task category.
- Manage active and archived tasks (edit, close, or delete).
- Review incoming task submissions and approve or reject them.
- Monitor coin balance and purchase additional coins through Stripe integration.

- View payment history, task engagement analytics, and submission success rates.

4.3 Admin:

- Admins manage the entire platform, ensuring fair use and resolving conflicts. Their panel includes:
- Access to all user accounts for oversight and role changes (e.g., promote or ban users).
- Complete visibility into all tasks, submissions, and financial transactions.
- Tools to process or review pending withdrawal requests from Workers.
- Ability to delete inappropriate tasks or flag malicious user behavior.
- Platform analytics dashboard to monitor system usage, revenue, and task volume.
- Configuration of global settings such as default reward rates, content moderation rules, or banner messages.

5. Database Design

The platform's database is designed using MongoDB, a NoSQL document-oriented database. It features multiple collections representing different functional aspects of the platform. The schema has been fully normalized to the Third Normal Form (3NF) to eliminate redundancy and ensure data integrity.

10.1. Users Collection:

Stores information for all users including Workers, Task-Creators, and Admins.

Fields:

- userId (primary key)
- name
- email
- password
- role ("worker", "task_creator", "admin")
- profileImage (URL from ImageBB)
- coinBalance
- createdAt, updatedAt

10.2. Tasks Collection:

Represents tasks created by Task-Creators.

Fields:

- taskId (primary key)
- title
- description
- instructions
- reward (in coins)
- deadline
- creatorId (foreign key referencing Users)
- status ("open", "closed")
- createdAt, updatedAt

10.3. Submissions Collection:

Holds task submissions made by Workers.

Fields:

- submissionId (primary key)
- taskId (foreign key referencing Tasks)
- workerId (foreign key referencing Users)
- submissionText / proofImage
- status ("pending", "approved", "rejected")
- submittedAt, reviewedAt

10.4. Payments Collection:

Tracks coin purchases made by Task-Creators.

Fields:

- paymentId (primary key)
- taskCreatorId (foreign key referencing Users)
- amount
- paymentMethod (e.g., "Stripe")
- transactionId
- status ("completed", "failed")
- paidAt

10.5. Withdrawals Collection:

Records withdrawal requests submitted by Workers.

Fields:

- withdrawalId (primary key)
- workerId (foreign key referencing Users)
- amount
- method ("Bkash", "Rocket", "Nagad")
- status ("pending", "approved", "rejected")

- requestedAt, processedAt

10.6. Notifications Collection:

Handles user notifications for task updates, approvals, and system messages.

Fields:

- notificationId (primary key)
- userId (foreign key referencing Users)
- message
- type ("submission", "system", "payment")
- isRead (boolean)
- createdAt

Indexes are applied on key fields such as userId, taskId, and status to optimize querying. Relationships between collections are managed using references (foreign keys) and enforced via application logic.

6. Normalization

Database normalization was performed in three stages to ensure data consistency, minimize redundancy, and improve overall structure. The steps are as follows:

Unnormalized Form (UNF):

In the initial unnormalized state, data is stored in a flat structure where a single record may contain repeating groups and multivalued attributes.

Example: A user may have multiple task submissions and their details stored in the same row, violating atomicity.

First Normal Form (1NF):

- Repeating groups were eliminated.
- All values are atomic, meaning each field contains only one value.
- Data was split into multiple rows as necessary to represent each task submission uniquely.
- Example: Each submission by a Worker is recorded in a separate row rather than a nested list.

Second Normal Form (2NF):

- Partial dependencies were removed.
- Composite keys, if used, no longer determine non-key attributes partially.
- Tables were decomposed to ensure every non-key attribute depends on the entire primary key.

Example: User details are stored in a separate Users table, and only the user ID is referenced in the Submissions or Tasks tables.

Third Normal Form (3NF):

- Transitive dependencies were eliminated.
- Non-key attributes do not depend on other non-key attributes.

