IS4302 Group 6

GitHub Repo: https://github.com/hanhan4/IS4302_TaskMate

TaskMate

Name	Matric Number
Andrew Ang	A0252801N
Aravindh Rajagopalan	A0295227X
Gopika Sarasvathi Kothandaraman	A0274980R
Kwek Jie Han	A0234221X
Sylviya Alexander	A0276584M

Table of Contents

1. Business Case

With the number of households using smart home devices projected to exceed 785 million by 2028 (Statista, 2023), the rapid adoption of connected devices has brought unprecedented convenience. However, it also presents significant challenges in securing and managing access to these devices. Traditional access control systems often rely on centralised frameworks or third-party intermediaries, which can raise privacy concerns, create inefficiencies, and limit user autonomy.

This project harnesses blockchain technology to establish a decentralised access management system, empowering users to securely and transparently share access to their smart devices—such as cameras or irrigation systems—with trusted individuals. Drawing inspiration from self-sovereign identity (SSI) principles, the solution ensures users retain full control over their data and interactions while eliminating dependency on intermediaries (Weigl et al., 2023).

We aim to develop a decentralised application that revolutionises house-sitting arrangements by enabling secure, automated, and transparent device access. Homeowners can grant temporary access to their smart devices, while caretakers are rewarded with tokens for completing assigned tasks as per the agreed schedule. This approach enhances trust and security, as well as fosters a seamless user experience.

1.1 Background

The exponential growth of Internet of Things (IoT) devices has transformed homes into interconnected hubs of automation. These devices, offering unparalleled convenience and efficiency, are increasingly being integrated into residences to create smart living environments. (*Top 9 IoT Device Management Platforms in 2024*, 2024) However, most IoT systems currently rely on centralised platforms for managing device access, often requiring users to interact through third-party intermediaries or cloud-based systems (Vardakis et al., 2024).

1.2 Problem Statement

Current approaches to IoT device access management face several critical challenges that undermine their security, efficiency, and scalability. While centralised systems offer convenience, they come with significant limitations. The top one among these is their vulnerability to cyberattacks. Centralised servers act as single points of failure, making them attractive targets for hackers. This exposes sensitive devices, such as security cameras or smart locks, to unauthorised access, jeopardising user safety and privacy.

Another issue is the lack of user autonomy over data and access permissions. In centralised models, third-party platforms control device access, leaving users with minimal say in how their devices are managed. Temporary access scenarios, such as granting a neighbour permission to monitor a security camera or water plants, further highlight the inefficiencies of traditional methods. Sharing passwords or physical keys is not only cumbersome but also poses significant security risks, increasing the likelihood of misuse.

Additionally, centralised systems make it difficult to revoke or modify permissions efficiently, leading to a frustrating and inflexible user experience. These systems also lack transparency, offering little visibility into **who** accessed a device and when. This lack of accountability erodes trust and raises concerns about the long-term reliability of such solutions.

As IoT ecosystems grow more complex, these limitations underscore the urgent need for a secure, transparent, and user-centric solution. Addressing these challenges will be pivotal in ensuring that IoT systems can scale effectively while maintaining trust, security, and usability.

2. Proposed Solution

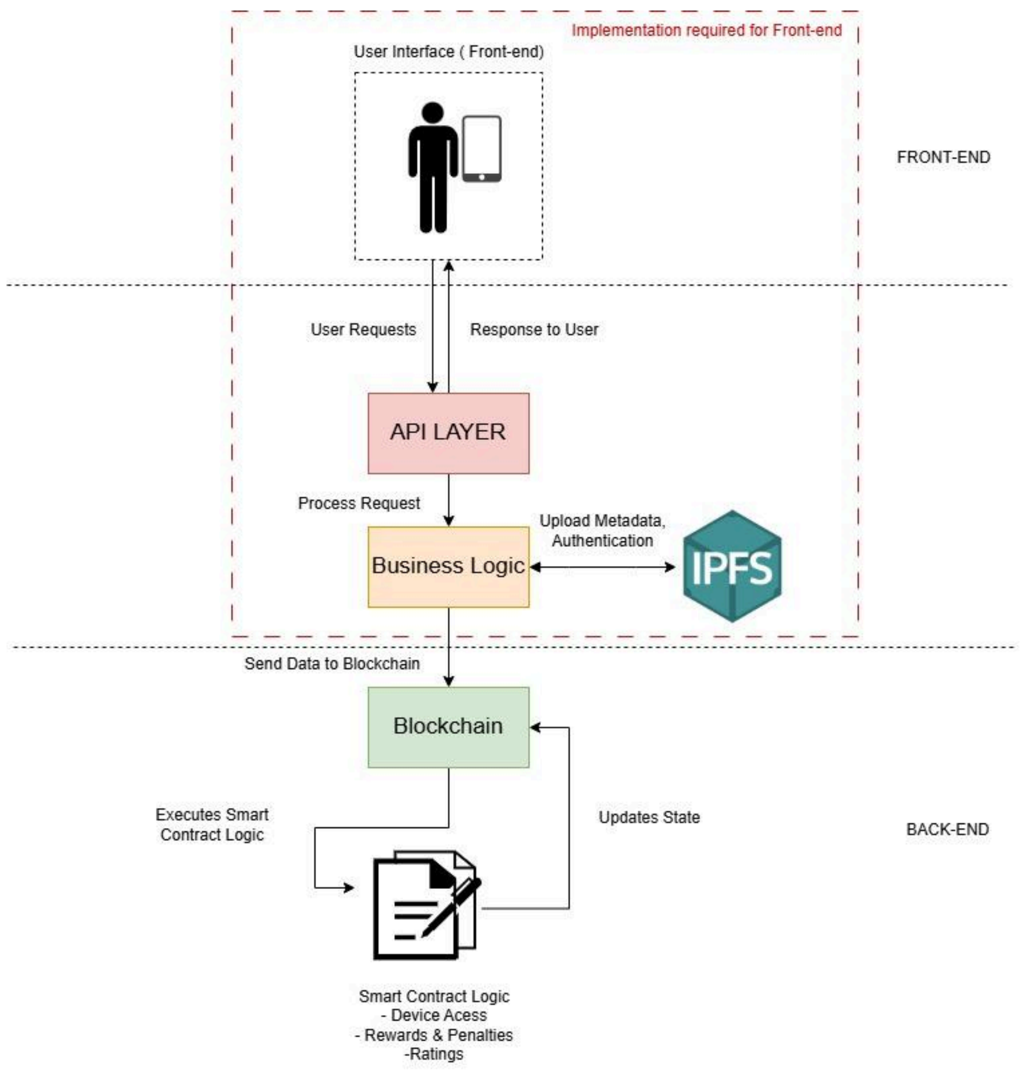
The proposed solution is a decentralised access management system that empowers homeowners to securely share access to their smart home devices with trusted individuals, such as neighbours, without relying on third-party intermediaries or centralised control. By leveraging blockchain technology, the system offers a secure, transparent, and tamper-proof method for managing permissions, ensuring homeowners retain full control over their devices. This can be accomplished through the following:

