# CryptoKids Decentralised Application

Final Technical Report

**National College of Ireland**
Higher Diploma in Science in Computing
Blockchain Specialization
2022/2023

Project (HDAIML_SEPBL) and Blockchain Application Development (HDBC_SEPBL)
**Cross Assessment – Part Two**

**Submitted by:**
Luiz Canedo
canedobox@gmail.com

# Table of Contents

# 1. Executive Summary

Blockchain technology is rapidly evolving, revolutionizing various industries, and changing the way we transact, store, and secure digital assets. Although the use of blockchain has increased massively in the past few years, its concept and applications are still a hard subject for most people to understand. Hence why, blockchain education for children is a crucial step to equip the younger generation with the knowledge and skills needed to navigate the digital world confidently and make informed decisions about their digital assets and transactions.

Upon research, a few blockchain courses designed for children were found available online but no tools or applications were found to help children learn about blockchain technology in a practical environment. The CryptoKids project was proposed as a solution for this issue.

CryptoKids is an innovative and interactive decentralised application (Dapp) designed to provide a safe, fun, and exciting platform where children can learn about blockchain technology and its real-world applications, such as digital transactions, digital assets, and decentralization. CryptoKids aims to demystify complex concepts of blockchain technology and inspire the next generation of crypto enthusiasts and innovators while promoting financial literacy, curiosity for learning, problem-solving skills and critical thinking.

The CryptoKids Dapp is composed of two main components, the smart contract and the frontend application. The smart contract was written using Solidity and deployed to the Sepolia Ethereum Testnet to ensure a risk-free space for children to experiment, make mistakes, and learn without any real financial risks or concerns. The frontend application was implemented using JavaScript, HTML, Tailwind CSS, React.js, and Ethers.js, was deployed and hosted using Vercel, and was carefully designed to have a clean, modern, and user-friendly interface suitable for children.

Thorough testing was performed during the development process and after deployment, to ensure both the smart contract and frontend application are secure, error-free, and that they work as intended.

# 2. Introduction

## 2.1. Background

Blockchain technology is revolutionising various industries and changing the way we transact, store, and secure digital assets. It is very important to recognise that blockchain technology is not limited to adults alone and that children are an essential part of our digital future. Therefore, blockchain education for children is crucial to ensuring that the younger generation is ready for an imminent future as blockchain technology continues to develop and becomes more prevalent in our daily lives.

Although there are a few blockchain courses designed for children available on the internet, there aren't any applications that allow children to use what they've learned in a practical environment while enabling them to experiment and make mistakes without any real financial risks or concerns.

CryptoKids enable parents to introduce their children to blockchain concepts at a young age, empowering children to become active participants in the crypto world and providing them with the knowledge and skills to navigate the digital world confidently and make informed decisions about their digital assets and transactions.

## 2.2. Aims

The CryptoKids application aims to:

- Introduce children to the fascinating world of blockchain technology and cryptocurrencies in a safe, fun, and engaging manner.
- Help children demystify complex concepts of blockchain technology and its applications, such as digital transactions, digital assets, and decentralization.
- Inspire the next generation of crypto enthusiasts and innovators.
- Help children develop basic financial literacy skills, such as earning, saving, and spending.
- Help children develop problem-solving and critical thinking skills by completing tasks and challenges.
- Promote curiosity for learning by providing a fun, interactive, and gamified experience.
- Enable parents to actively participate in their children's blockchain learning journey.
- Promote collaborative learning and family engagement.

## 2.3. Benefits

- **Blockchain education:**
  CryptoKids helps to introduce children to the concept of blockchain and its real-world applications with practical activities, which provide children with a foundational understanding of blockchain technology, digital transactions, digital assets, and decentralization.

- **Gamified learning:**
  CryptoKids provides an interactive, fun, and engaging learning experience through gamification components, which motivates children to actively participate, learn, and apply their knowledge in a practical setting.

- **Skills development:**
  In addition to helping children learn about blockchain technology, CryptoKids also focuses on its real-world applications, such as decentralization, smart contracts, digital ownership, cryptocurrencies, and digital transactions. While also helping to promote financial literacy, curiosity for learning, problem-solving skills, and critical thinking.

- **Empowerment:**
  CryptoKids empowers children with knowledge and skills to navigate the crypto world confidently, enabling them to make informed decisions regarding digital assets and financial management.

- **Safe environment:**
  CryptoKids provides a safe and controlled environment for children to learn and explore blockchain technology. The application runs on the Sepolia Ethereum Testnet, ensuring a risk-free space for children to experiment, make mistakes, and learn without any real financial risks or concerns.

- **Parental involvement:**
  Parents can actively participate in their children's learning journey by assigning tasks and rewards based on the children's interests, which encourages them to actively participate and learn, while also promoting collaborative learning and family engagement.

## 2.4. How does CryptoKids work?

By completing tasks and challenges assigned by a parent, such as household chores and educational activities, children can earn tokens that can be exchanged for rewards in a virtual marketplace. This gamified experience encourages children to actively participate, learn, and apply their knowledge in a practical setting. For example:

**1)** Alice, a parent, signs up for CryptoKids Dapp and adds her child, Bob, to her family group.
**2)** Alice creates and assigns a task to Bob, in this scenario a household chore, which is “Clean your bedroom”, the task carries a reward of 10 tokens.
**3)** Bob completes the task within a given deadline and marks it as completed on the application.
**4)** Alice reviews the task, finds it well done, and approves its completion.
**5)** The smart contract then transfers 10 tokens to Bob's wallet as a reward for completing the task.
**6)** With the earned tokens, Bob visits the marketplace in the application and decides to purchase a “Fun Day out at the Zoo” reward (which was created and assigned by Alice), he exchanges 10 tokens for the rewards.
**7)** Bob redeems the reward and can now talk to Alice to plan their fun day out.

## 2.5. What is a decentralised application?

A decentralised application, also known as Dapp, is an application that runs on a decentralised peer-to-peer network, usually a blockchain. Unlike the traditional applications that are centrally controlled (by an individual or organisation), Dapps are designed to be open source, transparent, and resistant to censorship or manipulation. This is only possible because of the use of smart contracts, a Dapp’s backend, which are pieces of code that live on the blockchain and run exactly as programmed.

## 2.6. Technologies

There were a few requirements when choosing the technologies for this project:

- The technology should be widely used and provide great documentation.
- The technology should be easy to learn in a short period of time.
- All the technologies chosen should work well together.

Find below, a brief description of the technologies used and how they contributed to the project.

### 2.6.1. Tools


- **ClickUp:** is a cloud-based collaboration and project management tool. ClickUp was used to create the project's plan, manage its deliverables and track the time spent on each of them.

- **Figma:** is a cloud-based design and prototyping tool. Figma was used to design the application's user interfaces and create wireframes.

- **Visual Studio Code:** is a source code editor that supports various programming languages and provides a customizable environment for software development, an integrated terminal and a large number of extensions, which helps to optimise the development process by having access to all the tools needed in one place. Visual Studio Code was the code editor used to develop the smart contract and frontend application.

- **GitHub:** is a web-based platform for version control and collaboration. GitHub was used to store and manage the project's source code, allowing any changes made to the code to be easily tracked by the use of Git and its version control system.

- **NPM (Node Package Manager):** is a package manager for Node.js and JavaScript. NPM was used to install and manage all the libraries and dependencies necessary for the application, such as React.js, Ethers.js, OpenZeppelin, Hardhat, and Tailwind CSS.

- **MetaMask:** is a popular browser extension that serves as a cryptocurrency wallet and allows users to interact with Ethereum dapps. It provides a secure way to manage digital assets, sign transactions, and connect to decentralized applications. MetaMask was used for user authentication and transaction signing.

- **Infura:** is a very reliable service that provides access to Ethereum and IPFS (InterPlanetary File System) networks. It allows developers to interact with the blockchain without running their own infrastructure. Infura provides APIs that simplify tasks such as sending transactions, reading data from the blockchain, and deploying smart contracts. MetaMask uses Infura by default as its service provider.

- **Vercel:** is a cloud platform for deploying and hosting web applications. It provides a seamless and scalable hosting solution for web applications and dapps. Vercel was used to deploy and host the application.

### 2.6.2. Smart contract


- **Solidity:** is a programming language specifically designed for writing smart contracts on the Ethereum blockchain. Solidity was used to write the CryptoKids smart contract.

- **OpenZeppelin:** is an open-source library that provides a collection of secure and reusable smart contracts for Ethereum-based decentralised applications development. OpenZeppelin ERC20 (Ethereum Request for Comment 20) token smart contract was used to develop the CryptoKids token.

- **Hardhat:** is an Ethereum development environment that provides a suite of tools and features for smart contract development, including a local Ethereum network. Hardhat was used for smart contract testing, deployment and source code verification.

- **Sepolia Ethereum Testnet:** is a testing environment for Ethereum blockchain applications. It allows developers to experiment, deploy and interact with smart contracts without using real ETH (Ether cryptocurrency) or affecting the main Ethereum blockchain network. Sepolia Testnet was used to deploy the CryptoKids smart contract, which works as the backend of the application.

### 2.6.3. Frontend application


- **JavaScript:** is the most used programming language for web development. It allows developers to add interactivity and dynamic behaviour to web pages. JavaScript was used as the main language for the frontend application.

- **HTML (Hypertext Markup Language):** is the standard markup language for creating web pages and structuring their content. HTML was used to provide the basic structure and elements of the frontend application.

- **Tailwind CSS:** is a utility-first CSS (Cascading Style Sheets) framework that provides a set of pre-defined utility classes to style web pages. Tailwind CSS was used to rapidly build and style the frontend application's responsive user interface.

- **React.js:** is a JavaScript library for building interactive user interfaces. It provides a component-based approach, enabling developers to create reusable UI (User Interface) elements and efficiently manage state changes. React.js was used to build and structure the frontend application.

- **Ethers.js:** is a JavaScript library that provides a comprehensive set of tools and utilities for interacting with the Ethereum blockchain, in this case, Sepolia Ethereum Testnet. Ethers.js was used to simplify tasks such as connecting to the smart contract, fetching data, and sending transactions.

## 2.7. Report Structure

Starting with the executive summary and introduction sections, which provide an overview of the project's concept, purpose, aims, and benefits, this report is structured to offer a comprehensive view of the CryptoKids Dapp project and development process.

The following sections of this report are structured as follows:

- **Section 3. System:**
  This section covers in detail the various components and stages of the CryptoKids Dapp development, which includes requirements specification, design, architecture, implementation, user interface, deployment, testing and evaluation.

- **Section 4. Conclusions:**
  This section summarises the project's key achievements and outlines its results.

- **Section 5. Further development:**
  This section outlines ideas to improve the application and user experience.

- **Section 6. References:**
  This section lists the references used for research while writing this report.

- **Section 7. Table of figures:**
  This section lists all the figures used throughout the report.

# 3. System

## 3.1. Requirements

This section outlines the requirements specification for the CryptoKids Dapp (decentralised application). As the application development progressed, requirements were added or adjusted based on the application and users' needs.

e.g. the ability for users to edit their profiles, the ability for parents to filter tasks and rewards by children, and the addition of measurable metrics to the application's homepage to help anyone measure its success and quantify its impact.

### 3.1.1. Functional requirements

Functional requirements define how a system must behave, and describes the functionalities that must or should be implemented to enable users to achieve their goals.

The application's functional requirements are as follows:

**Users:**

- Parents shall be able to sign up using MetaMask.
- Parents should be able to delete their accounts.
- Parents and children shall be able to log in using MetaMask.
- Parents and children should be able to edit their profiles.

**Family Groups:**

- Parents shall be able to manage family groups.
- Parents shall be able to add children to their family groups.
- Parents shall be able to remove children from their family groups.

**Tasks:**

- Parents shall be able to create tasks and assign them to their children.
- Parents should be able to edit and delete tasks that have not been completed.
- Parents should be able to filter tasks by children.
- Children shall be able to view their assigned tasks.
- Children shall be able to mark tasks as completed.
- Children should be able to cancel task completion if not approved.
- Parents shall be able to approve task completion.
- Children shall be able to earn tokens by completing tasks.

**Rewards:**

- Parents shall be able to create rewards and assign them to their children.
- Parents should be able to edit and delete rewards that have not been purchased.
- Parents should be able to filter rewards by children.
- Children shall be able to view their assigned rewards.
- Children shall be able to exchange tokens for rewards.
- Children shall be able to redeem their rewards.
- Children should be able to cancel reward redemption if not approved
- Parents shall be able to approve reward redemption.

The use case diagram below (Figure 1) illustrates the application's key functional requirements:

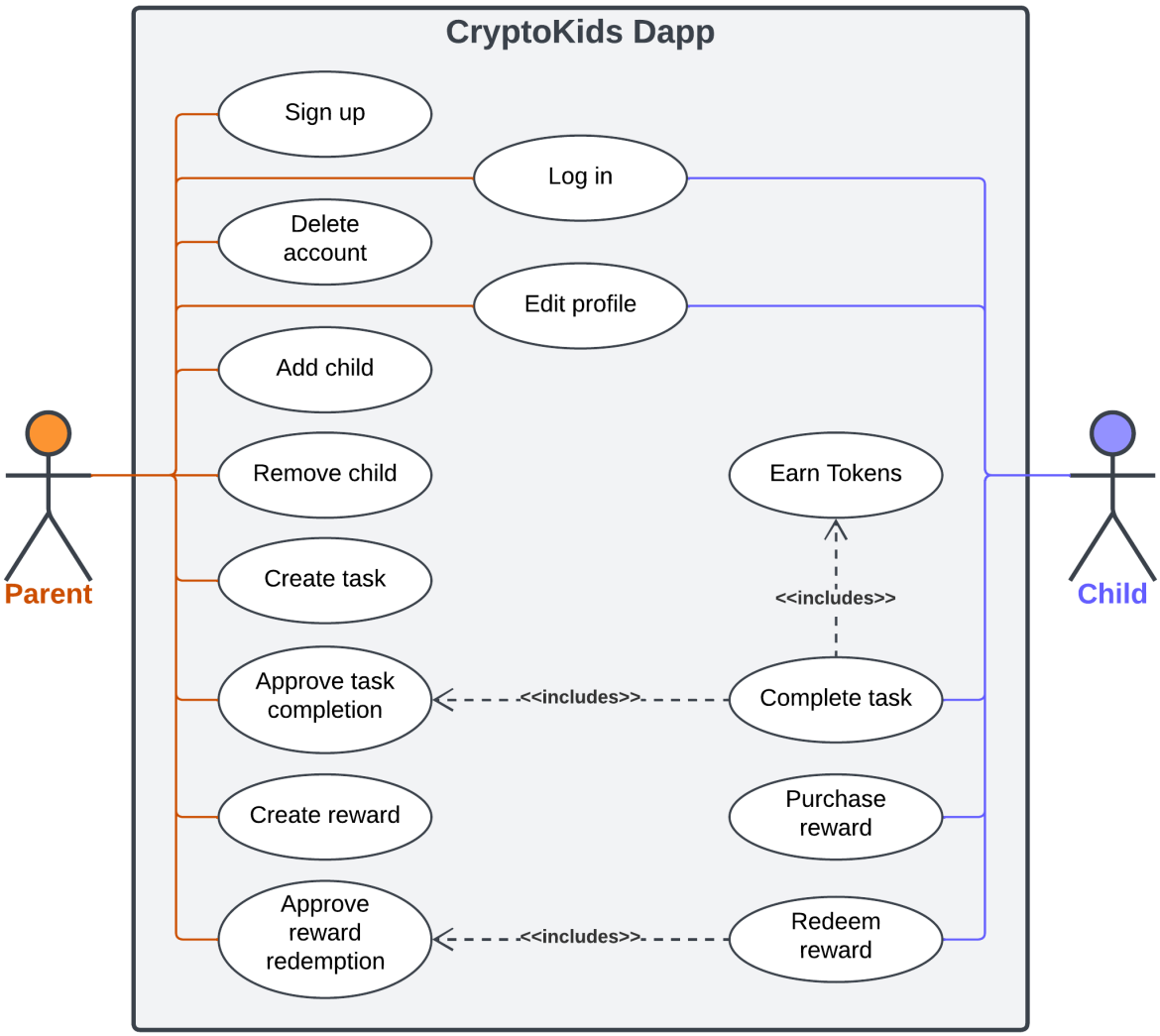



*Figure 1: CryptoKids Dapp use case diagram.*

### 3.1.2. Data requirements

Data requirements describe how the data should be stored, handled, and accessed.

The application's data requirements are as follows:

- Data shall be solely stored in the application's smart contract.
- User data shall only be accessible after user authentication.
- User data shall only be accessible by the owner or its parent (in case the user is a child).
- Authentication and authorization mechanisms shall be implemented to securely store and retrieve data.

### 3.1.3. User requirements

User requirements describe what is required from a user in order to access and use the application and its functionalities.

The application's user requirements are as follows:

- Users shall have an internet connection.
- Users shall have an internet browser that supports MetaMask installed, such as Chrome, Firefox, Brave, Edge, Opera, and MetaMask mobile application browser.
- Users shall have MetaMask installed and Sepolia Network configured.
- Users shall have an Ethereum account.
- Users should have Sepolia ETH (Ether) available in their wallets for transaction fees.

The CryptoKids Dapp was designed as a tool for parents to introduce their children to blockchain technology and its applications. Therefore, it is expected that parents have the knowledge and skills to meet the user requirements and help their children set up their accounts.

### 3.1.4. Environmental requirements

Environmental requirements define the minimum requirements for a system to function.

The application's environmental requirements are as follows:

- The application's smart contract shall be deployed to the Sepolia Ethereum Testnet.
- The application frontend shall be deployed to the cloud and made available online.
- The application should work on any device (such as smartphones, tablets, and computers) with an internet browser that supports MetaMask installed, such as Chrome, Firefox, Brave, Edge, Opera, and the MetaMask mobile application browser.
- The application should score a minimum of 80 in performance testing.

### 3.1.5. Usability requirements

Usability requirements describe how easy it should be for a user to use a system.

The application's usability requirements are as follows:

- The application should have a user-friendly interface suitable for children.
- The application navigation should be intuitive and straightforward.
- The application should be responsive and provide a smooth user experience on any device, such as smartphones, tablets, and computers.
- The application should have an informative homepage to help users understand the application, its functionalities, and its benefits.
- The application should display measurable metrics on the homepage, such as the number of parents registered, children added, tasks created, tasks completed, rewards created, rewards purchased, tokens earned, and tokens spent. This is to help anyone measure the application's success and quantify its impact.
- The application should display loading indicators when waiting for a transaction to be confirmed.

## 3.2. Design and Architecture

The CryptoKids Dapp is composed of two main components, the smart contract and the frontend application. The smart contract works as the backend of the application, where all the data is stored, and the frontend application is how the user interacts with the smart contract, through the use of a web browser.