- Example: Task title and reward values are attributes of the Tasks table and not derived from each other.
- By normalizing to 3NF, the database design avoids data anomalies during insertions, updates, and deletions, and ensures optimal performance for complex querying and data retrieval.

7. Entity Relationship Diagram

The Entity Relationship Diagram (ERD) illustrates how different entities in the Micro-Task and Earning Platform are interrelated. The major entities and their relationships are as follows:

Users

- Users can have one of three roles: Worker, Task-Creator, or Admin.
- A single user (Task-Creator) can create multiple Tasks.
- A user (Worker) can make multiple Submissions.

- A user (Worker) can submit multiple Withdrawals.
- A user (Task-Creator) can have multiple Payments.
- Each user can receive multiple Notifications.

Tasks:

- Each Task is created by one Task-Creator (User).
- A Task can receive multiple Submissions from different Workers.

Submissions:

- Each Submission references one Task and one Worker (User).
- Submission includes proof (text or image), and status (pending, approved, rejected).

Payments:

- Each Payment references one Task-Creator (User).
- Contains transaction details and status.

Withdrawals:

- Each Withdrawal references one Worker (User).
- Includes withdrawal method, amount, and status.

Notifications:

- Each Notification is targeted to a User.
- Includes message, type, read status, and timestamp.

Relationship Summary:

One-to-Many: Users → Tasks, Users → Submissions, Users → Payments, Users → Withdrawals, Users → Notifications

One-to-Many: Tasks → Submissions

This logical structure ensures that all entities are normalized and related efficiently for querying and scalability. The ERD forms the blueprint for the physical database schema implemented in MongoDB.

8. Challenges

During the development of the Micro-Task and Earning Platform, several technical and operational challenges were encountered:

- Role-Based Access Enforcement: Ensuring each user role only has access to authorized features required the implementation of robust middleware and testing of user flows.
- Secure Image Handling: Integrating with ImageBB and managing secure image uploads without exposing malicious content or vulnerabilities posed a challenge.
- Reliable Payment Integration: Handling payments via Stripe and supporting withdrawal methods like Bkash, Rocket, and Nagad required careful configuration to manage errors and ensure accurate financial tracking.
- Data Consistency: Maintaining consistent states across tasks, submissions, and notifications required precise data modeling and transactional logic.
- Scalability and Performance: Optimizing MongoDB indexes and API response times was crucial as the platform was built to scale.
- Mobile Responsiveness: Ensuring the UI/UX remained consistent across various devices required rigorous front-end testing and adaptive design.

9. Future Enhancements

Several enhancements are proposed to extend the platform's capabilities and improve the user experience:

- Email Notification System: Implementing automated email alerts for submission approvals, coin balances, and system messages.
- Task Search and Filtering: Enhancing the task browsing experience with advanced filters and keyword-based search functionality.
- Dispute and Reporting System: Allowing users to report invalid or abusive tasks and submissions for admin review.
- Gamification: Introducing badges, leaderboards, or milestone rewards to increase user engagement.
- Admin Analytics Dashboard: Providing advanced analytics to monitor user trends, task completion rates, and financial flows.
- Multi-language Support: Making the platform accessible to non-English speaking users to expand its reach.
- Scheduled Tasks and Reminders: Allowing creators to schedule task publishing and set reminders for deadlines.

10. Conclusion

The Micro-Task and Earning Platform effectively addresses the need for a streamlined, secure, and user-friendly solution for managing micro-tasks and earnings. By leveraging the MERN stack, Firebase, and external APIs, the system delivers seamless task creation, submission, review, and reward processes.

Its modular architecture, responsive interface, and robust role-based access control ensure reliability and scalability. With the proposed future enhancements, the platform is well-positioned for continuous growth and broader adoption. The development process also provided valuable insights into full-stack web development, secure API integration, and user-centric design.

The platform meets its functional goals by providing an engaging and secure environment for micro-task transactions. Built using the MERN stack and modern APIs, it is designed to be scalable and user-friendly.