1. Dynamic Permission Management: Homeowners can grant or revoke access permissions as needed. Temporary access can be assigned for specific tasks, such as watering plants or monitoring a security camera, ensuring flexibility and security.
2. Task-Based Access: Homeowners can create task requests, and caretakers can accept assignments to gain access. This ensures that access is purpose-driven and time-limited, reducing the risk of unauthorised use.
3. Incentive Mechanism: Upon completing tasks, caretakers are rewarded with blockchain-based tokens, motivating them to fulfil responsibilities efficiently. This creates a fair and transparent reward system.
4. Reputation Management and Penalties: The system incorporates a reputation management mechanism to encourage accountability and discourage misuse. Penalties for non-compliance or misuse further reinforce good behaviour.
5. Privacy and Security: The decentralised framework ensures that user identities, data, and device interactions remain private and secure. Blockchain eliminates the vulnerabilities inherent in centralised platforms, such as single points of failure and external manipulation.
6. Transparent Operations: The blockchain ledger provides an immutable record of all access permissions and task completions, enabling verifiable, trust-based interactions between homeowners and caretakers.

Through the above functionalities, we have devised a decentralised, user-centric approach that addresses the inefficiencies, security risks, and lack of transparency associated with traditional IoT access management. We thus offer a scalable and robust solution that prioritises trust, privacy, and user control in an increasingly connected world.