The smart contract is stored in the Ethereum blockchain (Sepolia Ethereum Testnet) and the frontend application is hosted on a web server (Vercel), in order to establish communication between them and allow the user to interact with the smart contract, a provider and a signer are require. The provider (Infura) provides a connection to the blockchain, and the signer (MetaMask) enables users to sign and send transactions to the blockchain.

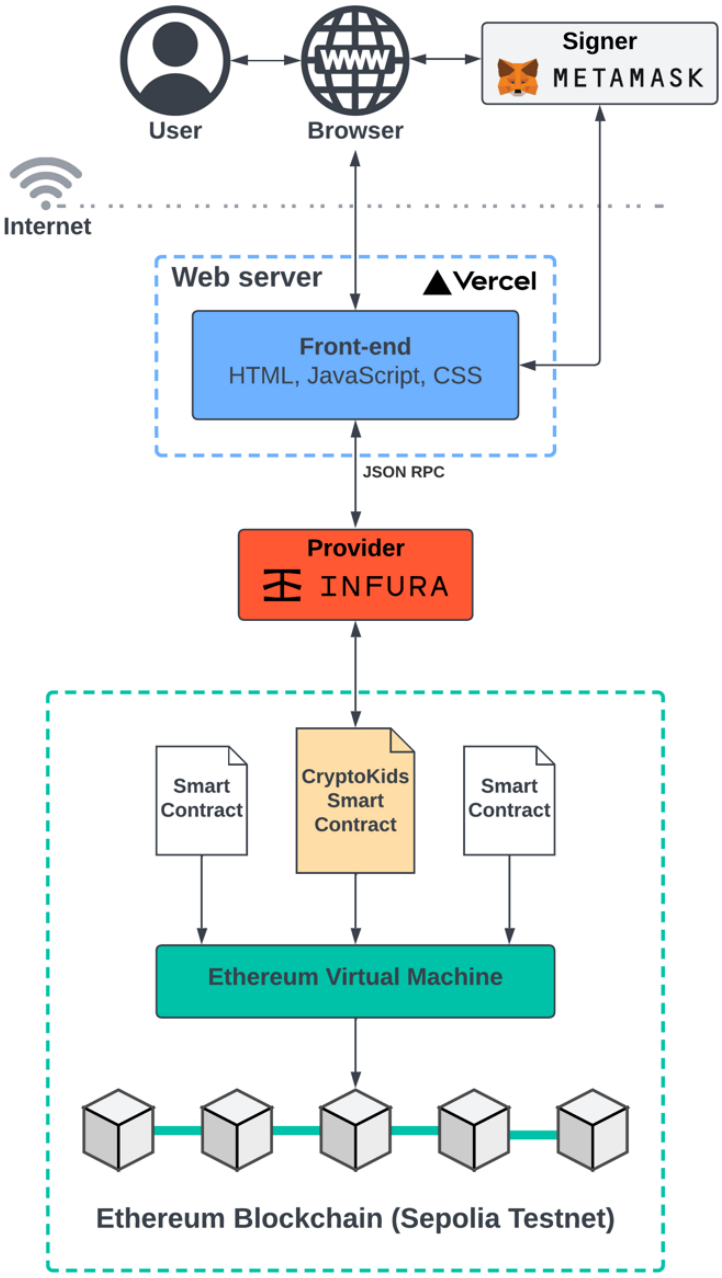



*Figure 2: CryptoKids Dapp architectural diagram.*

### 3.2.1. Pages Structure Diagram

Apart from the CryptoKids Dapp homepage, all other pages have restricted access based on the user's account type, which is defined once the user's wallet is connected to the application using MetaMask. There are three account types:

1) **Not registered:** the user is not registered in the smart contract as either a parent or a child. This account type has access to the following page:
    - **Sign up:** where a user can register as a parent.

2) **Parent:** the user is registered in the smart contract as a parent. This account type has access to the dashboard with the following pages:
    - **Family group:** where a parent can manage the children in its family group.
    - **Tasks:** where a parent can manage tasks and approve task completion.
    - **Rewards:** where a parent can manage rewards and approve reward redemption.

3) **Child:** the user is registered in the smart contract as a child. This account type has access to the dashboard with the following pages:
    - **Tasks:** where a child can see and complete its assigned tasks.
    - **Rewards:** where a child can see and redeem its purchased rewards.
    - **Marketplace:** where a child can see and purchase its assigned rewards.

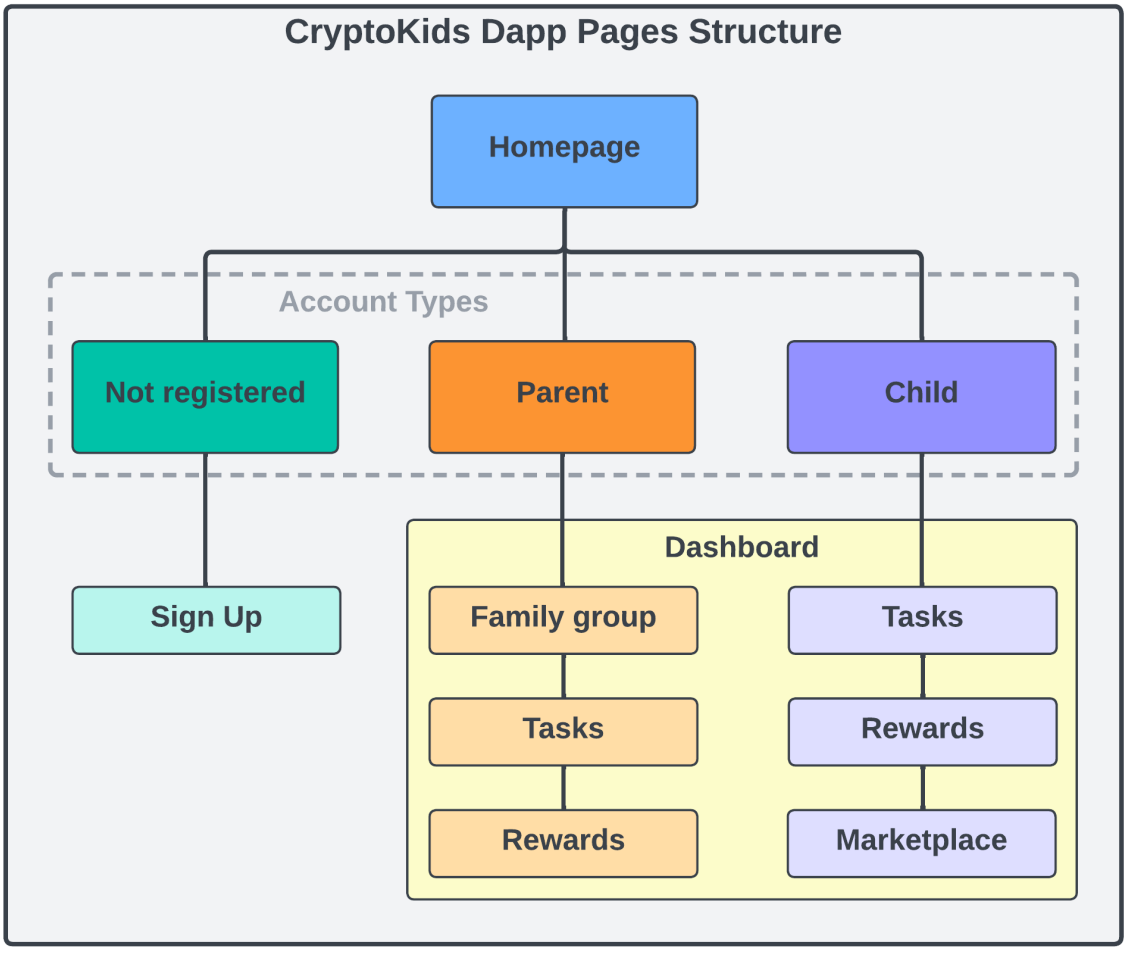



*Figure 3: CryptoKids Dapp pages structure diagram.*

## 3.3. Implementation

### 3.3.1. Project management

ClickUp was the tool used to help manage the project effectively. ClickUp is a cloud-based collaboration and project management tool that makes the process of planning a project and managing its deliverables and milestones very simple and easy.

The project's deliverables were divided into two phases (see Figure 4): Application Planning and Application Development.

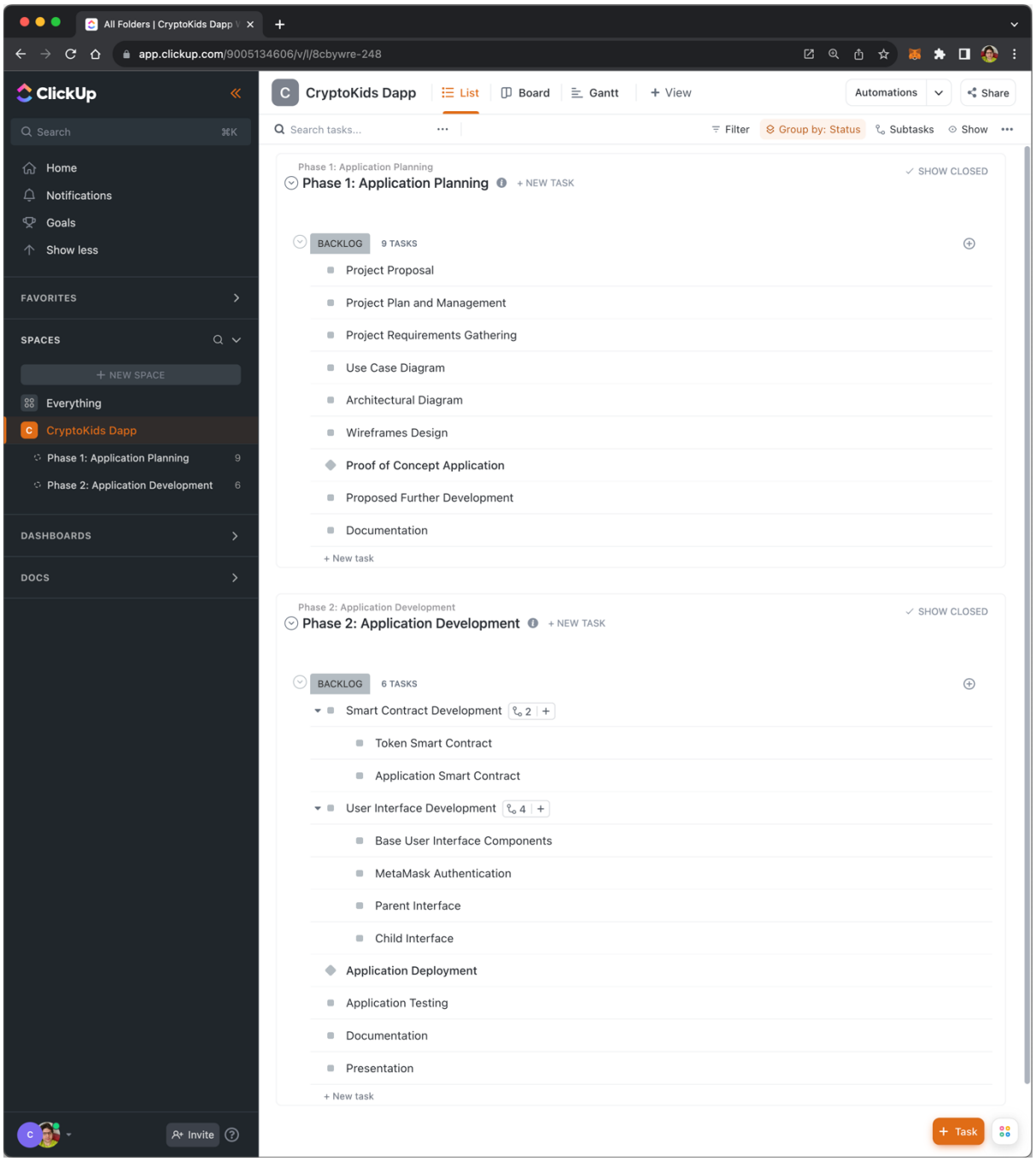



*Figure 4: Project management: project's deliverables on ClickUp.*

ClickUp also provides Gantt charts, which offer a detailed view of all the project's deliverables with specified start and due dates.

See below (Figure 5) the original Gantt chart for the project's deliverables:

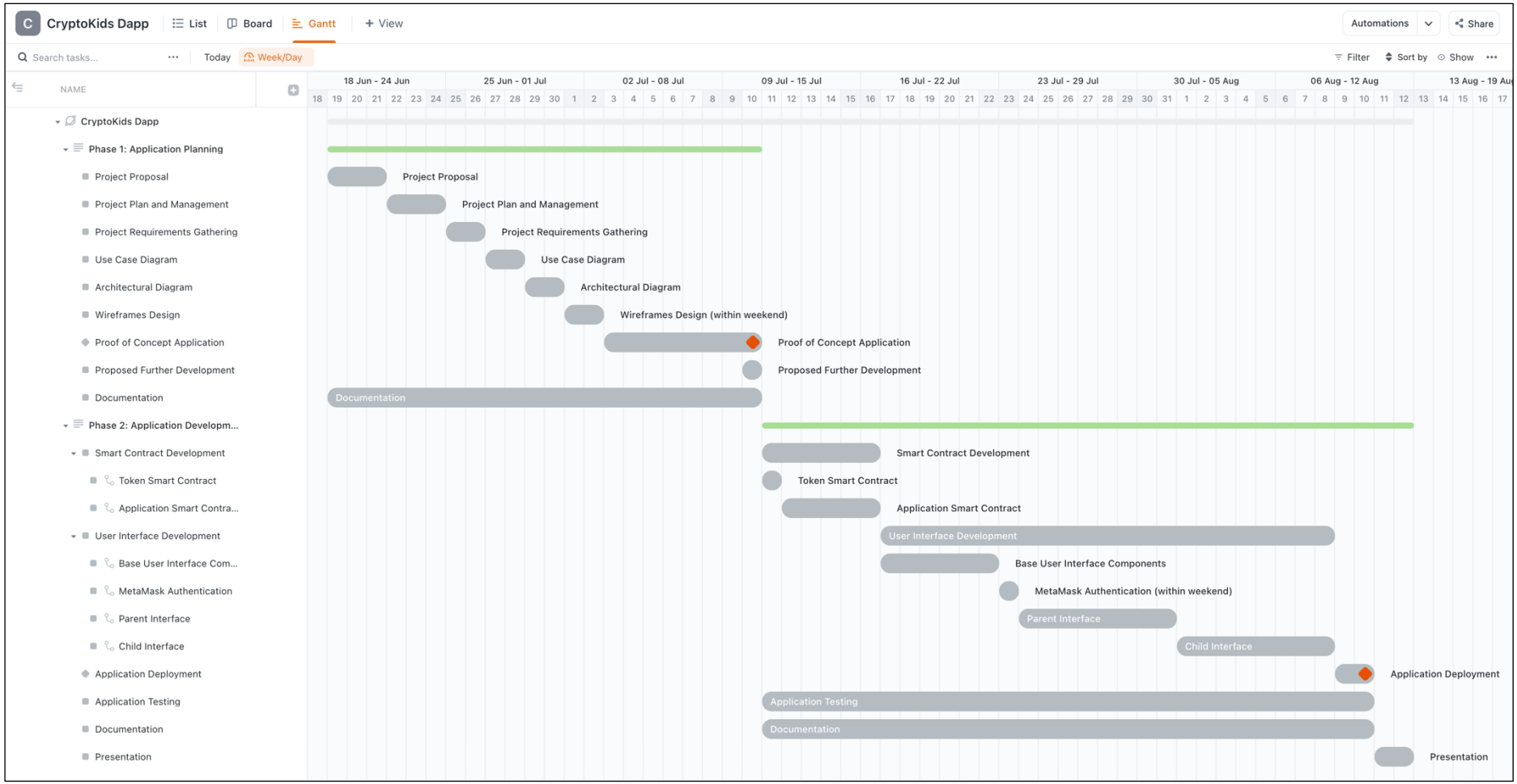



*Figure 5: Project management: original Gantt chart for the project's deliverables.*

See below (Figure 6) the final Gantt chart for the project's deliverables:

- Phase 1, Application Planning, was completed without any changes or delays.
- Phase 2, Application Development, had its start date delayed, which caused minor changes to the start and due dates of all its deliverables.
- All the project's deliverables were completed successfully.

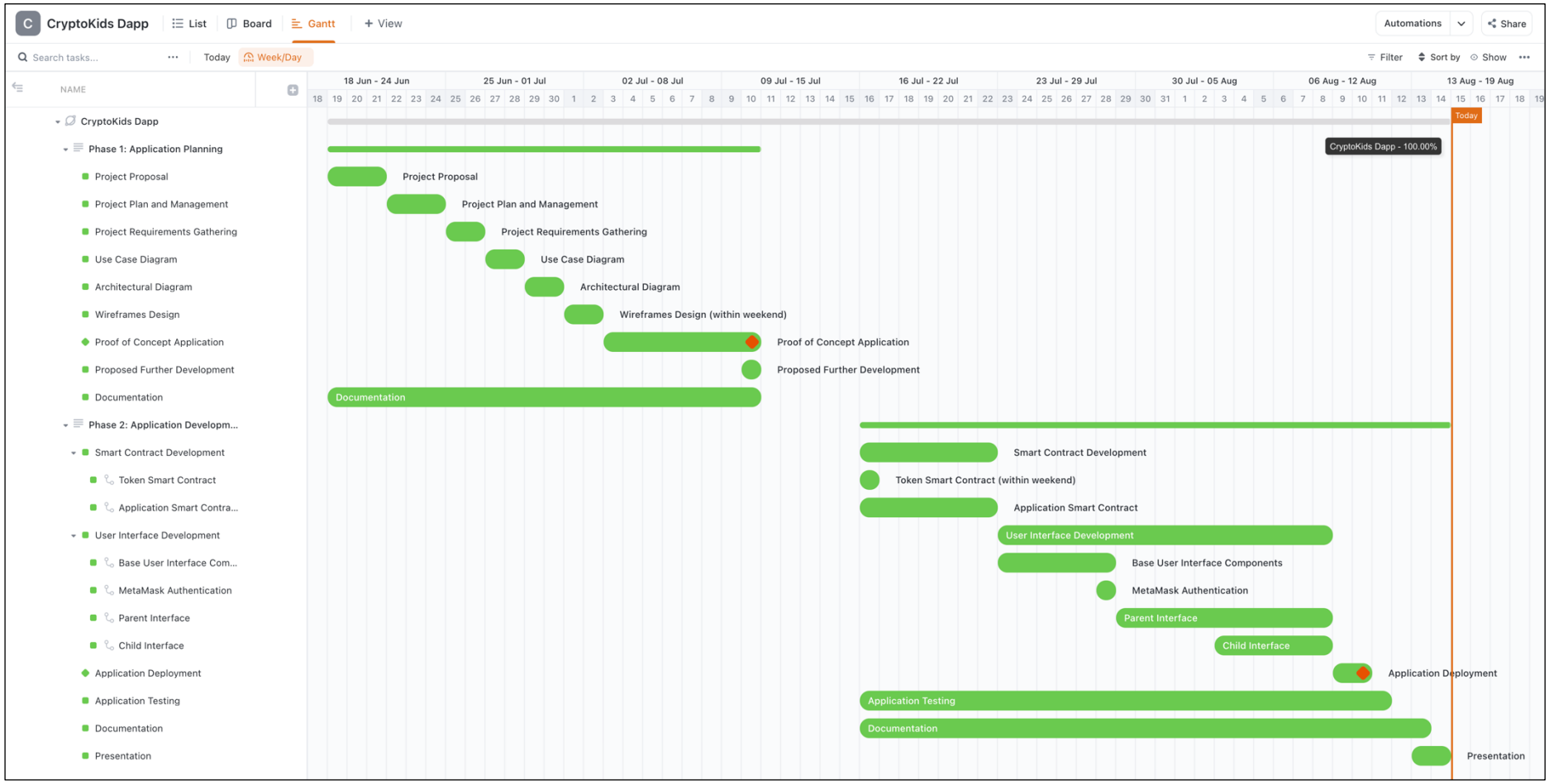
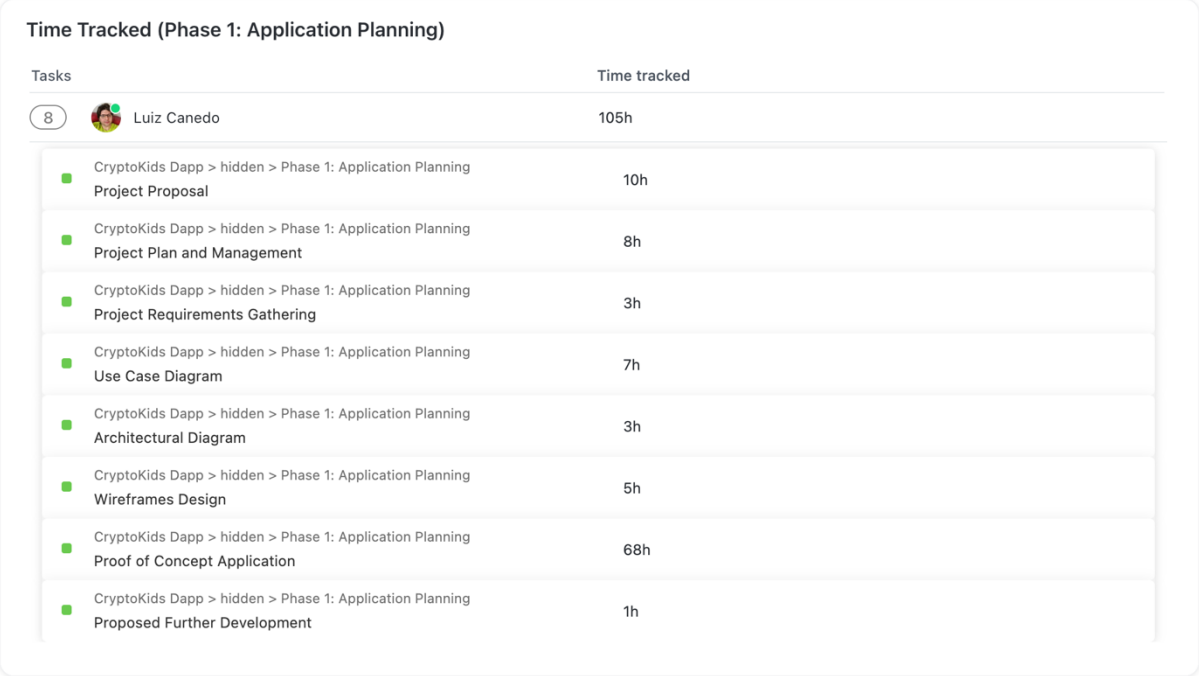



*Figure 6: Project management: final Gantt chart for the project's deliverables.*

There was a total of 317 hours and 15 minutes spent on the project, 105 hours on Phase 1 and 212 hours and 15 minutes on Phase 2.

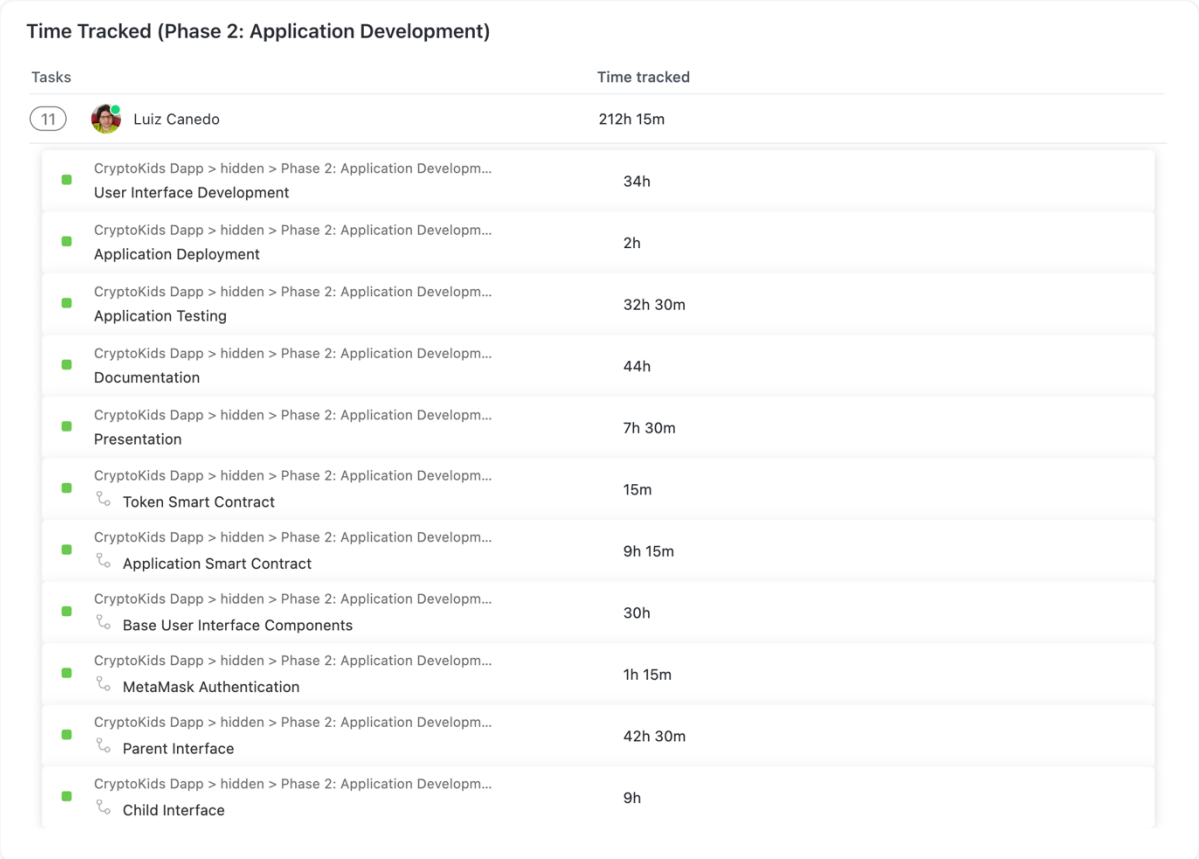
See below (Figures 7 and 8) the breakdown of the time tracked for each deliverable:



Time Tracked (Phase 1: Application Planning)

| Tasks | Time tracked |
|---|---|
| 8 Luiz Canedo | 105h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Project Proposal | 10h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Project Plan and Management | 8h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Project Requirements Gathering | 3h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Use Case Diagram | 7h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Architectural Diagram | 3h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Wireframes Design | 5h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Proof of Concept Application | 68h |
| CryptoKids Dapp > hidden > Phase 1: Application Planning<br>Proposed Further Development | 1h |

*Figure 7: Project management: phase 1 time tracked.*



Time Tracked (Phase 2: Application Development)

| Tasks | Time tracked |
|---|---|
| 11 Luiz Canedo | 212h 15m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>User Interface Development | 34h |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Application Deployment | 2h |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Application Testing | 32h 30m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Documentation | 44h |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Presentation | 7h 30m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Token Smart Contract | 15m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Application Smart Contract | 9h 15m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Base User Interface Components | 30h |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>MetaMask Authentication | 1h 15m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Parent Interface | 42h 30m |
| CryptoKids Dapp > hidden > Phase 2: Application Developm...<br>Child Interface | 9h |

*Figure 8: Project management: phase 2 time tracked.*

### 3.3.2. Tools used

Visual Studio Code was the code editor used to implement the CryptoKids smart contract and frontend application. Visual Studio Code offers an integrated terminal and a large amount of extensions, which helped on optimizing the development process by having access to all the tools needed in one place.

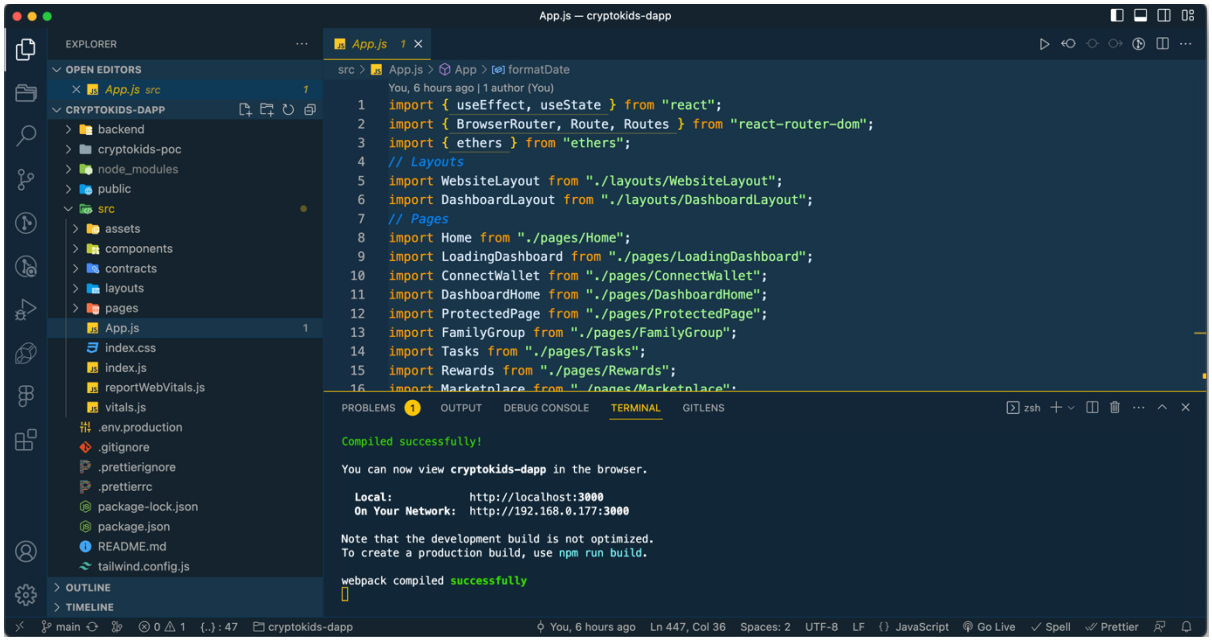



*Figure 9: Tools used: CryptoKids Dapp development in Visual Studio Code.*

GitHub was used to store and manage the application's source code, allowing any changes made to the code to be easily tracked by the use of Git and its version control system.

**GitHub link:**
https://github.com/canedobox/cryptokids-dapp

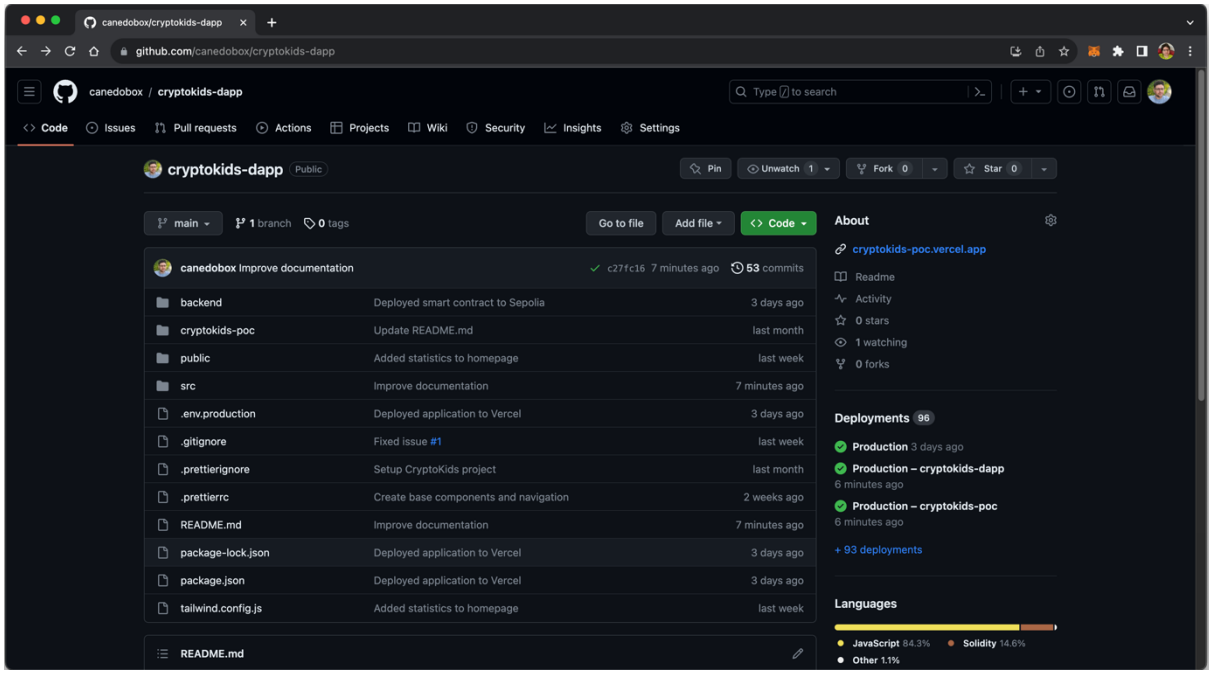



*Figure 10: Tools used: CryptoKids Dapp source code in GitHub.*

### 3.3.3. Smart contract implementation

The CryptoKids smart contract was carefully designed and implemented to meet the requirements specification for the application:

- All the functional requirements were implemented.
- The application and user data are solely stored in the smart contract.
- User data is only accessible by the owner or its parent (in case the user is a child).
- Authorization mechanisms were implemented to securely store and retrieve data.
- The smart contract includes public measurable metrics data, such as the number of parents registered, children added, tasks created, tasks completed, rewards created, rewards purchased, tokens earned, and tokens spent.

**CryptoKids smart contract source code:**
https://github.com/canedobox/cryptokids-dapp/blob/main/backend/contracts/CryptoKids.sol

See below the key functionalities of the CryptoKids smart contract.

#### 3.3.3.1. ERC20 token:

One of the main functionalities of the CryptoKids Dapp is to allow children to earn tokens that can later be exchanged for rewards. For that reason, the CryptoKids smart contract is of type ERC20 (Ethereum Request for Comments 20), which is a standard for fungible tokens. This means that 1 CK (CryptoKids) token is and will always be exactly the same (in type and value) as all other CK tokens.

OpenZeppelin is the most used library for secure smart contract development. Its ERC20 token smart contract was developed following the Ethereum token standards and it is continually tested by its developers and community. It also includes all the token functionalities needed for the application, such as minting, transferring and burning tokens.

**OpenZeppelin ERC20 token documentation:** https://docs.openzeppelin.com/contracts/4.x/erc20

```
1    // SPDX-License-Identifier: MIT
2    // Author: @canedobox
3
4    pragma solidity ^0.8.0;
5
6    import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7
8    contract CryptoKids is ERC20 {
9      ...
176    /***** CONSTRUCTOR *****/
177    /**
178     * CryptoKids smart contract constructor.
179     * @param name_ Token name.
180     * @param symbol_ Token symbol.
181     * NOTE: The token name and symbol are required by the ERC20 standard.
182     */
183    constructor(
184      string memory name_,
185      string memory symbol_
186    ) ERC20(name_, symbol_) {
187      // ...
188    }
189    ...
1233 }
```

*Figure 11: CryptoKids.sol: ERC20 token constructor.*

### 3.3.3.2. Register a parent:

```
206  /**
207   * Register caller as a parent.
208   * @param name_ Parent's name.
209   * Requirements:
210   * - Caller must not be already registered as a parent.
211   * - Caller must not be already registered as a child.
212   */
213  function registerParent(string memory name_) external {
214    // Check if the caller is already registered as a parent.
215    require(
216      !_parents[msg.sender].exists,
217      "You are already registered as a parent."
218    );
219    // Check if the caller is already registered as a child.
220    require(
221      !_children[msg.sender].exists,
222      "You are already registered as a child."
223    );
224
225    // Add caller to the parents mapping.
226    _parents[msg.sender] = Parent(name_, true);
227
228    // Increment parents registered counter.
229    accountsCounter.parentsRegistered++;
230
231    // Emit event.
232    emit ParentRegistered(msg.sender);
233  }
```

*Figure 12: CryptoKids.sol code snippet: register a parent.*

### 3.3.3.3. Delete a parent account:

```
235  /**
236   * Delete caller's parent account and family group.
237   * NOTE: This function will also remove all children
238   *       in the caller's family group.
239   * Requirements:
240   * - The caller must be a parent.
241   */
242  function deleteParent() external onlyParent {
243    // Remove children from the caller's family group.
244    // Get the caller's family group.
245    address[] memory familyGroup = _familyGroups[msg.sender];
246    // Loop through the family group.
247    for (uint256 i = 0; i < familyGroup.length; i++) {
248      // Remove child from the family group.
249      removeChild(familyGroup[i]);
250    }
251
252    // Delete parent.
253    delete _parents[msg.sender];
254    // Delete caller's family group.
255    delete _familyGroups[msg.sender];
256
257    // Increment parents deleted counter.
258    accountsCounter.parentsDeleted++;
259
260    // Emit event.
261    emit ParentDeleted(msg.sender);
262  }
```