3. Technical Implementation

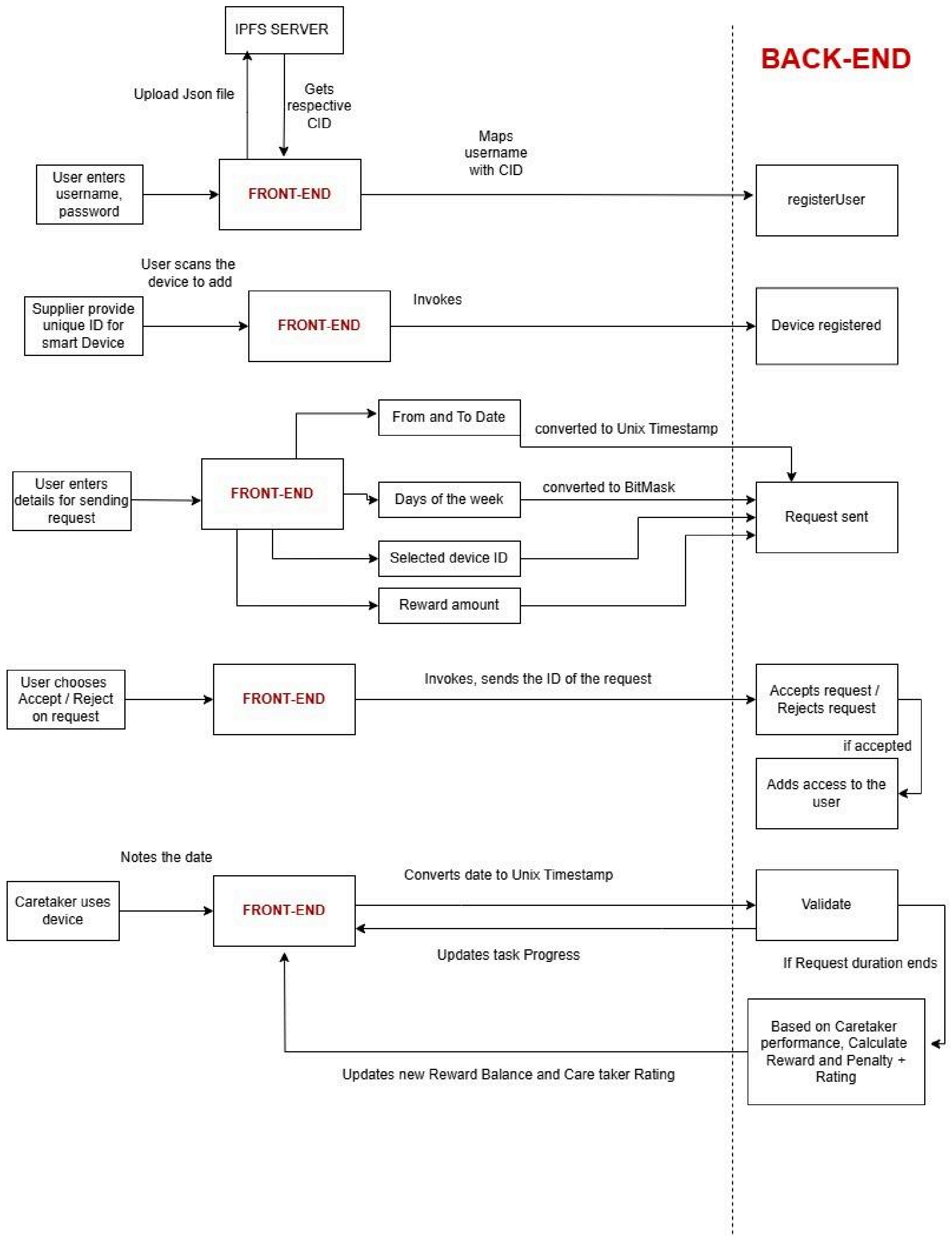
3.1 System Architecture and Flowchart



The project system architecture described in the figure above provides an overview of the system's structure, behaviour, and functionality. The flow begins from the user's point of contact, where user requests are processed in the intermediate layer (API).

The API Layer plays a major role in the system by processing the input into the desired format required by the back-end layer. For this project, all the necessary functionalities of the back end have been developed and integrated successfully.

The following flowchart diagram illustrates the system and discusses its technical implementation in detail. All interactions between the front end and back end occur through the API. For brevity, the API layer is not included in the diagram.



IPFS SERVER
BACK-END
Upload Json file
Gets respective CID
Maps username with CID
User enters username, password
FRONT-END
registerUser
User scans the device to add
Supplier provide unique ID for smart Device
FRONT-END
Invokes
Device registered
From and To Date
converted to Unix Timestamp
User enters details for sending request
FRONT-END
Days of the week
converted to BitMask
Request sent
Selected device ID
Reward amount
User chooses Accept / Reject on request
FRONT-END
Invokes, sends the ID of the request
Accepts request / Rejects request
if accepted
Adds access to the user
Notes the date
Caretaker uses device
FRONT-END
Converts date to Unix Timestamp
Validate
Updates task Progress
If Request duration ends
Based on Caretaker performance, Calculate Reward and Penalty + Rating
Updates new Reward Balance and Care taker Rating

User Registration Process

When a user registers in the DApp, all their details are stored in a JSON file, which is then uploaded to the IPFS server. For every file uploaded, a unique CID (Content Identifier) is generated. This CID is then mapped to the username and their corresponding contract address in the back end.

Each user is allotted an initial token value of 100 as part of the system incentive. This incentive is funded through the initial purchase in the DApp. In the future, if a user depletes their tokens, they will need to top up their wallet. However, this feature is not implemented in the current system.

Additionally, every user starts with a neutral rating, which may or may not change depending on their performance in handling requests or tasks they may work on later.

Device Registration Process

For device registration, the supplier provides a unique ID for the smart device, which the user scans through the front-end interface. This triggers the back-end deviceRegistered function to store and register the device details successfully to their address and name.

Sending a Request

When sending a request, the user inputs details such as their username, start and end dates, days of the week, the selected device ID, and the reward amount. These details are first processed in the API layer and then forwarded to the back end. The sendRequest function will then send the request to the specified user. The requestSent function sends the request to the specified user. To build the request, the username is encoded into an address using the getUserAddress function.

Since Solidity cannot directly handle date and day formats, they are converted into appropriate representations. Dates are encoded as Unix Timestamps. For instance, 06-Nov-2024 is represented as 1730925525. Similarly, days of the week are encoded into a BitMask. For example, if the user selects Tuesday and Friday, the value is 36 (0b0100100), where Tuesday corresponds to 4 and Friday to 32. Below is the mapping for each day of the week:

- Sunday = 1 (0b0000001)
- Monday = 2 (0b0000010)
- Tuesday = 4 (0b0000100)
- Wednesday = 8 (0b0001000)
- Thursday = 16 (0b0010000)
- Friday = 32 (0b0100000)
- Saturday = 64 (0b1000000

Further, the first date on which the task is scheduled to be performed is saved in the nextDueDate parameter within the request structure. This ensures that the system can track when the task should next be executed. Additionally, overall number of task to do is also calculated and saved in the structure.

Request Handling

The specified users can review the request and choose to accept or reject it through the front-end interface. The decision is sent to the back end, where the request is processed. If accepted, the system grants the necessary access to the device for the user who accepted the request. If rejected, the request then becomes invalid.

Task Validation

Once the caretaker begins using the device, every time they use it, the date is noted. This date is then converted to a Unix timestamp and sent to the back end for validation. The validateProgress function is responsible for comparing the logged date with the nextDueDate of the task. If the two dates match, it indicates that the caretaker performed the task on time. As part of the validation process, the number of tasks left is decremented, and the next due date is calculated for subsequent validations, if applicable.