*Figure 13: CryptoKids.sol code snippet: delete a parent account.*

### 3.3.3.4. Add a child to a family group:

```
264   /**
265    * Add a child to the caller's family group.
266    * @param child_ Child's address to be added.
267    * @param name_ Child's name.
268    * Requirements:
269    * - The caller must be a parent.
270    * - The child address must not be registered as a parent.
271    * - The child address must not be registered as a child.
272    */
273   function addChild(address child_, string memory name_) external onlyParent {
274     // Check if the child address is already registered as a parent.
275     require(
276       !_parents[child_].exists,
277       "Address already registered as a parent."
278     );
279     // Check if the child address is already registered as a child.
280     require(
281       !_children[child_].exists,
282       "Address already registered as a child."
283     );
284
285     // Add child to the children mapping.
286     _children[child_] = Child(child_, msg.sender, name_, true);
287     // Add child to the caller's family group.
288     _familyGroups[msg.sender].push(child_);
289
290     // Increment children added counter.
291     accountsCounter.childrenAdded++;
292
293     // Emit event.
294     emit ChildAdded(msg.sender, child_);
295   }
```

*Figure 14: CryptoKids.sol code snippet: add a child to a family group.*

### 3.3.3.5. Remove a child from a family group:

Removing a child from a family group will not delete the child's tasks, rewards, and tokens. Once a child that was previously removed is re-added to the same family group (or a different one), the child's account is reactivated, together with its tasks, rewards, and tokens. This function was designed this way to protect the child, by not allowing parents to delete tasks waiting for approval or rewards that have been purchased.

```
297  /**
298   * Remove a child from the caller's family group.
299   * @param child_ Child's address to be removed.
300   * NOTE: This function will NOT delete the child's tasks, rewards or tokens.
301   * Requirements:
302   * - The caller must be a parent.
303   * - The child address must belong to a child.
304   * - The child address must be part of the caller's family group.
305   */
306  function removeChild(address child_) public onlyParent {
307    // Check if the address provided belongs to a child.
308    require(_children[child_].exists, "Address does not belong to a child.");
309    // Check if the child is part of the caller's family group.
310    require(
311      _children[child_].familyGroup == msg.sender,
312      "This child is not part of your family group."
313    );
314
315    // Remove child from the caller's family group.
316    // Get the caller's family group.
317    address[] storage familyGroup = _familyGroups[msg.sender];
318    // Loop through the family group.
319    for (uint256 i = 0; i < familyGroup.length; i++) {
320      // Find the index of the child in the array.
321      if (familyGroup[i] == child_) {
322        // Replace the child with the last child in the array and...
323        familyGroup[i] = familyGroup[familyGroup.length - 1];
324        // ... delete the last child.
325        familyGroup.pop();
326        // Stop the loop.
327        break;
328      }
329    }
330
331    // Delete child.
332    delete _children[child_];
333
334    // Increment children removed counter.
335    accountsCounter.childrenRemoved++;
336
337    // Emit event.
338    emit ChildRemoved(msg.sender, child_);
339  }
```

*Figure 15: CryptoKids.sol code snippet: remove a child from a family group.*

### 3.3.3.6. Edit profile:

```
401  /**
402   * Edit caller's profile.
403   * @param name_ New name.
404   * Requirements:
405   * - The caller must be registered as either a parent or a child.
406   */
407  function editProfile(string memory name_) external {
408    // Check if caller is registered as either a parent or a child.
409    require(
410      (_parents[msg.sender].exists || _children[msg.sender].exists),
411      "You are not registered."
412    );
413
414    // If caller is registered as a parent.
415    if (_parents[msg.sender].exists) {
416      // Updated parent profile.
417      _parents[msg.sender].name = name_;
418      // Emit event.
419      emit ParentProfileEdited(msg.sender);
420    }
421    // If caller is registered as a child.
422    else if (_children[msg.sender].exists) {
423      // Updated child profile.
424      _children[msg.sender].name = name_;
425      // Emit event.
426      emit ChildProfileEdited(msg.sender);
427    }
428  }
```

*Figure 16: CryptoKids.sol code snippet: edit profile.*

### 3.3.3.7. Add a task:

```
434   /**
435    * Add a task and assign it to a child.
436    * @param child_ Child's address to assign the task to.
437    * @param description_ Task's description.
438    * @param reward_ Task's reward in CK tokens.
439    * @param dueDate_ Task's due date (unix timestamp), provide 0 for no due date.
440    * Requirements:
441    * - The caller must be a parent.
442    * - The child address must belong to a child.
443    * - The child must be part of the caller's family group.
444    */
445   function addTask(
446     address child_,
447     string memory description_,
448     uint256 reward_,
449     uint256 dueDate_
450   ) external onlyParent {
451     // Check if the address provided belongs to a child.
452     require(_children[child_].exists, "Address does not belong to a child.");
453     // Check if the child is part of the caller's family group.
454     require(
455       _children[child_].familyGroup == msg.sender,
456       "This child is not part of your family group."
457     );
458
459     // Increment tasks added counter.
460     tasksCounter.added++;
461
462     // Add task.
463     _tasks[tasksCounter.added] = Task(
464       tasksCounter.added,
465       child_,
466       description_,
467       reward_,
468       dueDate_,
469       false,
470       0,
471       false,
472       0,
473       true
474     );
475
476     // Assign task to the child.
477     _childTasks[child_].push(tasksCounter.added);
478
479     // Emit event.
480     emit TaskAdded(tasksCounter.added, child_, description_, reward_, dueDate_);
481   }
```

*Figure 17: CryptoKids.sol code snippet: add a task.*

### 3.3.3.8. Edit a task:

```
483   /**
484    * Edit a task.
485    * @param taskId_ Task's ID.
486    * @param description_ Task's description.
487    * @param reward_ Task's reward in CK tokens.
488    * @param dueDate_ Task's due date (unix timestamp), provide 0 for no due date.
489    * Requirements:
490    * - The caller must be a parent.
491    * - The task ID must be valid.
492    * - The child the task is assigned to must be part of the caller's family group.
493    * - The task must not have been completed.
494    */
495   function editTask(
496     uint256 taskId_,
497     string memory description_,
498     uint256 reward_,
499     uint256 dueDate_
500   ) external onlyParent {
501     // Check if the task ID is valid.
502     require(_tasks[taskId_].exists, "Invalid task ID.");
503     // Check if the child the task is assigned to is part of the caller's family group.
504     require(
505       _children[_tasks[taskId_].assignedTo].familyGroup == msg.sender,
506       "The child this task is assigned to is not part of your family group."
507     );
508     // Check if the task has not been completed.
509     require(!_tasks[taskId_].completed, "Task has already been completed.");
510
511     // Update task details.
512     _tasks[taskId_].description = description_;
513     _tasks[taskId_].reward = reward_;
514     _tasks[taskId_].dueDate = dueDate_;
515
516     // Emit event.
517     emit TaskEdited(taskId_, description_, reward_, dueDate_);
518   }
```

*Figure 18: CryptoKids.sol code snippet: edit a task.*

### 3.3.3.9. Delete a task:

```
520  /**
521   * Delete a task.
522   * @param taskId_ Task's ID.
523   * Requirements:
524   * - The caller must be a parent.
525   * - The task ID must be valid.
526   * - The child the task is assigned to must be part of the caller's family group.
527   * - The task must not have been completed.
528   */
529  function deleteTask(uint256 taskId_) external onlyParent {
530    // Check if the task ID is valid.
531    require(_tasks[taskId_].exists, "Invalid task ID.");
532    // Check if the child the task is assigned to is part of the caller's family group.
533    require(
534      _children[_tasks[taskId_].assignedTo].familyGroup == msg.sender,
535      "The child this task is assigned to is not part of your family group."
536    );
537    // Check if the task has not been completed.
538    require(!_tasks[taskId_].completed, "Task has already been completed.");
539
540    // Remove task from the child's tasks.
541    // Get the child's tasks.
542    uint256[] storage childTasks = _childTasks[_tasks[taskId_].assignedTo];
543    // Loop through the child's tasks.
544    for (uint256 i = 0; i < childTasks.length; i++) {
545      // Find the index of the task in the array.
546      if (childTasks[i] == taskId_) {
547        // Replace the task with the last task in the array and...
548        childTasks[i] = childTasks[childTasks.length - 1];
549        // ... delete the last task.
550        childTasks.pop();
551        // Stop the loop.
552        break;
553      }
554    }
555
556    // Delete task.
557    delete _tasks[taskId_];
558
559    // Increment tasks deleted counter.
560    tasksCounter.deleted++;
561
562    // Emit event.
563    emit TaskDeleted(taskId_);
564  }
```

*Figure 19: CryptoKids.sol code snippet: delete a task.*

### 3.3.3.10. Complete a task:

```
566   /**
567    * Complete a task.
568    * @param taskId_ Task's ID.
569    * Requirements:
570    * - The caller must be a child.
571    * - The task ID must be valid.
572    * - The task must belong to the caller.
573    * - The task must not have been completed.
574    * - The task must not have expired.
575    */
576   function completeTask(uint256 taskId_) external onlyChild {
577     // Check if the task ID is valid.
578     require(_tasks[taskId_].exists, "Invalid task ID.");
579     // Check if the task belongs to the caller.
580     require(_tasks[taskId_].assignedTo == msg.sender, "This is not your task.");
581     // Check if the task has not been completed.
582     require(!_tasks[taskId_].completed, "You already completed this task.");
583     // Check if the task has not expired.
584     if (_tasks[taskId_].dueDate > 0) {
585       require(
586         _tasks[taskId_].dueDate >= block.timestamp,
587         "This task has expired."
588       );
589     }
590
591     // Complete task.
592     _tasks[taskId_].completed = true;
593     _tasks[taskId_].completionDate = block.timestamp;
594
595     // Increment tasks completed counter.
596     tasksCounter.completed++;
597
598     // Emit event.
599     emit TaskCompleted(taskId_, msg.sender, _tasks[taskId_].completionDate);
600   }
```

*Figure 20: CryptoKids.sol code snippet: complete a task.*

### 3.3.3.11. Cancel task completion:

```
602  /**
603   * Cancel task completion.
604   * @param taskId_ Task's ID.
605   * Requirements:
606   * - The caller must be a child.
607   * - The task ID must be valid.
608   * - The task must belong to the caller.
609   * - The task must have been completed.
610   * - The task completion must not have been approved.
611   */
612  function cancelTaskCompletion(uint256 taskId_) external onlyChild {
613    // Check if the task ID is valid.
614    require(_tasks[taskId_].exists, "Invalid task ID.");
615    // Check if the task belongs to the caller.
616    require(_tasks[taskId_].assignedTo == msg.sender, "This is not your task.");
617    // Check if the task has been completed.
618    require(_tasks[taskId_].completed, "You haven't completed this task yet.");
619    // Check if the task completion has not been approved.
620    require(
621      !_tasks[taskId_].approved,
622      "Task completion has already been approved."
623    );
624
625    // Cancel task completion.
626    _tasks[taskId_].completed = false;
627    _tasks[taskId_].completionDate = 0;
628
629    // Decrement tasks completed counter.
630    tasksCounter.completed--;
631
632    // Emit event.
633    emit TaskCompletionCancelled(taskId_);
634  }
```

*Figure 21: CryptoKids.sol code snippet: cancel task completion.*

### 3.3.3.12. Approve task completion:

```
636  /**
637   * Approve task completion.
638   * @param taskId_ Task's ID.
639   * Requirements:
640   * - The caller must be a parent.
641   * - The task ID must be valid.
642   * - The child the task is assigned to must be part of the caller's family group.
643   * - The task must have been completed.
644   * - The task completion must not have been approved.
645   */
646  function approveTaskCompletion(uint256 taskId_) external onlyParent {
647    // Check if the task ID is valid.
648    require(_tasks[taskId_].exists, "Invalid task ID.");
649    // Check if the child the task is assigned to is part of the caller's family group.
650    require(
651      _children[_tasks[taskId_].assignedTo].familyGroup == msg.sender,
652      "The child this task is assigned to is not part of your family group."
653    );
654    // Check if the task has been completed.
655    require(_tasks[taskId_].completed, "Task has not been completed yet.");
656    // Check if the task completion has not been approved.
657    require(
658      !_tasks[taskId_].approved,
659      "Task completion has already been approved."
660    );
661
662    // Approve task completion.
663    _tasks[taskId_].approved = true;
664    _tasks[taskId_].approvalDate = block.timestamp;
665
666    // Send token reward to the child.
667    _mint(_tasks[taskId_].assignedTo, _tasks[taskId_].reward);
668    // Increment tokens earned counter.
669    tasksCounter.tokensEarned += _tasks[taskId_].reward;
670
671    // Increment tasks approved counter.
672    tasksCounter.approved++;
673
674    // // Emit event.
675    emit TaskCompletionApproved(
676      taskId_,
677      msg.sender,
678      _tasks[taskId_].approvalDate
679    );
680  }
```

*Figure 22: CryptoKids.sol code snippet: approve task completion.*

### 3.3.3.13. Add a reward:

```
805   /**
806    * Add a reward and assign it to a child.
807    * @param child_ Child's address to assign the reward to.
808    * @param description_ Reward's description.
809    * @param price_ Reward's price in CK tokens.
810    * Requirements:
811    * - The caller must be a parent.
812    * - The address provided must belong to a child.
813    * - The child must be part of the caller's family group.
814    */
815   function addReward(
816     address child_,
817     string memory description_,
818     uint256 price_
819   ) external onlyParent {
820     // Check if the address provided belongs to a child.
821     require(_children[child_].exists, "Address does not belong to a child.");
822     // Check if the child is part of the caller's family group.
823     require(
824       _children[child_].familyGroup == msg.sender,
825       "This child is not part of your family group."
826     );
827
828     // Increment rewards added counter.
829     rewardsCounter.added++;
830
831     // Add reward.
832     _rewards[rewardsCounter.added] = Reward(
833       rewardsCounter.added,
834       child_,
835       description_,
836       price_,
837       false,
838       0,
839       false,
840       0,
841       false,
842       0,
843       true
844     );
845
846     // Assign reward to the child.
847     _childRewards[child_].push(rewardsCounter.added);
848
849     // Emit event.
850     emit RewardAdded(rewardsCounter.added, child_, description_, price_);
851   }
```

*Figure 23: CryptoKids.sol code snippet: add a reward.*

### 3.3.3.14. Edit a reward:

```
853   /**
854    * Edit a reward.
855    * @param rewardId_ Reward's ID.
856    * @param description_ Reward's description.
857    * @param price_ Reward's price in CK tokens.
858    * Requirements:
859    * - The caller must be a parent.
860    * - The reward ID must be valid.
861    * - The child the reward is assigned to must be part of the caller's family group.
862    * - The reward must not have been purchased.
863    */
864   function editReward(
865     uint256 rewardId_,
866     string memory description_,
867     uint256 price_
868   ) external onlyParent {
869     // Check if the reward ID is valid.
870     require(_rewards[rewardId_].exists, "Invalid reward ID.");
871     // Check if the child the reward is assigned to is part of the caller's family group.
872     require(
873       _children[_rewards[rewardId_].assignedTo].familyGroup == msg.sender,
874       "The child this reward is assigned to is not part of your family group."
875     );
876     // Check if the reward has not been purchased.
877     require(
878       !_rewards[rewardId_].purchased,
879       "Reward has already been purchased."
880     );
881
882     // Update reward details.
883     _rewards[rewardId_].description = description_;
884     _rewards[rewardId_].price = price_;
885
886     // Emit event.
887     emit RewardEdited(rewardId_, description_, price_);
888   }
```

*Figure 24: CryptoKids.sol code snippet: edit a reward.*

### 3.3.3.15. Delete a reward:

```
890   /**
891    * Delete a reward.
892    * @param rewardId_ Reward's ID.
893    * Requirements:
894    * - The caller must be a parent.
895    * - The reward ID must be valid.
896    * - The child the reward is assigned to must be part of the caller's family group.
897    * - The reward must not have been purchased.
898    */
899   function deleteReward(uint256 rewardId_) external onlyParent {
900     // Check if the reward ID is valid.
901     require(_rewards[rewardId_].exists, "Invalid reward ID.");
902     // Check if the child the reward is assigned to is part of the caller's family group.
903     require(
904       _children[_rewards[rewardId_].assignedTo].familyGroup == msg.sender,
905       "The child this reward is assigned to is not part of your family group."
906     );
907     // Check if the reward has not been purchased.
908     require(
909       !_rewards[rewardId_].purchased,
910       "Reward has already been purchased."
911     );
912
913     // Remove reward from the child's rewards.
914     // Get the child's rewards.
915     uint256[] storage childRewards = _childRewards[
916       _rewards[rewardId_].assignedTo
917     ];
918     // Loop through the child's rewards.
919     for (uint256 i = 0; i < childRewards.length; i++) {
920       // Find the index of the reward in the array.
921       if (childRewards[i] == rewardId_) {
922         // Replace the reward with the last reward in the array and...
923         childRewards[i] = childRewards[childRewards.length - 1];
924         // ... delete the last reward.
925         childRewards.pop();
926         // Stop the loop.
927         break;
928       }
929     }
930
931     // Delete reward.
932     delete _rewards[rewardId_];
933
934     // Increment rewards deleted counter.
935     rewardsCounter.deleted++;
936
937     // Emit event.
938     emit RewardDeleted(rewardId_);
939   }
```

*Figure 25: CryptoKids.sol code snippet: delete a reward.*

### 3.3.3.16. Purchase a reward:

```
941  /**
942   * Purchase a reward.
943   * @param rewardId_ Reward's ID.
944   * Requirements:
945   * - The caller must be a child.
946   * - The reward ID must be valid.
947   * - The reward must belong to the caller.
948   * - The reward must not have been purchased.
949   * - The child must have enough CK tokens.
950   */
951  function purchaseReward(uint256 rewardId_) external onlyChild {
952    // Check if the reward ID is valid.
953    require(_rewards[rewardId_].exists, "Invalid reward ID.");
954    // Check if the reward belongs to the caller.
955    require(
956      _rewards[rewardId_].assignedTo == msg.sender,
957      "This is not your reward."
958    );
959    // Check if the reward has not been purchased.
960    require(
961      !_rewards[rewardId_].purchased,
962      "You already purchased this reward."
963    );
964    // Check if child has enough tokens.
965    require(
966      balanceOf(msg.sender) >= _rewards[rewardId_].price,
967      "You don't have enough CK tokens."
968    );
969
970    // Burn tokens.
971    _burn(msg.sender, _rewards[rewardId_].price);
972    // Increment tokens spent counter.
973    rewardsCounter.tokensSpent += _rewards[rewardId_].price;
974
975    // Purchase reward.
976    _rewards[rewardId_].purchased = true;
977    _rewards[rewardId_].purchaseDate = block.timestamp;
978
979    // Increment rewards purchased counter.
980    rewardsCounter.purchased++;
981
982    // Emit event.
983    emit RewardPurchased(
984      rewardId_,
985      _rewards[rewardId_].price,
986      msg.sender,
987      _rewards[rewardId_].purchaseDate
988    );
989  }
```

*Figure 26: CryptoKids.sol code snippet: purchase a reward.*

### 3.3.3.17. Redeem a reward:

```
991   /**
992    * Redeem a reward.
993    * @param rewardId_ Reward's ID.
994    * Requirements:
995    * - The caller must be a child.
996    * - The reward ID must be valid.
997    * - The reward must belong to the caller.
998    * - The reward must have been purchased.
999    * - The reward must not have been redeemed.
1000   */
1001  function redeemReward(uint256 rewardId_) external onlyChild {
1002    // Check if the reward ID is valid.
1003    require(_rewards[rewardId_].exists, "Invalid reward ID.");
1004    // Check if the reward belongs to the caller.
1005    require(
1006      _rewards[rewardId_].assignedTo == msg.sender,
1007      "This is not your reward."
1008    );
1009    // Check if the reward has been purchased.
1010    require(
1011      _rewards[rewardId_].purchased,
1012      "You haven't purchased this reward yet."
1013    );
1014    // Check if the reward has not been redeemed.
1015    require(!_rewards[rewardId_].redeemed, "You already redeemed this reward.");
1016
1017    // Redeem reward.
1018    _rewards[rewardId_].redeemed = true;
1019    _rewards[rewardId_].redemptionDate = block.timestamp;
1020
1021    // Increment rewards redeemed counter.
1022    rewardsCounter.redeemed++;
1023
1024    // Emit event.
1025    emit RewardRedeemed(
1026      rewardId_,
1027      msg.sender,
1028      _rewards[rewardId_].redemptionDate
1029    );
1030  }
```

*Figure 27: CryptoKids.sol code snippet: redeem a reward.*

### 3.3.3.18. Cancel reward redemption:

```
1032  /**
1033   * Cancel reward redemption.
1034   * @param rewardId_ Reward's ID.
1035   * Requirements:
1036   * - The caller must be a child.
1037   * - The reward ID must be valid.
1038   * - The reward must belong to the caller.
1039   * - The reward must have been redeemed.
1040   * - The reward redemption must not have been approved.
1041   */
1042  function cancelRewardRedemption(uint256 rewardId_) external onlyChild {
1043    // Check if the reward ID is valid.
1044    require(_rewards[rewardId_].exists, "Invalid reward ID.");
1045    // Check if the reward belongs to the caller.
1046    require(
1047      _rewards[rewardId_].assignedTo == msg.sender,
1048      "This is not your reward."
1049    );
1050    // Check if the reward has not been redeemed.
1051    require(
1052      _rewards[rewardId_].redeemed,
1053      "You haven't redeemed this reward yet."
1054    );
1055    // Check if the reward redemption has not been approved.
1056    require(
1057      !_rewards[rewardId_].approved,
1058      "Reward redemption has already been approved."
1059    );
1060
1061    // Cancel reward redemption.
1062    _rewards[rewardId_].redeemed = false;
1063    _rewards[rewardId_].redemptionDate = 0;
1064
1065    // Decrement rewards redeemed counter.
1066    rewardsCounter.redeemed--;
1067
1068    // Emit event.
1069    emit RewardRedemptionCancelled(rewardId_);
1070  }
```

*Figure 28: CryptoKids.sol code snippet: cancel reward redemption.*

### 3.3.3.19. Approve reward redemption:

```
1072  /**
1073   * Approve reward redemption.
1074   * @param rewardId_ Reward's ID.
1075   * Requirements:
1076   * - The caller must be a parent.
1077   * - The reward ID must be valid.
1078   * - The reward must have been redeemed.
1079   * - The reward redemption must not have been approved.
1080   */
1081  function approveRewardRedemption(uint256 rewardId_) external onlyParent {
1082    // Check if the reward ID is valid.
1083    require(_rewards[rewardId_].exists, "Invalid reward ID.");
1084    // Check if the child the reward is assigned to is part of the caller's family group.
1085    require(
1086      _children[_rewards[rewardId_].assignedTo].familyGroup == msg.sender,
1087      "The child this reward is assigned to is not part of your family group."
1088    );
1089    // Check if the reward has been redeemed.
1090    require(_rewards[rewardId_].redeemed, "Reward has not been redeemed yet.");
1091
1092    // Check if the reward redemption has not been approved.
1093    require(
1094      !_rewards[rewardId_].approved,
1095      "Reward redemption has already been approved."
1096    );
1097
1098    // Approve reward redemption.
1099    _rewards[rewardId_].approved = true;
1100    _rewards[rewardId_].approvalDate = block.timestamp;
1101
1102    // Increment rewards approved counter.
1103    rewardsCounter.approved++;
1104
1105    // Emit event.
1106    emit RewardRedemptionApproved(
1107      rewardId_,
1108      msg.sender,
1109      _rewards[rewardId_].approvalDate
1110    );
1111  }
```

*Figure 29: CryptoKids.sol code snippet: approve reward redemption.*

### 3.3.4. Application frontend implementation

The CryptoKids application frontend was carefully designed and implemented to meet the requirements specification for the application:

- All the functional requirements were implemented.
- Authentication mechanisms were implemented to protect pages with user data, which are only accessible after user authentication using MetaMask.
- An informative homepage was implemented to help users understand the application, its functionalities, and its benefits.
- Measurable metrics are displayed on the homepage, such as the number of parents registered, children added, tasks created, tasks completed, rewards created, rewards purchased, tokens earned, and tokens spent. Helping users measure the application's success and quantify its impact.
- Loading indicators are displayed when users are waiting for transactions to be confirmed.

**CryptoKids Dapp source code:**
https://github.com/canedobox/cryptokids-dapp

See below the key functionalities of the CryptoKids application frontend.

#### 3.3.4.1. Wallet connection using MetaMask:

```
48  /**
49   * Connects to MetaMask and get user's account.
50   */
51  const connectionHandler = async () => {
52    // If user is already connected, log out of the application.
53    if (account) {
54      logout();
55    }
56
57    // Check if MetaMask is installed.
58    if (window.ethereum && window.ethereum.isMetaMask) {
59      // Connect using MetaMask and get account.
60      await window.ethereum
61        .request({ method: "eth_requestAccounts" })
62        .then((accounts) => {
63          // Store user account.
64          setAccount(accounts[0]);
65          // Reset some state values.
66          setAccountType(null);
67          setErrorMessage(null);
68          // Initialize the application.
69          appInit();
70        })
71        .catch((error) => {
72          setErrorMessage(error);
73          setAccount(null);
74        });
75    }
76    // If MetaMask is not installed.
77    else {
78      setErrorMessage("Please, install MetaMask.");
79      setAccount(null);
80    }
81  };
```

*Figure 30: App.js code snippet: wallet connection using MetaMask.*

### 3.3.4.2. Listen for network and account changes on MetaMask:

```
// Check if MetaMask is installed.
if (window.ethereum && window.ethereum.isMetaMask) {
  // Set up an event listener for when the network changes on MetaMask.
  window.ethereum.on("chainChanged", () => {
    // Reload the page.
    window.location.reload();
  });
  // Set up an event listener for when the account changes on MetaMask.
  window.ethereum.on("accountsChanged", async () => {
    // Reconnect to MetaMask and get new user's account.
    connectionHandler();
  });
}
```

*Figure 31: App.js code snippet: listen for network and account changes on MetaMask.*

### 3.3.4.3. Establish communication with the smart contract:

```
/**
 * Initialize the application by establishing communication
 * with the contract and fetching initial data.
 */
const appInit = async () => {
  // Start loading dashboard.
  setIsDashboardLoading(true);
  setErrorMessage(null);

  // Get provider.
  const provider = new ethers.providers.Web3Provider(window.ethereum);
  // Get signer.
  const signer = provider.getSigner();

  // Get network.
  const network = await provider.getNetwork();
  // Check if the network is supported.
  if (!contractAddress[network.chainId]) {
    setErrorMessage("Network not supported, connect to Sepolia test network instead.");
    // Stop loading dashboard.
    setIsDashboardLoading(false);
    // Stop the function.
    return;
  }

  // Get contract.
  const contract_ = new ethers.Contract(
    contractAddress[network.chainId].address,
    contractAbi,
    signer
  );
  // Store contract.
  setContract(contract_);

  // Get token symbol.
  const tokenSymbol_ = await contract_.symbol().catch((error) => {setErrorMessage(error);});
  // Store token symbol.
  setTokenSymbol(tokenSymbol_);

  // Get token decimals.
  const tokenDecimals_ = await contract_.decimals().catch((error) => {setErrorMessage(error);});
  // Store token decimals.
  setTokenDecimals(tokenDecimals_);
```

*Figure 32: App.js code snippet: establish communication with the smart contract.*

### 3.3.4.4. Fetch data from the smart contract:

```
/**
 * Fetch data from the contract based on the user's account type:
 * parent or child.
 */
const fetchData = async () => {
  // Start loading data.
  setIsDataLoading(true);
  // Reset data.
  resetData();
  setErrorMessage(null);

  // Check if user is a parent.
  if (contract && account && accountType === "parent") {
    // Get user's family group.
    const familyGroup_ = await contract.getFamilyGroup().catch((error) => {
      setErrorMessage(error);
    });
    // Store family group.
    setFamilyGroup(familyGroup_);

    // Get user's family group tasks.
    const tasks_ = await contract.getFamilyGroupTasks().catch((error) => {
      setErrorMessage(error);
    });
    // Store all tasks.
    setAllTasks(tasks_);

    // Get user's family group rewards.
    const rewards_ = await contract.getFamilyGroupRewards().catch((error) => {
      setErrorMessage(error);
    });
    // Store all rewards.
    setAllRewards(rewards_);
  }
  // Check if user is a child.
  else if (contract && account && accountType === "child") {
    // Get user's tasks.
    const tasks_ = await contract.getChildTasks().catch((error) => {
      setErrorMessage(error);
    });
    // Store all tasks.
    setAllTasks(tasks_);

    // Get user's rewards.
    const rewards_ = await contract.getChildRewards().catch((error) => {
      setErrorMessage(error);
    });
    // Store all rewards.
    setAllRewards(rewards_);

    // Get user's accountBalance.
    const accountBalance_ = await contract
      .balanceOf(account)
      .catch((error) => {
        setErrorMessage(error);
      });
    // Store accountBalance.
    setAccountBalance(accountBalance_);
  }

  // Stop loading data.
  setIsDataLoading(false);
};
```

*Figure 33: App.js code snippet: fetch data from the smart contract.*

### 3.3.4.5. Format a date to a more readable format for children.

```
/**
 * Format a date to a more readable format, making it easier for children to understand.
 * @param {string} date - Date to be format.
 * @param {string} prefix - Prefix to be added to the date.
 * @returns {string} Formatted date.
 */
const formatDate = (date, prefix) => {
  // Formatted date to be returned.
  let formattedDate = "";
  // Get current date.
  const startDate = new Date();
  // Get end date.
  const endDate = new Date(date);

  // Calculate the difference in days.
  const differenceInTime = endDate.getTime() - startDate.getTime();
  const differenceInDays = Math.ceil(
    differenceInTime / (1000 * 60 * 60 * 24)
  );

  // If the date is in the past, return the number of days and date.
  if (differenceInDays < -1) {
    const options = { day: "2-digit", month: "numeric", year: "numeric" };
    const fullDate = new Date(date).toLocaleDateString("en-GB", options);
    // Format the date. Example: "30 days ago, 01/07/2023"
    formattedDate = `${Math.abs(differenceInDays)} days ago, ${fullDate}`;
  }
  // If the date is yesterday.
  else if (differenceInDays === -1) {
    formattedDate = "yesterday";
  }
  // If the date is today.
  else if (differenceInDays === 0) {
    formattedDate = "today";
  }
  // If the date is in tomorrow.
  else if (differenceInDays === 1) {
    formattedDate = "tomorrow";
  }
  // If the date is in the next three days, return the day of the week.
  else if (differenceInDays > 1 && differenceInDays <= 3) {
    // return the day of the week
    const options = { weekday: "long" };
    const fullDate = new Date(date).toLocaleDateString("en-GB", options);
    // Format the date. Example: "this Monday"
    formattedDate = `this ${fullDate}`;
  }
  // If is more than 3 days away, return the number of days and date.
  else if (differenceInDays > 3) {
    const options = { day: "2-digit", month: "numeric", year: "numeric" };
    const fullDate = new Date(date).toLocaleDateString("en-GB", options);
    // Format the date. Example: "in 30 days, 01/09/2023"
    formattedDate = `in ${Math.abs(differenceInDays)} days, ${fullDate}`;
  }

  // Return formatted date.
  return `${prefix} ${formattedDate}`;
};
```

*Figure 34: App.js code snippet: format a date to a more readable format for children.*

### 3.3.4.6. Examples of smart contract calls with and without user input.

The CryptoKids smart contract handles most of the logic of the application, as it is designed to function even without a frontend application. Hence why the smart contract calls made from the frontend application are very similar, only changing a few lines of code in most cases.

See below (Figures 35, 36 and 37) some examples of smart contract calls with and without user input:

```
/**
 * Register a parent to the contract.
 * @param event - Event that triggered the function.
 */
const registerParent = (event) => {
  // Prevent default form submission.
  event.preventDefault();
  // Reset error message.
  setErrorMessage(null);
  // Start loading indicator.
  setIsSignUpPending(true);

  // Call the `registerParent` function on the contract.
  contract
    .registerParent(event.target.parentName.value) // User input.
    .then(async (receipt) => {
      // Wait for the transaction to be mined.
      receipt.wait().then(() => {
        connectionHandler();
        // Stop loading indicator.
        setIsSignUpPending(false);
      });
    })
    .catch((error) => {
      // Set error message.
      setErrorMessage(error);
      // Stop loading indicator.
      setIsSignUpPending(false);
    });
};
```

*Figure 35: WebsiteLayout.js code snippet: example of a smart contract call with user input.*

```
/**
 * Delete parent account from the contract.
 */
const deleteParent = () => {
  // Reset error message.
  setErrorMessage(null);
  // Start loading indicator.
  setIsDeletePending(true);

  // Call the `deleteTask` function on the contract.
  contract
    .deleteParent() // Without user input.
    .then(async (receipt) => {
      // Wait for the transaction to be mined.
      receipt.wait().then(() => {
        utils.logout();
        // Stop loading indicator.
        setIsDeletePending(false);
      });
    })
    .catch((error) => {
      // Set error message.
      setErrorMessage(error);
      // Stop loading indicator.
      setIsDeletePending(false);
    });
};
```

*Figure 36: Sidebar.js code snippet: example of a smart contract call without user input.*

```
/**
 * Add a task to the contract.
 * @param event - Event that triggered the function.
 * @param formRef - Form reference.
 */
const addTask = (event, formRef) => {
  // Prevent default form submission.
  event.preventDefault();
  // Reset error message.
  setErrorMessage(null);
  // Start loading indicator.
  setIsAddEditPending(true);

  // Get the task reward.
  const taskReward = utils.numberToEther(event.target.taskReward.value);

  // Get the task due date.
  let taskDueDate = event.target.taskDueDate.value;
  // If the task due date is not set, set it to 0.
  if (!taskDueDate) {
    taskDueDate = 0;
  }
  // If the task due date is set, convert it to a Unix timestamp.
  else {
    // Get task due date as a Date object.
    taskDueDate = new Date(taskDueDate);
    // Set the task due date to the end of the day.
    taskDueDate.setHours(23, 59, 59);
    // Check if the due date is greater than the current date.
    if (taskDueDate >= new Date()) {
      // Convert the task due date to a Unix timestamp.
      taskDueDate = Math.round(taskDueDate.getTime() / 1000);
    } else {
      // Set the task due date to 0.
      taskDueDate = 0;
    }
  }

  // Call the `addTask` function on the contract.
  contract
    .addTask(
      event.target.childAddress.value,
      event.target.taskDescription.value,
      taskReward,
      taskDueDate
    )
    .then(async (receipt) => {
      // Wait for the transaction to be mined.
      receipt.wait().then(() => {
        utils.fetchData();
        deselectTask(formRef);
        // Stop loading indicator.
        setIsAddEditPending(false);
      });
    })
    .catch((error) => {
      // Set error message.
      setErrorMessage(error);
      // Stop loading indicator.
      setIsAddEditPending(false);
    });
};
```

*Figure 37: Tasks.js code snippet: example of a complex smart contract call with multiple user inputs.*

## 3.4. Graphical User Interface (GUI)

The CryptoKids GUI (Graphical User Interface) was designed to have a clean, modern, and user-friendly interface suitable for children, while also making sure the navigation is intuitive and straightforward. The GUI was implemented based on the following wireframe design:

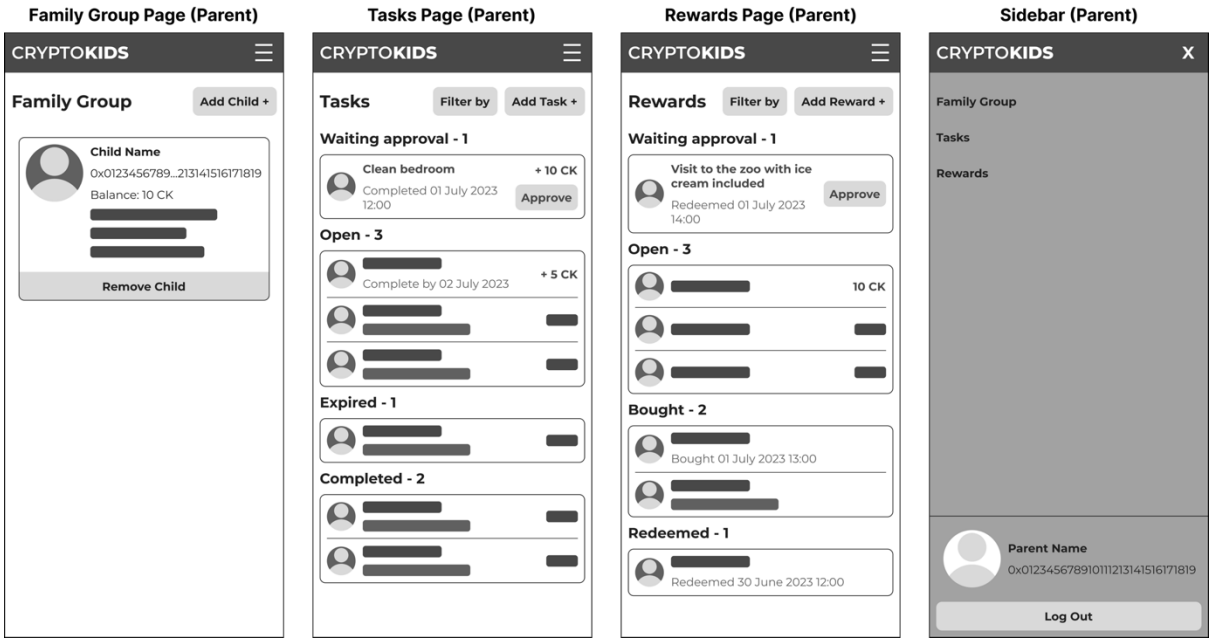
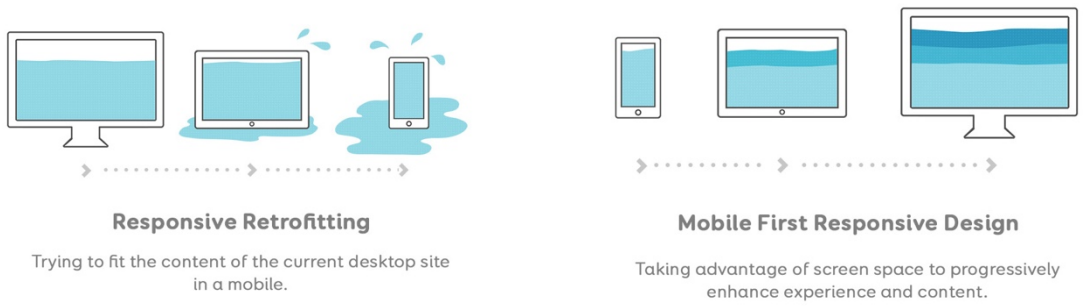



*Figure 38: CryptoKids GUI: dashboard wireframes.*

By applying the Mobile First approach, the GUI was carefully designed to be fully responsive and provide a smooth user experience on any device (such as smartphones, tablets, and computers), allowing parents and children to access the application even on small-screen smartphones.