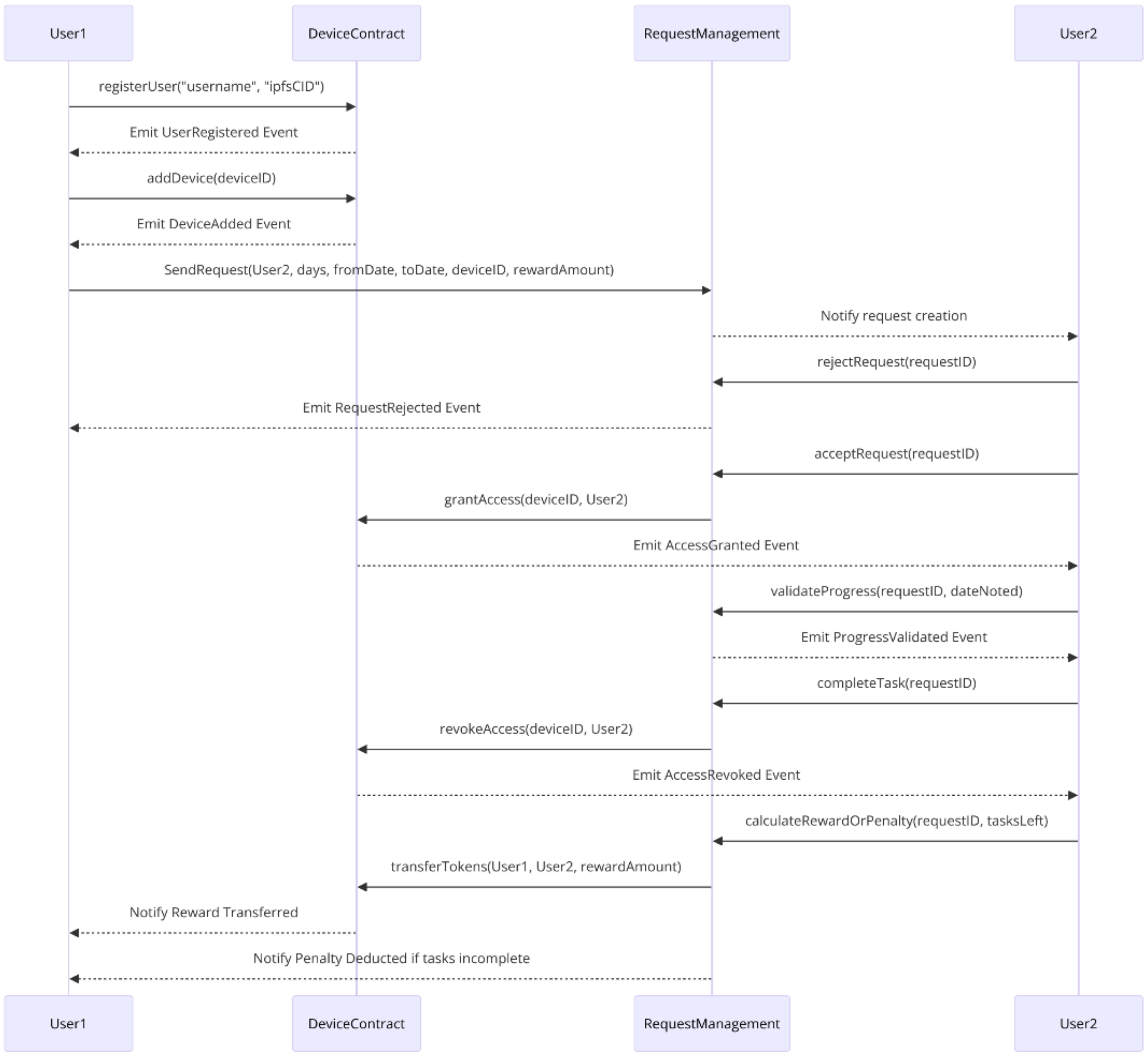
However, if the number of tasks left reaches 0, it indicates that the request has been completed successfully, meaning the user has performed all the tasks within the request. On the other hand, if the end date is reached but the tasks left are not 0, it suggests that the caretaker may have skipped at least one task during the process, indicating an incomplete or missed task. This validation mechanism ensures that the task is tracked accurately and that caretakers are rewarded or penalized based on their performance and adherence to the schedule.

Reward and Penalty Calculation

Based on the caretaker's performance, the system calculates rewards, applies penalties if necessary, and updates the caretaker's rating. The following rules are applied depending on the completion percentage of the assigned tasks:

- **100% Completion**: If the caretaker successfully completes all the tasks as required, they receive the full reward.
- **> 70% Completion**: If the caretaker completes more than 70% but less than 100% of the tasks, they will receive **half the reward**. Additionally, the requester is refunded the remaining as compensation for the incomplete tasks.
- **≤ 70% Completion**: If the caretaker completes 70% or fewer of the tasks, requester is refunded fully. The strict condition ensures caretaker accountability. However, in the future, a chat feature could be added, allowing caretakers to communicate with requesters. If the caretaker provides a valid reason for not completing the task, the requester could choose to exempt them from the penalty.

The following sequence diagram illustrates the interaction flow between the front-end and back-end systems for handling requests, task progress, and validation. It visualizes the step-by-step process of user registration, device management, request creation, and task validation, showing how different components communicate to ensure the smooth execution of tasks and reward calculations.


User1
DeviceContract
RequestManagement
User2
registerUser("username", "ipfsCID")
Emit UserRegistered Event
addDevice(deviceID)
Emit DeviceAdded Event
SendRequest(User2, days, fromDate, toDate, deviceID, rewardAmount)
Notify request creation
rejectRequest(requestID)
Emit RequestRejected Event
acceptRequest(requestID)
grantAccess(deviceID, User2)
Emit AccessGranted Event
validateProgress(requestID, dateNoted)
Emit ProgressValidated Event
completeTask(requestID)
revokeAccess(deviceID, User2)
Emit AccessRevoked Event
calculateRewardOrPenalty(requestID, tasksLeft)
transferTokens(User1, User2, rewardAmount)
Notify Reward Transferred
Notify Penalty Deducted if tasks incomplete
User1
DeviceContract
RequestManagement
User2

3.2 Modules and Functionality

This section outlines the key functions in the system, discussing their purpose in the system.

Function: registerUser

Functionality:

- Registers a user by their username and IPFS CID.
- The function ensures that the user does not already exist (by checking if their username is already mapped to an IPFS CID).
- It then stores the provided IPFS CID for the given username.

Parameters:

- string memory _username: The username of the user to be registered.
- string memory _ipfsCID: The IPFS CID associated with the user's data.

Events Emitted:

- UserRegistered: Indicates that a new user has been successfully registered.

Function: getUserCID

Functionality:

- Retrieves the IPFS CID of a user based on their username.

Parameters:

- string memory _username: The username of the user whose IPFS CID is to be retrieved.

Returns:

- string memory: The IPFS CID associated with the user.

Function: addDevice

Functionality:

- Adds a new device to the system by assigning it to the caller (device owner).
- The function ensures that the device ID does not already exist and adds the device to the caller's list of devices.

Parameters:

- uint256 deviceID: The ID of the device to be added.

Returns:

- uint256: The ID of the device that was added.

Events Emitted:

DeviceAdded: Indicates that a new device has been added successfully.

Function: getDeviceOwner

Functionality:

- Retrieves the owner of a given device by its device ID.

Parameters:

- uint256 deviceID: The ID of the device whose owner is to be fetched.

Returns:

- address: The address of the owner of the device.

Function: getAccessList

Functionality:

- Retrieves the list of addresses that have access to a given device.

Parameters:

- uint256 deviceID: The ID of the device whose access list is to be retrieved.

Returns:

- address[] memory: A list of addresses that have access to the device.

Function: getDevicesByOwner

Functionality:

- Retrieves the list of device IDs that belong to the caller (the device owner).

Returns:

- uint256[] memory: A list of device IDs owned by the caller.

Function: SendRequest

Functionality:

- Sends a request from the caller (device owner) to another user (potential helper) for access to a specific device on specific days within a date range.

- The function calculates the total number of tasks based on the days of the week and the specified date range.

Parameters:

- address _toUser: The address of the user to whom the request is being sent.
- uint8 _days: A bitmask representing the days of the week (e.g., 0x7F for all days).
- uint256 _fromDate: The start date of the request (Unix timestamp).
- uint256 _toDate: The end date of the request (Unix timestamp).
- uint256 _deviceID: The ID of the device for which access is being requested.

Events Emitted:

- RequestCreated: Indicates that a new request has been created successfully.

<u>Function: calculateTotalTasks</u>

Functionality:

- Calculates the total number of tasks based on the provided date range and days of the week (using a bitmask).

Parameters:

- uint256 _fromDate: The start date (Unix timestamp).
- uint256 _toDate: The end date (Unix timestamp).
- uint8 _daysOfWeek: The bitmask representing the selected days.

Returns:

- uint256: The first date when a task is due (Unix timestamp).
- uint256: The total number of tasks within the given date range.

Function: revokeAccess

Functionality:

- Revokes access from a user on a specific device (except for the owner of the device).
- The function checks if the user is on the device's access list and removes them if found.

Parameters:

- uint256 deviceID: The ID of the device from which the access is to be revoked.
- address user: The address of the user whose access is being revoked.

Events Emitted:

- AccessRevoked: Indicates that access has been revoked from the user.

Function: rejectRequest

Functionality:

- Rejects a request from a user (recipient of the request).
- The request's status is updated to "Rejected," and it is deactivated.

Parameters:

- uint256 requestID: The ID of the request to be rejected.

Events Emitted:

- RequestRejected: Indicates that the request has been rejected.

<u>Function: acceptRequest</u>

Functionality:

- Accepts a request from a user (recipient of the request).
- If the request is valid, the user is granted access to the requested device.

Parameters:

- uint256 requestID: The ID of the request to be accepted.

Events Emitted:

- AccessGranted: Indicates that access has been granted to the user.

<u>Function: getAccess</u>

Functionality:

- Grants access to a device for a user after a request has been accepted.
- The user is added to the device's access list if they have been granted access.

Parameters:

- uint256 deviceID: The ID of the device for which access is granted.
- uint256 requestID: The ID of the request that is being processed.

Events Emitted:

- AccessGranted: Indicates that access has been granted to the user.

Function: validateProgress

Functionality:

- Validates the progress of a task based on the date it is completed.
- It checks if the task is completed on time, late, or very late and adjusts the progress accordingly.
- If all tasks are completed, a reward is given, and access to the device is revoked.

Parameters:

- uint256 requestID: The ID of the request being processed.
- uint256 dateNoted: The date when the task was completed.

Returns:

- uint256: The completion progress percentage (0 to 100).

Events Emitted:

- ProgressValidated: Indicates that progress has been validated.
- TaskCompleted: Indicates that the task has been completed.
- RewardGiven: Indicates that a reward has been given for completing the task.

Function: calculateNextDueDate

Functionality:

- Calculates the next due date based on the current due date and the selected days of the week (bitmask).
- It ensures that the task is due on a valid day as per the given schedule.

Parameters:

- uint256 currentDueDay: The current due date (Unix timestamp).
- uint8 daysBitmask: A bitmask representing the selected days.

Returns:

- uint256: The next valid due date.

Function: rewardOrPenalty

Functionality:

Determines the reward or penalty based on the completion percentage of tasks.

- **Full Completion:** If all tasks are completed, the full reward amount is given.
- **Partial Completion:** If at least 70% of tasks are completed, half of the reward amount is given.
- **No Completion:** If less than 70% of tasks are completed, the full reward amount is penalized and transferred back.

Parameters:

- uint256 requestID: The ID of the request being processed.
- uint256 _tasksLeft: The number of tasks left to complete.
- uint256 rewardAmount: The reward amount for completing the tasks.

Returns:

- void: The function does not return any value. It handles the internal state updates and token transfers.

<u>Function: updateRating</u>

Functionality:

- Updates the user's rating based on the completion of a request.
- It adjusts the user's TotalRating and recalculates their OverallRating after each task.
- Positive or negative ratings are applied depending on whether the task was completed successfully, partially, or not completed.
- The function emits a RatingUpdated event to notify the rating changes.

Parameters:

- address _username: The address of the user whose rating is being updated.
- int256 _ratingChange: The change in rating to be applied to the user's total rating. Can be positive, zero, or negative.

Returns:

- void: The function does not return any value. It updates the user's rating in the system and emits a RatingUpdated event.