**Source:** https://stephaniewalter.design/blog/freebies-responsive-retrofitting-vs-mobile-first-responsive-strategy-illustration/

*Figure 39: Mobile first responsive design approach.*

### 3.4.1. Homepage

The homepage GUI was designed to be informative and help users understand the application, its functionalities, and its benefits. Having in mind that most of the users accessing the homepage will be parents looking for a solution to help teach their children about blockchain.

The homepage is divided into four sections:

1) **Hero:** this is the first section a user sees when accessing the application, hence why it is important to provide as much information as possible to try and get the user's attention. This section provides a clear and short message about the application's purpose, a preview of the dashboard being used on different devices, and metrics to help users measure the application's success and quantify its impact.
2) **Getting Started:** this section summarises how the application works into five easy steps.
3) **Benefits:** this section summarises and highlights the benefits of using the application.
4) **Frequently Asked Questions**: this section helps answer questions a user may still have by providing detailed information about the application, its functionalities, and its benefits.

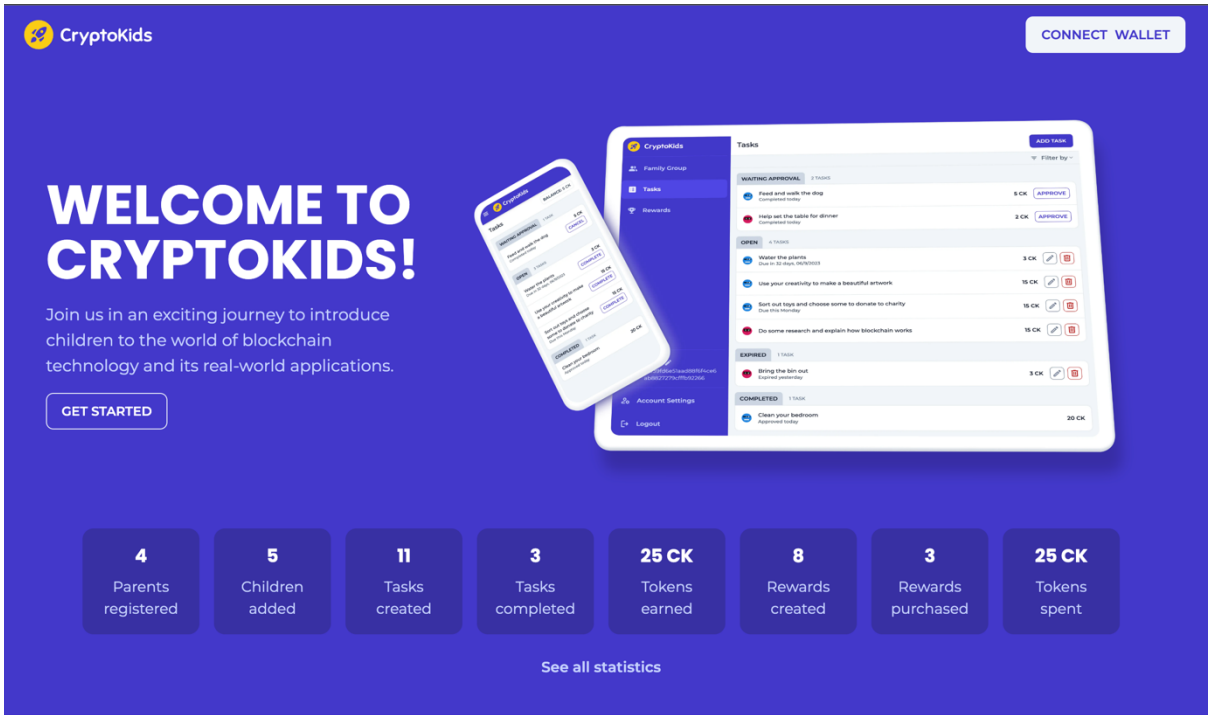



*Figure 40: Homepage GUI: hero section.*

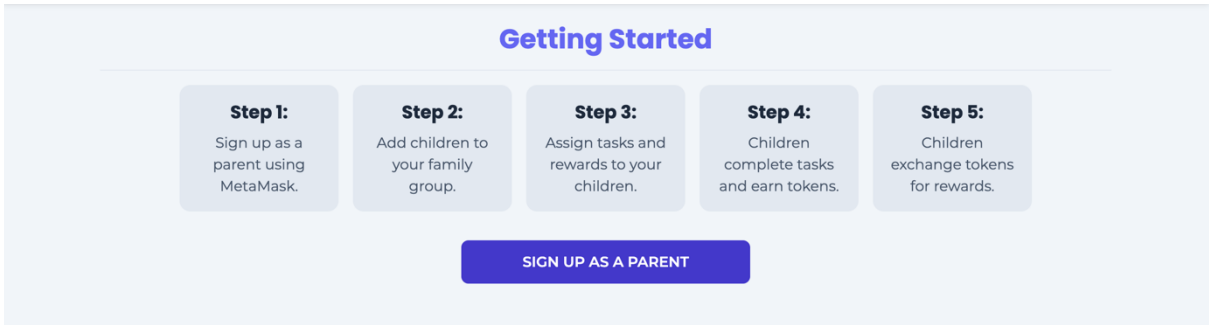



*Figure 41: Homepage GUI: getting started section.*


*Figure 42: Homepage GUI: benefits section.*


*Figure 43: Homepage GUI: frequently asked questions.*

See below (Figure 44) a preview of the homepage responsive design on multiple devices (smartphone, tablet, and computer):

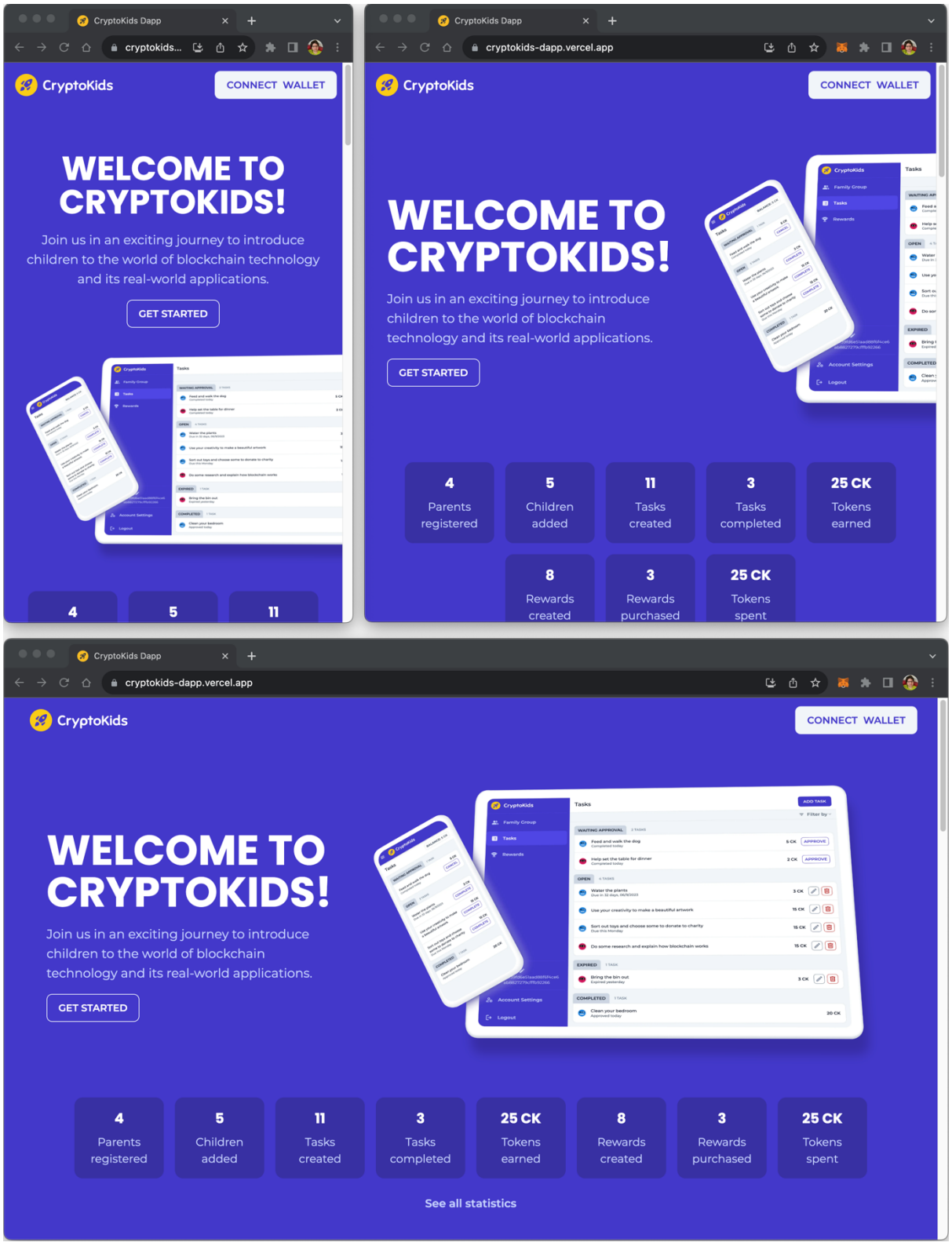



*Figure 44: Homepage GUI: responsive homepage (smartphone, tablet, and computer).*

### 3.4.1. Dashboard

The dashboard GUI was designed to be simple and have a user-friendly interface suitable for children, while also making sure the navigation is intuitive and straightforward.

The navigation sidebar is the most important component within the dashboard. It enables users to navigate between pages, edit the user's profile, and log out of the application.

It is important to mention that:

- The sidebar content is based on the user's account type (parent or child).
- On smaller screens, such as smartphones, the sidebar can be opened using the top navbar.
- On larger screens, the sidebar is fixed.


*Figure 45: Dashboard GUI: top navbar.*

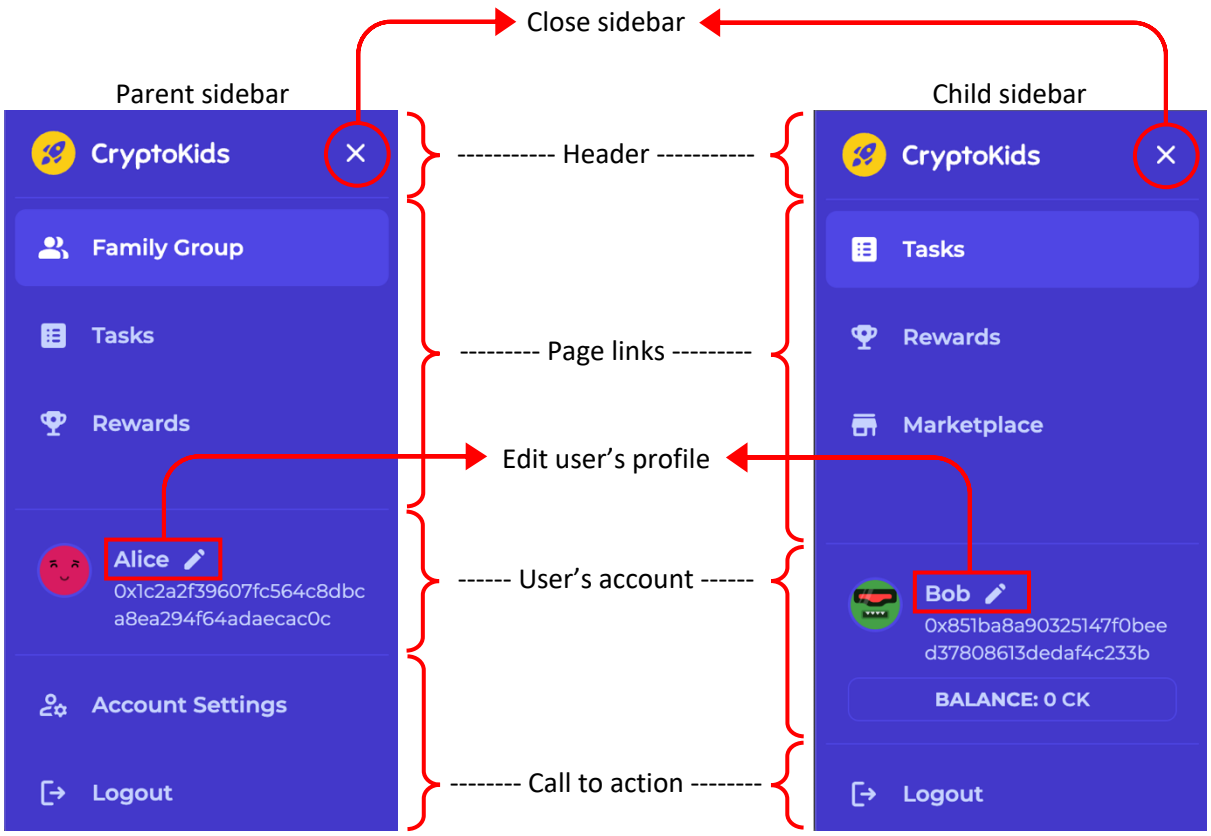



*Figure 46: Dashboard GUI: navigation sidebar.*

It can take a while for a transaction to be confirmed once it is sent to the blockchain. For that reason, loading indicators were implemented to let users know when a transaction is waiting to be confirmed.

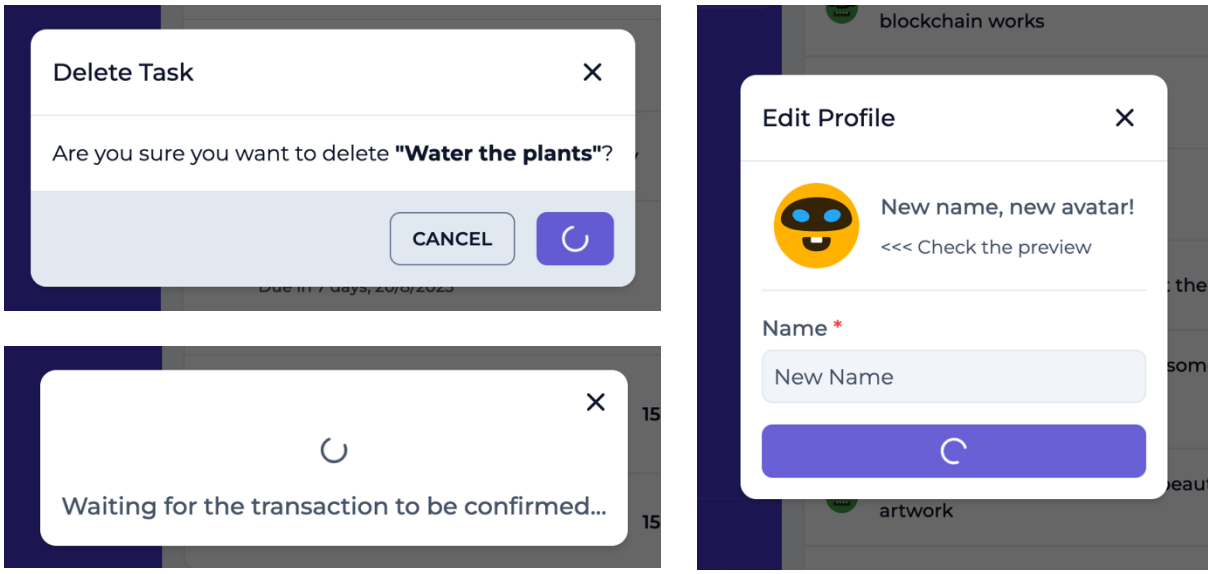



*Figure 47: Dashboard GUI: loading indicators.*

Errors coming from MetaMask can be hard to understand as they are extensive and include all the transaction's details. For that reason, the errors are formatted to a more readable format, extracting only the short error message and displaying it as a title, making it easier for children to understand.