<u>Function: transferTokens</u>

Functionality:

- Transfers tokens between two users, either as a reward or penalty.
- It ensures that the sender has sufficient balance before making the transfer.
- The function updates the balances of both users and emits a TokensTransferred event to notify the transaction.
- We implemented Mutex to avoid reentrancy attacks which was important here as this is the place where we pay(in a sense) and according to [5] we felt a place where send, receive and call are called is important to protect from reentrancy attacks.

Parameters:

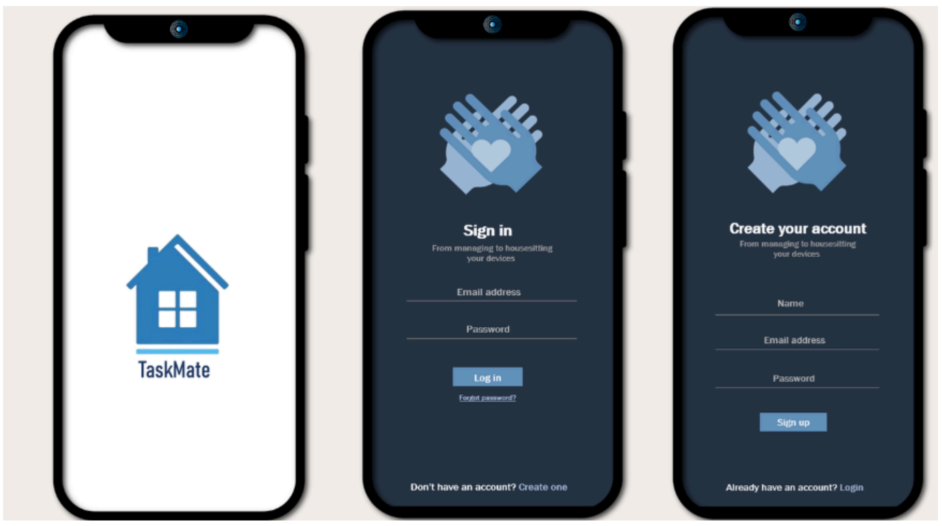
- address _from: The address of the sender.
- address _to: The address of the recipient.
- uint256 _amount: The amount of tokens to be transferred.

Returns:

- void: The function does not return any value. It performs the token transfer and emits a TokensTransferred event.

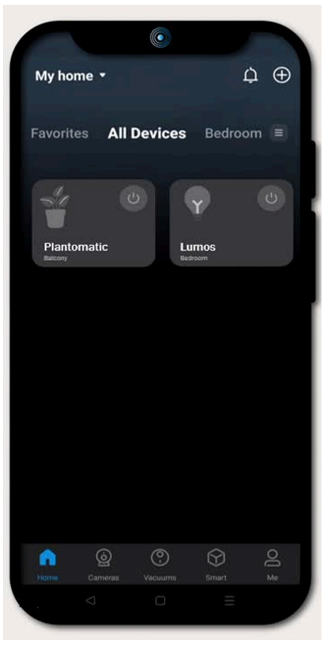
4. Front End Mockups

Basic Functionality - Login/ Sign up


TaskMate
Sign in
From managing to housesitting your devices
Email address
Password
Log in
Forgot password?
Don't have an account? Create one
Create your account
From managing to housesitting your devices
Name
Email address
Password
Sign up
Already have an account? Login

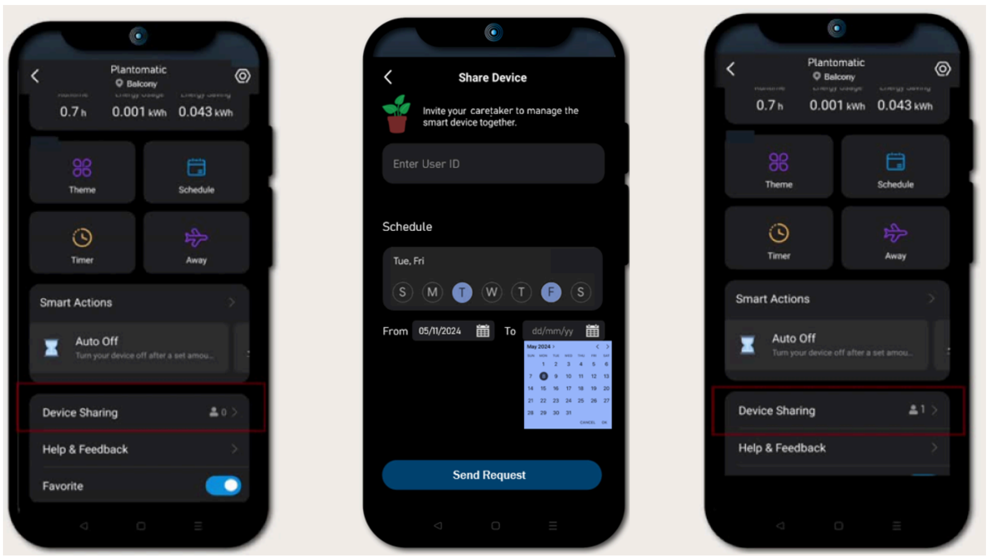
This is the User Interface of the "TaskMate" app, which includes a home screen, a sign-in page, and create account page.

Home Page - Home Owner



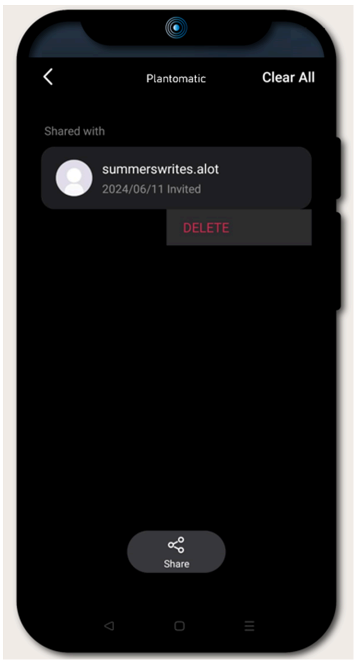
The owner can view their devices under the “All Devices” section. The interface likely allows users to interact with or control their devices.

Sending the Request - Home Owner



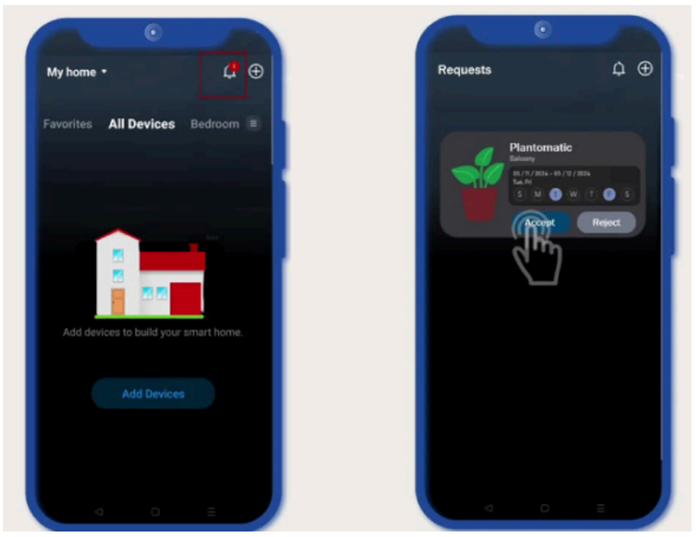
Homeowner can send the request to specific user, sharing access to their device along with the scheduled day, startDate and endDate for the task. The reward amount for the task is displayed and the request can then be sent.

Removing the Access - Home Owner



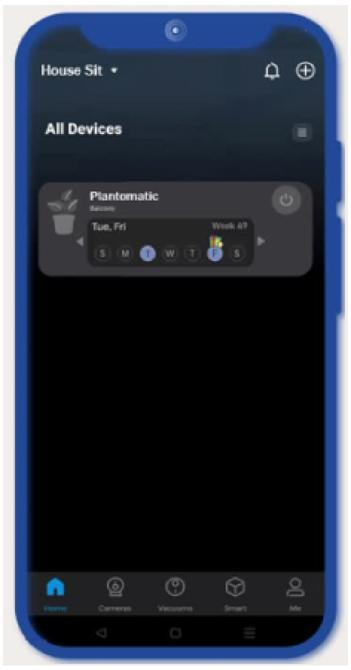
This shows the process of removing access to a device by the homeowner. The Owner can select the "delete" button and revoke their device access anytime.

Accepting the Request - CareTaker



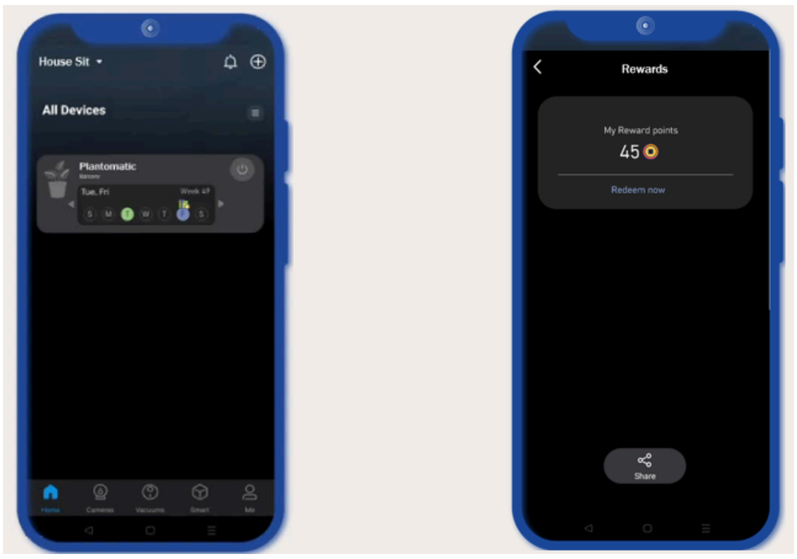
In this screen, after the owner sends a request, the caretaker receives a notification in the "Requests" section. The caretaker can then choose to either "Accept" or "Reject" the request for device access.

Task Progress - CareTaker



After accepting the request, they can track the progress of their task. The task is displayed on the device card, showing its current status along with the scheduled days for monitoring.

Task Completion and Reward - CareTaker



After completing the task within the scheduled time, the caretaker earns reward points. The points are displayed on the “Rewards” screen, and they can choose to redeem or share their rewards.

5. Difficult Areas

1. **Oracle for Task Completion Notification from Smart Devices**

Challenge: Running Oracle services add to the computational and operational costs of the system, particularly for real-time applications requiring frequent updates.

Impact: We replaced Oracles with dummy contracts to replicate the functionalities performed by smart devices.

2. **Off-Chain Data Management Using IPFS**

Challenge: Managing authentication data off-chain was essential to reduce blockchain bloat and improve scalability. However, it introduced dependency on the IPFS network for availability and reliability.

Impact: Ensuring data availability and implementing robust pinning services were critical to prevent disruptions caused by potential network unavailability.

3. **Lower Gas Costs Trade-Off**

Challenge: While storing only IPFS hashes on-chain significantly reduced gas costs, it added complexity to data management and retrieval processes.

Impact: The trade-off required careful design to balance cost-efficiency with performance and reliability.

4. **Authentication Data Privacy**

Challenge: Protecting user data privacy while offering flexible encryption and granular access control added to the system's complexity.

Impact: Encryption strategies and fine-tuned access control mechanisms were implemented to ensure data security without compromising usability.