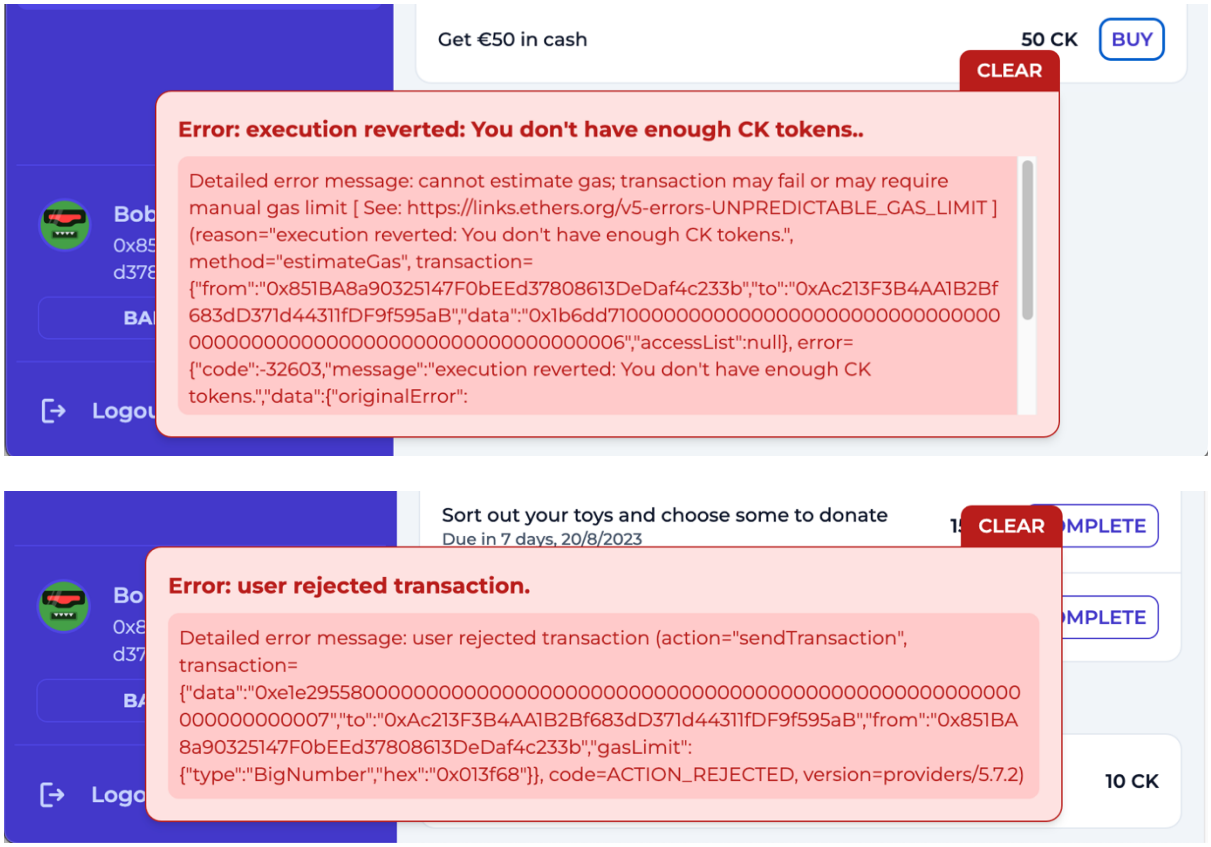



*Figure 48: Dashboard GUI: error messages.*

See below (Figure 49) a preview of the parent dashboard responsive design on multiple devices (smartphone, tablet, and computer):

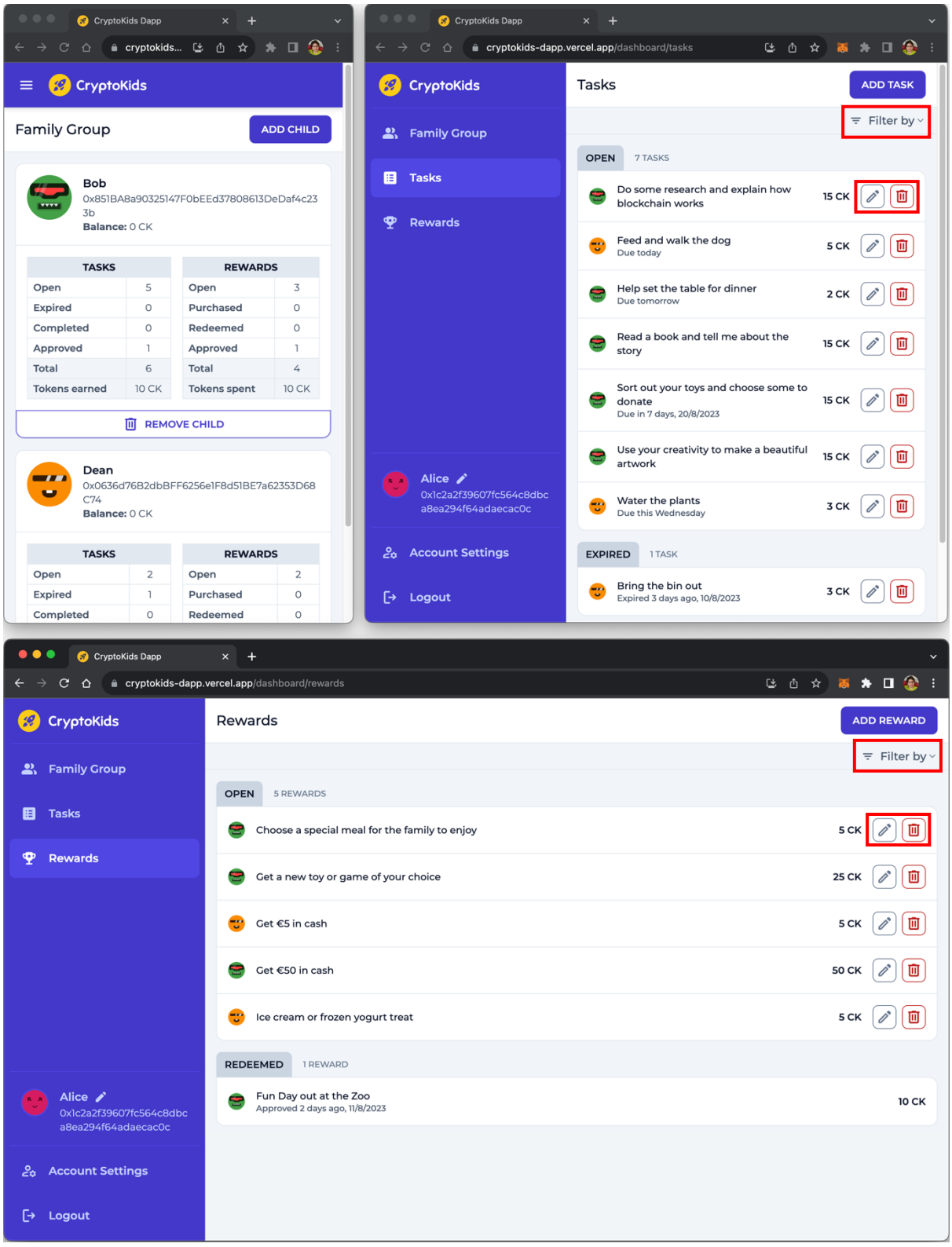



*Figure 49: Dashboard GUI: responsive parent dashboard (smartphone, tablet, and computer).*

As you can see highlighted above (Figure 49), the parent dashboard has some special functionalities, which allow tasks and rewards to be edited, deleted, or filtered by children.

See below (Figure 50) a preview of the child dashboard responsive design on multiple devices (smartphone, tablet, and computer):

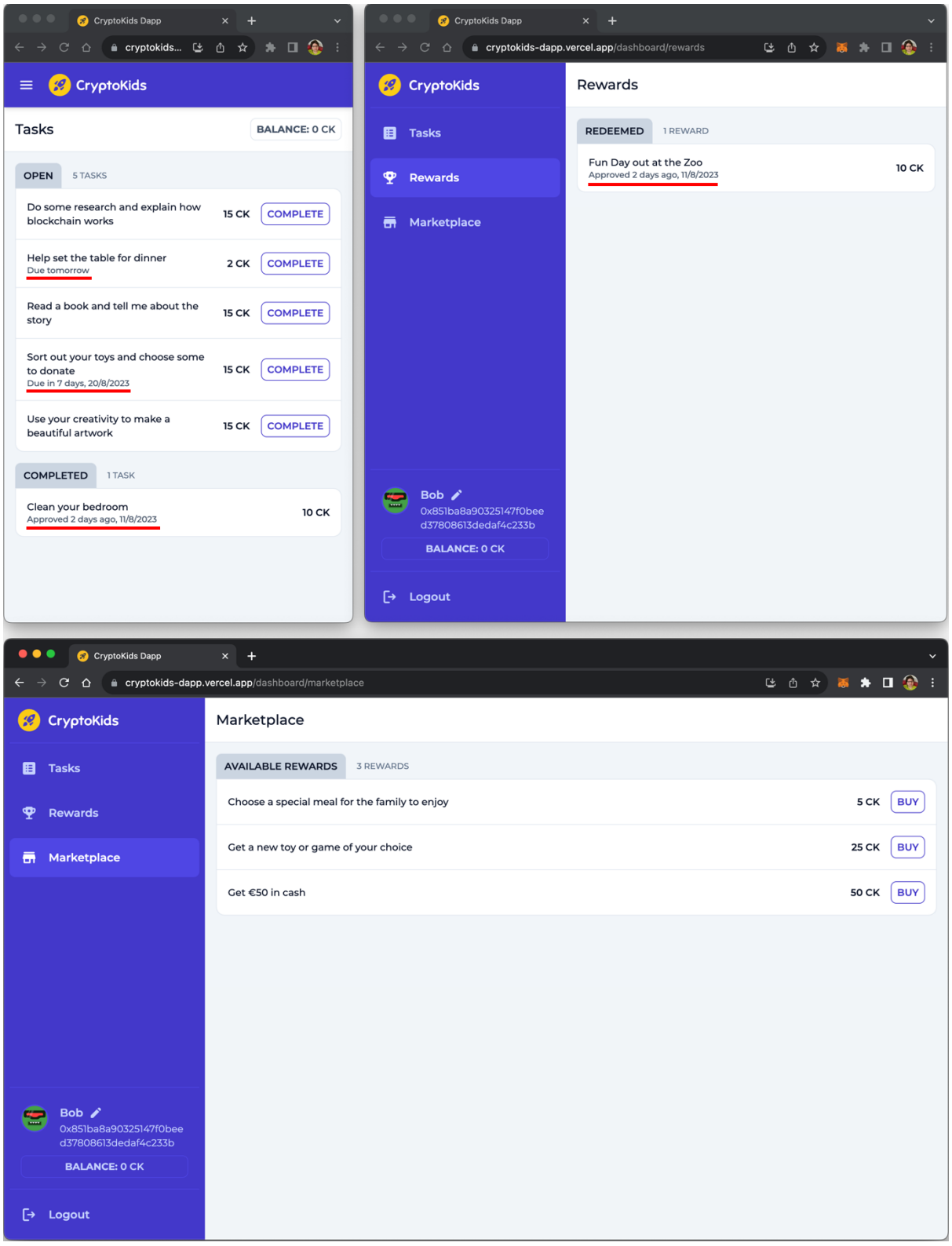



*Figure 50: Dashboard GUI: responsive child dashboard (smartphone, tablet, and computer).*

As you can see highlighted above (Figure 50), all the dates are formatted to a more readable format, making it easier for children to understand.

## 3.5. Deployment

### 3.5.1. Smart contract deployment

The CryptoKids smart contract was deployed to the Sepolia Ethereum Testnet using Hardhat.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

canedobox@Canedos-iMac cryptokids-dapp % npm run hardhat-deploy sepolia

> cryptokids-dapp@0.1.0 hardhat-deploy
> cd backend && npm run deploy sepolia


> cryptokids-dapp-backend@0.1.0 deploy
> hardhat run scripts/deploy.js --network sepolia

CryptoKids smart contract deployed to the sepolia network.
Network ChainID: 11155111
Contract address: 0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB
Token name: CryptoKids
Token symbol: CK
canedobox@Canedos-iMac cryptokids-dapp %
```

*Figure 51: Terminal: CryptoKids smart contract deployment using Hardhat.*

**CryptoKids smart contract address:**
0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB

**Deployment transaction link:**
https://sepolia.etherscan.io/tx/0xa5f30b61e78b4b2b150b2f6aaabc2a44b61ecd2f29b2d6c724570c37009afcbf

The CryptoKids smart contract was verified on Etherscan using Hardhat. This is to make the smart contract source code public and allow users to interact with it directly from Etherscan.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS  zsh

canedobox@Canedos-iMac cryptokids-dapp % npm run hardhat-verify sepolia 0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB "Crypto
Kids" "CK"


> cryptokids-dapp@0.1.0 hardhat-verify
> cd backend && npm run verify sepolia 0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB CryptoKids CK


> cryptokids-dapp-backend@0.1.0 verify
> hardhat verify --contract "contracts/CryptoKids.sol:CryptoKids" --network sepolia 0xAc213F3B4AA1B2Bf683dD371d44311fDF9f5
95aB CryptoKids CK

Successfully submitted source code for contract
contracts/CryptoKids.sol:CryptoKids at 0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB
for verification on the block explorer. Waiting for verification result...

Successfully verified contract CryptoKids on the block explorer.
https://sepolia.etherscan.io/address/0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB#code
canedobox@Canedos-iMac cryptokids-dapp %
canedobox@Canedos-iMac cryptokids-dapp %
```

*Figure 52: Terminal: CryptoKids smart contract verification using Hardhat.*

**CryptoKids verified smart contract link:**
https://sepolia.etherscan.io/address/0xAc213F3B4AA1B2Bf683dD371d44311fDF9f595aB#code

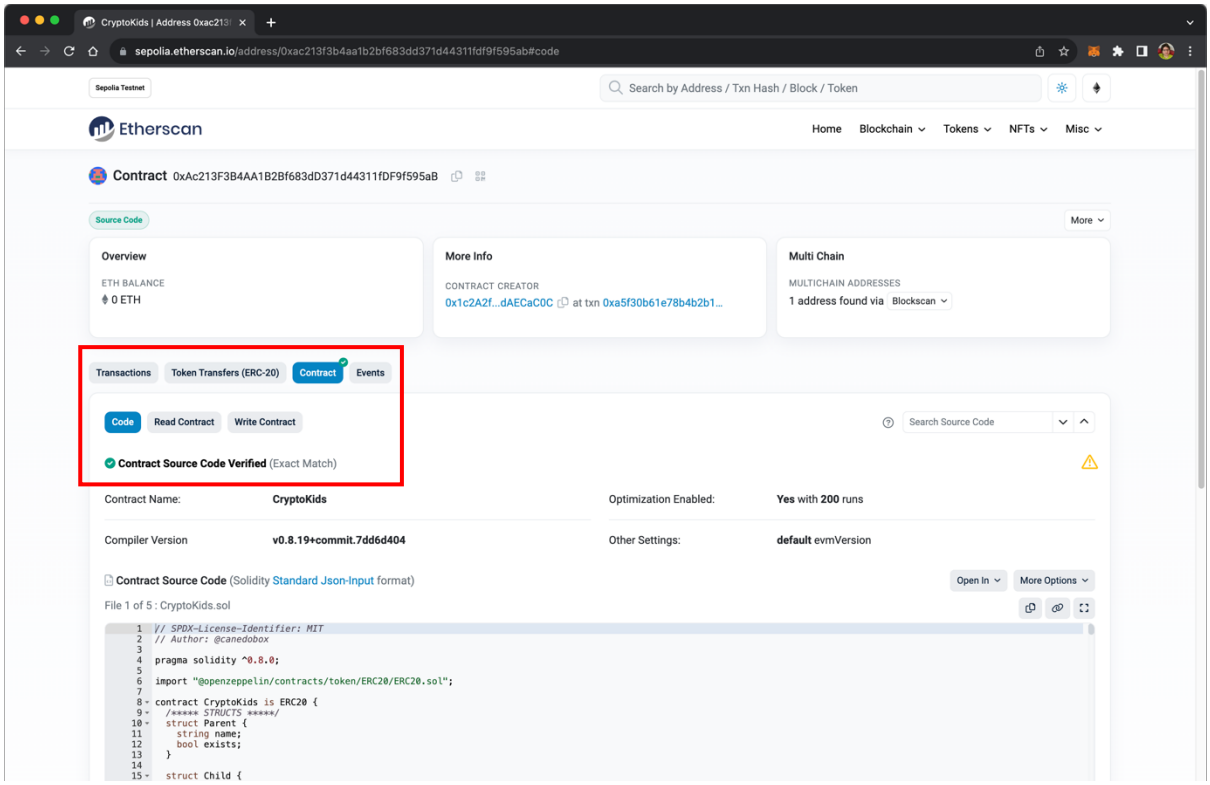



*Figure 53: CryptoKids smart contract verified on Etherscan.*

Hardhat makes the process of smart contract deployment very simple. All that is required is to add the networks you want to use to the Hardhat configuration file and create a deployment script.

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config({ path: ".env.local", override: true });

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: {
    version: "0.8.19",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200
      }
    }
  },
  networks: {
    localhost: {
      url: "http://127.0.0.1:8545",
      chainId: 31337
    },
    sepolia: {
      chainId: 11155111,
      url: `https://sepolia.infura.io/v3/${process.env.INFURA_API_KEY}`,
      accounts: [process.env.ACCOUNT_PRIVATE_KEY]
    }
  },
  etherscan: {
    apiKey: process.env.ETHERSCAN_API_KEY
  }
};
```

*Figure 54: hardhat.config.js: Hardhat configuration file.*

In order to connect to the Ethereum blockchain and deploy a smart contract, access to an Ethereum node is required. An Ethereum node is a computer running the Ethereum client software, which is connected to other computers running the same software, forming the Ethereum blockchain network.

As you can see above (Figure 54), Infura was the node provider used to deploy the smart contract to the Sepolia network. Infura is a very reliable node service provider, that allows developers to interact with the blockchain without running their own infrastructure. Infura provides APIs (Application Programming Interfaces) that simplify tasks such as sending transactions, reading data from the blockchain, and deploying smart contracts.

Once the Sepolia network was added to the Hardhat configuration file, the next step was to implement the deployment script, which enables the process to be automated and extra functionality to be executed every time a smart contract is deployed.

```
1  const { ethers, network, artifacts } = require("hardhat");
2  const fs = require("fs");
3  const path = require("path");
4
5  // Set constants for contract details.
6  const CONTRACT_NAME = "CryptoKids";
7  const TOKEN_NAME = "CryptoKids";
8  const TOKEN_SYMBOL = "CK";
9
10 /**
11  * Main.
12  */
13 async function main() {
14   // Deploy contract.
15   const contract = await ethers.deployContract(CONTRACT_NAME, [
16     TOKEN_NAME,
17     TOKEN_SYMBOL
18   ]);
19   await contract.waitForDeployment();
20
21   // Display contract deployment message.
22   console.log(
23     `${CONTRACT_NAME} smart contract deployed to the ${network.name} network.`
24   );
25   console.log(`Network ChainID: ${network.config.chainId}`);
26   console.log(`Contract address: ${contract.target}`);
27   console.log(`Token name: ${TOKEN_NAME}`);
28   console.log(`Token symbol: ${TOKEN_SYMBOL}`);
29
30   // Save contract files.
31   saveContractFiles(contract);
32 }
```

*Figure 55: deploy.js: Smart contract deployment script.*

In the CryptoKids deployment script, a function was implemented to save the contract address and ABI (Application Binary Interface) into separate files in the frontend directory every time the contract was deployed (see Figures 56 and 57).