5. **More Complex System Architecture**

Challenge: Incorporating additional network hops for authentication and ensuring interoperability with IPFS created a more complex system.

Impact: The complexity enabled improved security features and better preparation for varying regulatory requirements, but it demanded rigorous testing and optimization.

6. **Availability vs. Decentralisation**

Challenge: Relying on a decentralised IPFS network introduced potential availability issues, necessitating fallback authentication methods and redundant storage mechanisms.

Impact: To mitigate these risks, reliable pinning services were integrated, and alternative mechanisms for task validation were explored.

7. **Task Assignment and Market Competition**

Challenge: Assigning tasks directly to specific user IDs improved efficiency but risked monopolisation, especially in areas with fewer caretakers.

Impact: A balance was achieved by setting workload limits and conducting regular market price analyses to ensure fair competition and equitable participation.

8. **Geographic Coverage vs. Service Quality**

Challenge: Maintaining high service quality across underserved regions was difficult, requiring additional incentives to motivate caretakers.

Impact: Token-based rewards and reputation management mechanisms were employed to encourage equitable service distribution.

6. Lessons learned

1. **Advantages of Off-Chain Data Management**

Storing authentication data on IPFS reduced blockchain bloat, enabling sustainable scalability as the number of users and devices grows. The use of IPFS also lowered operational costs while maintaining data security.

2. **Efficiency Through Direct Task Assignments**

Assigning tasks using specific user IDs accelerated task initiation, enhanced user experience, and fostered stronger relationships between homeowners and caretakers.

3. **Balancing Complexity and Security**

Although introducing additional architectural complexity required significant effort, the improved security features and compliance flexibility were invaluable for long-term system robustness.

4. **Ensuring Privacy and Transparency**

By integrating flexible encryption and granular access control, the system successfully enhanced privacy without compromising functionality. Blockchain transparency further strengthened user trust.

5. **Incentivizing Underserved Areas**

Providing higher token rewards for caretakers in underserved regions improved service equity and incentivized participation, ensuring broader geographic coverage.

6. **Navigating Trade-Offs**

The trade-offs between decentralisation and availability highlighted the importance of fallback mechanisms, reliable network services, and user-centric designs in decentralised systems.

7. Economic Incentives

TaskMate's economic model incentivizes participation, ensures fair task distribution, and fosters trust between homeowners and caretakers. The system leverages blockchain-based tokens and a reputation management framework to drive engagement and maintain transparency.

Caretakers are rewarded with tokens for completing simple tasks such as watering plants or monitoring security systems. This token-based system motivates caretakers to provide reliable services while offering homeowners cost-effective task management.

A reputation management system links tokens to performance, rewarding consistent, high-quality work with better future opportunities. This discourages misconduct and encourages professionalism among caretakers.

Task payments are proportional to complexity and duration, allowing homeowners to attract skilled caretakers for critical tasks while keeping simpler tasks affordable. Penalties, such as reduced token rewards or reputation deductions, ensure accountability and reliability.

To address geographic disparities, the platform offers higher token rewards for caretakers in underserved areas, ensuring equitable service coverage. By eliminating third-party intermediaries, TaskMate also reduces costs for homeowners, making the system more accessible and efficient.

Tokens have long-term utility, as they can be exchanged for services within the ecosystem or converted to fiat currency. This enhances the platform's scalability, providing lasting value to users while ensuring sustainable engagement, all while ensuring the future sustainability of the platform.

8. Next Steps

To further develop and enhance the TaskMate platform, the following steps have been identified as priorities for implementation:

1. **Expanding Token Utility and Marketplace Features**

   Building a robust marketplace for token utility is a key focus. Future app expansions will incorporate advanced features such as premium services, staking for enhanced rewards, and voting rights within the platform. This will create a versatile, multi-functional currency system that incentivizes users to remain active and engaged while unlocking additional value as the ecosystem grows.

2. **Integrating Zero-Knowledge Proofs (ZKPs)**

   To enhance privacy and anonymity, the system will leverage zero-knowledge proofs. This will allow user selection and task assignment to be based on ratings rather than personal information, ensuring confidentiality while maintaining high trust. ZKP implementation will further align the platform with decentralised principles and regulatory compliance.

3. **Geographic and Market Expansion**

   TaskMate will pilot its services in underserved regions, offering higher token rewards to establish a foothold and attract initial users. Insights from these pilots will guide adjustments to the reward system and reputation mechanisms, ensuring adaptability to diverse markets.

4. **Stakeholder Education and Onboarding**

   Finally, the platform will launch an educational initiative to onboard new users and stakeholders. This includes tutorials on using the app, understanding token functionality, and navigating the marketplace. Educating users will build confidence and ensure a smooth adoption process.

By executing these steps, TaskMate aims to establish itself as a scalable, user-centric, and privacy-focused solution for decentralised smart home access management. These advancements will strengthen the platform’s utility, trustworthiness, and appeal across a global audience.

References

1. Statista. (2024, September 6). *Global: smart home number of users 2019-2028*. Statista. Retrieved November 22, 2024, from https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world
2. *Top 9 IoT Device Management Platforms in 2024*. (2024, May 8). Bytebeam. https://bytebeam.io/blog/top-iot-device-management-platforms/
3. Vardakis, G. (n.d.). *Review of Smart-Home Security Using the Internet of Things*. MDPI. Retrieved November 22, 2024, from https://doi.org/10.3390/electronics13163343
4. Weigl, L. (2023, October). *The construction of self-sovereign identity: Extending the interpretive flexibility of technology towards institutions*. Science Direct. https://doi.org/10.1016/j.giq.2023.101873
5. Shahda, W. (2022, June 24). Protect Your Solidity Smart Contracts From Reentrancy Attacks. *Medium*. https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21
6. *Epoch Converter*. (n.d.). Unix Timestamp Converter. https://www.epochconverter.com/