```
/**
 * Save contract address and ABI files in the frontend directory.
 */
function saveContractFiles(contract_) {
  // Set frontend directory path.
  const frontendDirPath = path.join(__dirname, "../../src/contracts");

  // Create directory if it does not exist.
  if (!fs.existsSync(frontendDirPath)) {
    fs.mkdirSync(frontendDirPath);
  }

  /***** CONTRACT ABI *****/
  // Get contract artifacts.
  const artifact = artifacts.readArtifactSync(CONTRACT_NAME);

  // Save contract ABI file.
  fs.writeFileSync(
    path.join(frontendDirPath, `${CONTRACT_NAME}-abi.json`),
    JSON.stringify(artifact.abi, null, 2)
  );

  /***** CONTRACT ADDRESS *****/
  // Set address file path.
  const addressFilePath = path.join(
    frontendDirPath,
    `${CONTRACT_NAME}-address.json`
  );

  // Create address file variable.
  let addressFile = {};
  // Get address file if it exists.
  if (fs.existsSync(addressFilePath)) {
    addressFile = require(addressFilePath);
  }

  // Store contract address using the network chain ID.
  addressFile[network.config.chainId] = {
    name: network.name,
    address: contract_.target
  };

  // Save contract address file.
  fs.writeFileSync(
    path.join(addressFilePath),
    JSON.stringify(addressFile, null, 2)
  );
}
```

*Figure 56: deploy.js: save contract address and ABI after deployment.*

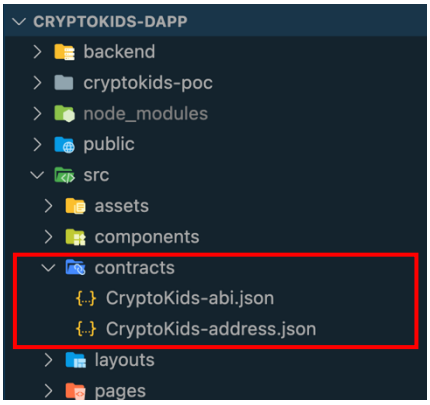



*Figure 57: Contract address and ABI files.*

### 3.5.2. Application frontend deployment

The CryptoKids application frontend was deployed and hosted online using Vercel.

**CryptoKids Dapp link:**
https://cryptokids-dapp.vercel.app/

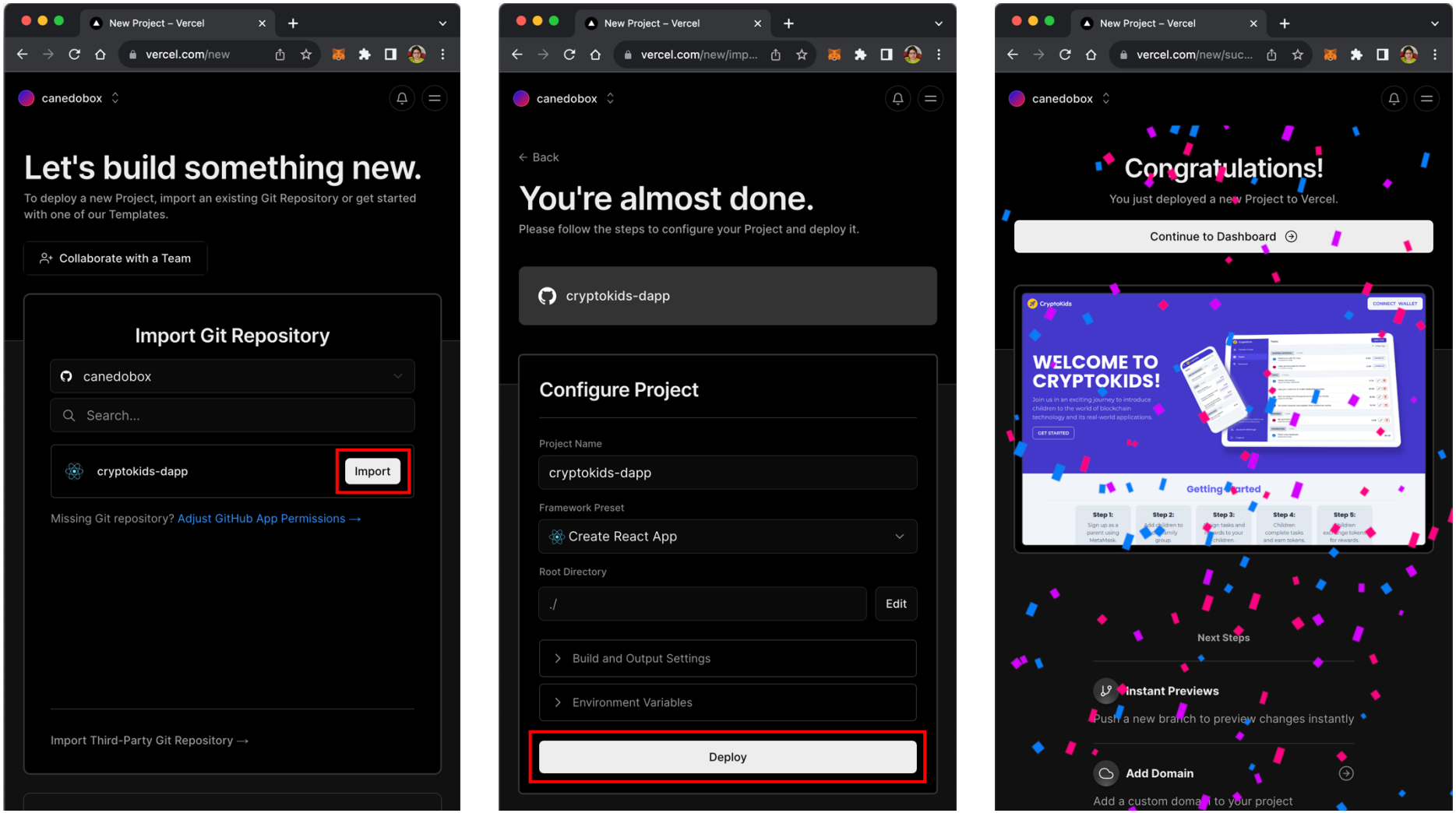



*Figure 58: CryptoKids application frontend deployment using Vercel.*

As you can see above (Figure 58), deploying the CryptoKids application frontend using Vercel was very simple, with only a few steps the application was deployed to the cloud and available online. All that is required is to import the git repository from GitHub, configure the project and deploy the application.

Apart from the easy setup and deployment, below are some other reasons why Vercel was used:

- Small projects are hosted on Vercel for free, with the option to scale for a small cost.
- Vercel provides a free customizable link (domain) with HTTPS (Hypertext Transfer Protocol Secure) connection by default, no extra configuration is needed.
- Vercel listens to any changes (commits) made to the git repository on GitHub and automatically re-deploys the application, removing that extra step and making any changes available and live in a few seconds.
- Vercel offers an amazing dashboard which includes web analytics and speed insights, which are essential for monitoring the application, discovering performance issues early, and measuring the success and impact of the application.

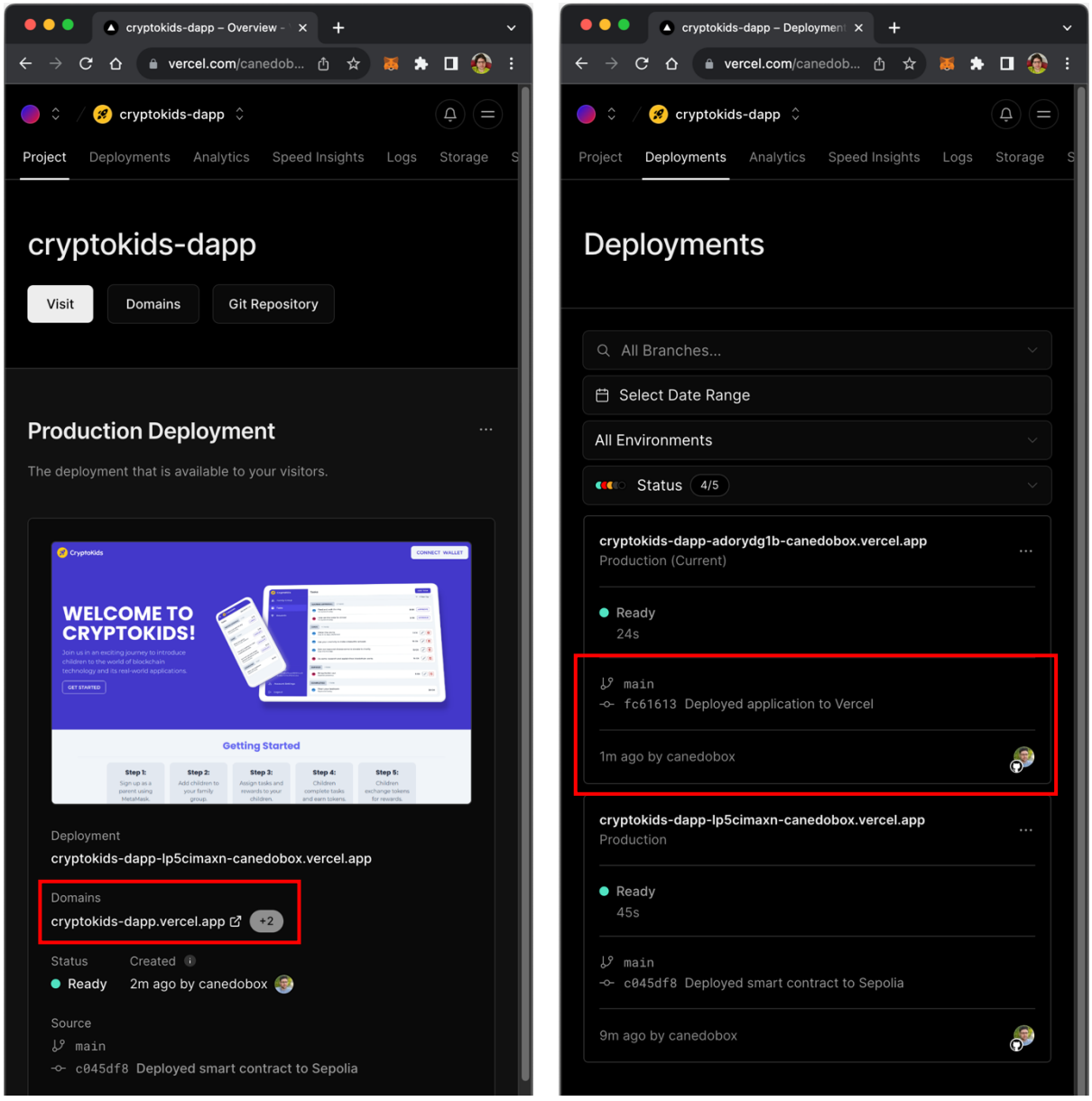



*Figure 59: Vercel dashboards: Project Overview and Deployments History.*

The Vercel web analytics dashboard provides detailed insights into application visitors, including metrics like top pages visited, demographics, operating systems and web browsers used. Those metrics help to measure the success and impact of the application.

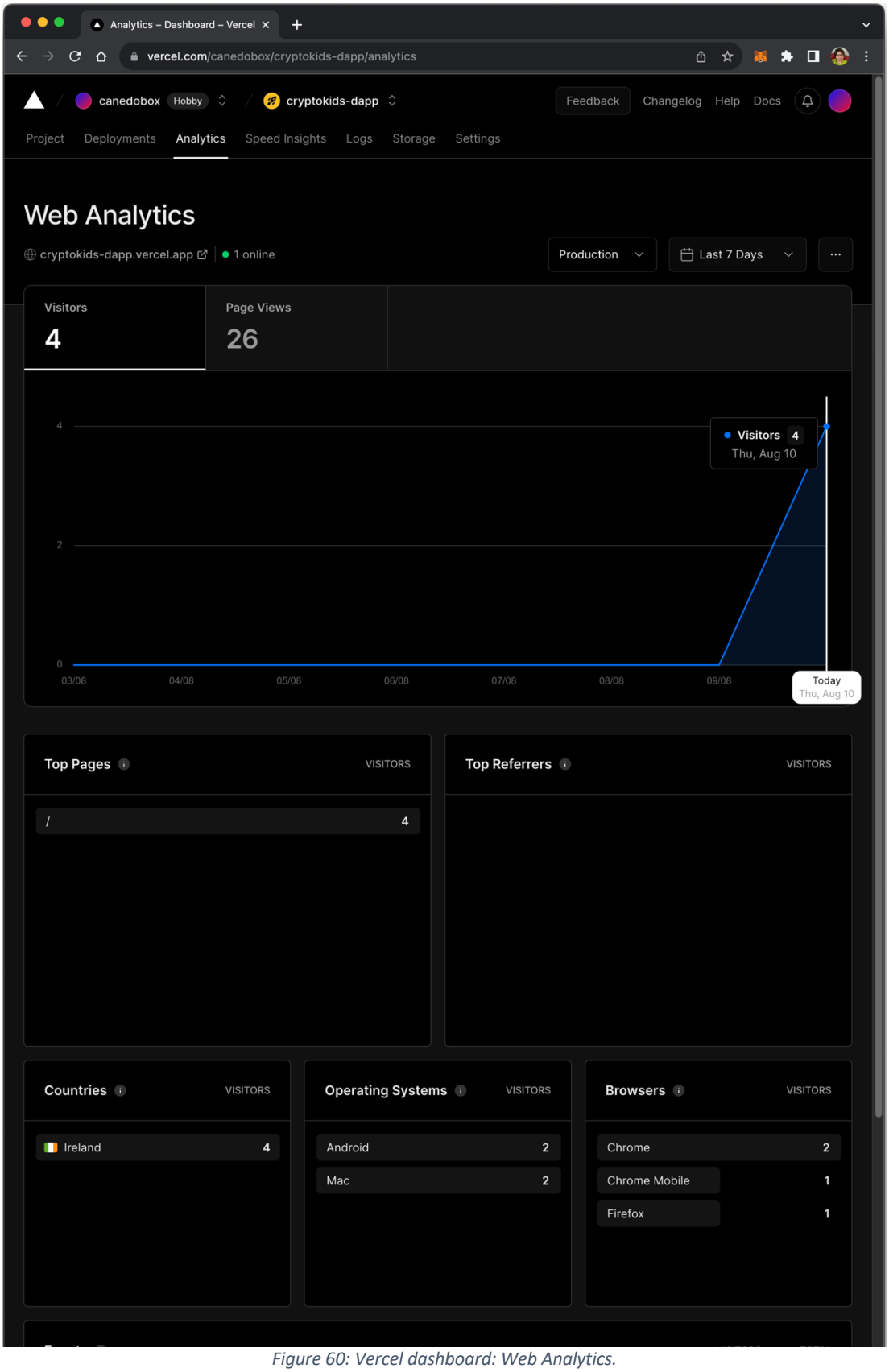



*Figure 60: Vercel dashboard: Web Analytics.*

The Vercel speed insights dashboard provides a view of the application's performance metrics based on visitors' data, facilitating informed decisions for its optimization. Those metrics help monitor the application and discover performance issues early.

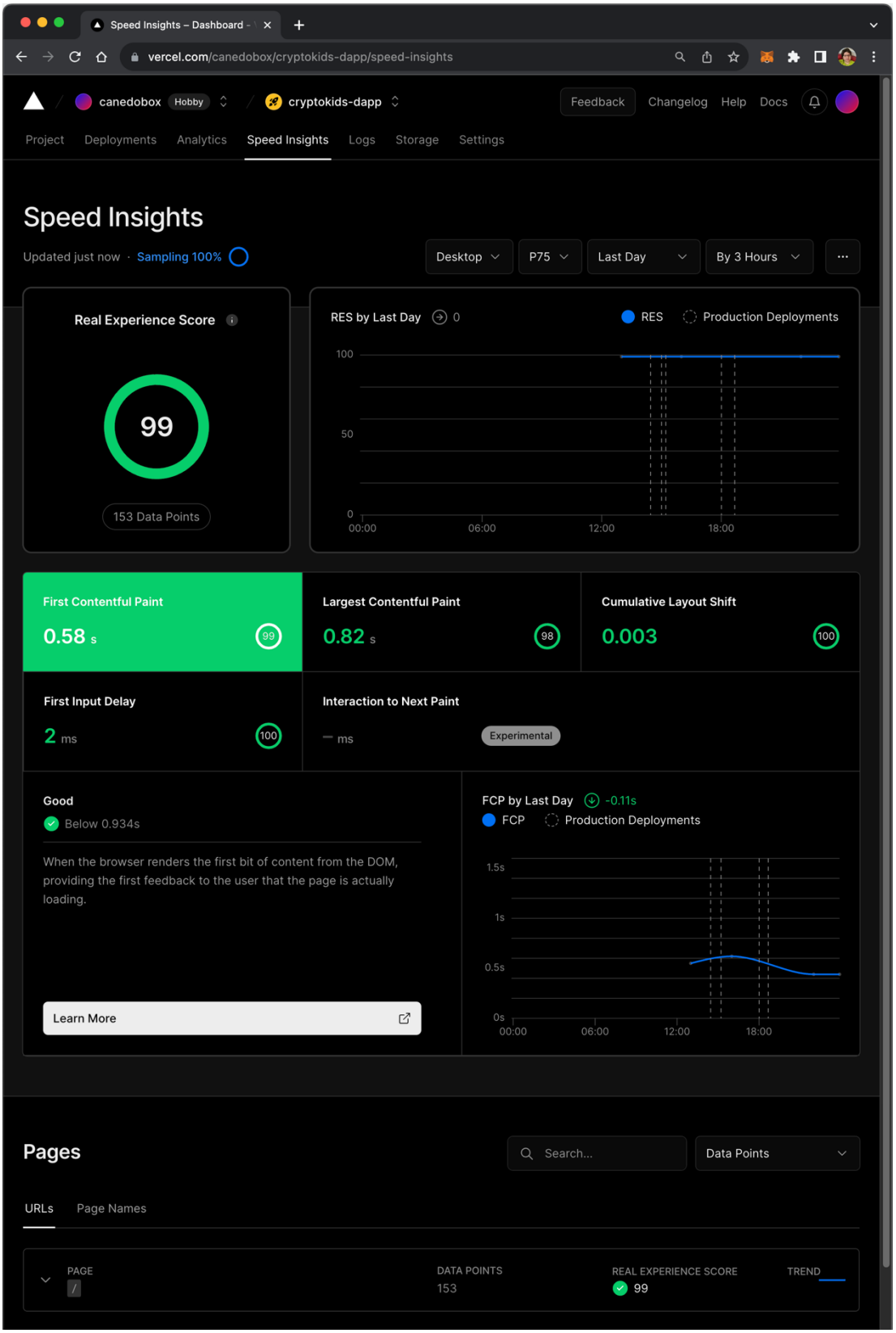



*Figure 61: Vercel dashboard: Speed Insights.*

## 3.6. Testing and Evaluation

The CryptoKids Dapp was continuously tested during the entire process of development and after deployment, making sure the application is secure, free of errors or bugs, and that it works as intended.

By continuously testing the smart contract and application during development, minor issues were resolved straight away before the code was committed to GitHub, avoiding major issues to develop. Only four issues were found (see Figure 62) after the code was committed to GitHub: one related to the smart contract, two related to the frontend application, and one related to website performance. The issues were minor and were resolved successfully.

For more details, please, visit the “issues” section on the project’s GitHub repository: https://github.com/canedobox/cryptokids-dapp/issues?q=is%3Aissue+is%3Aclosed

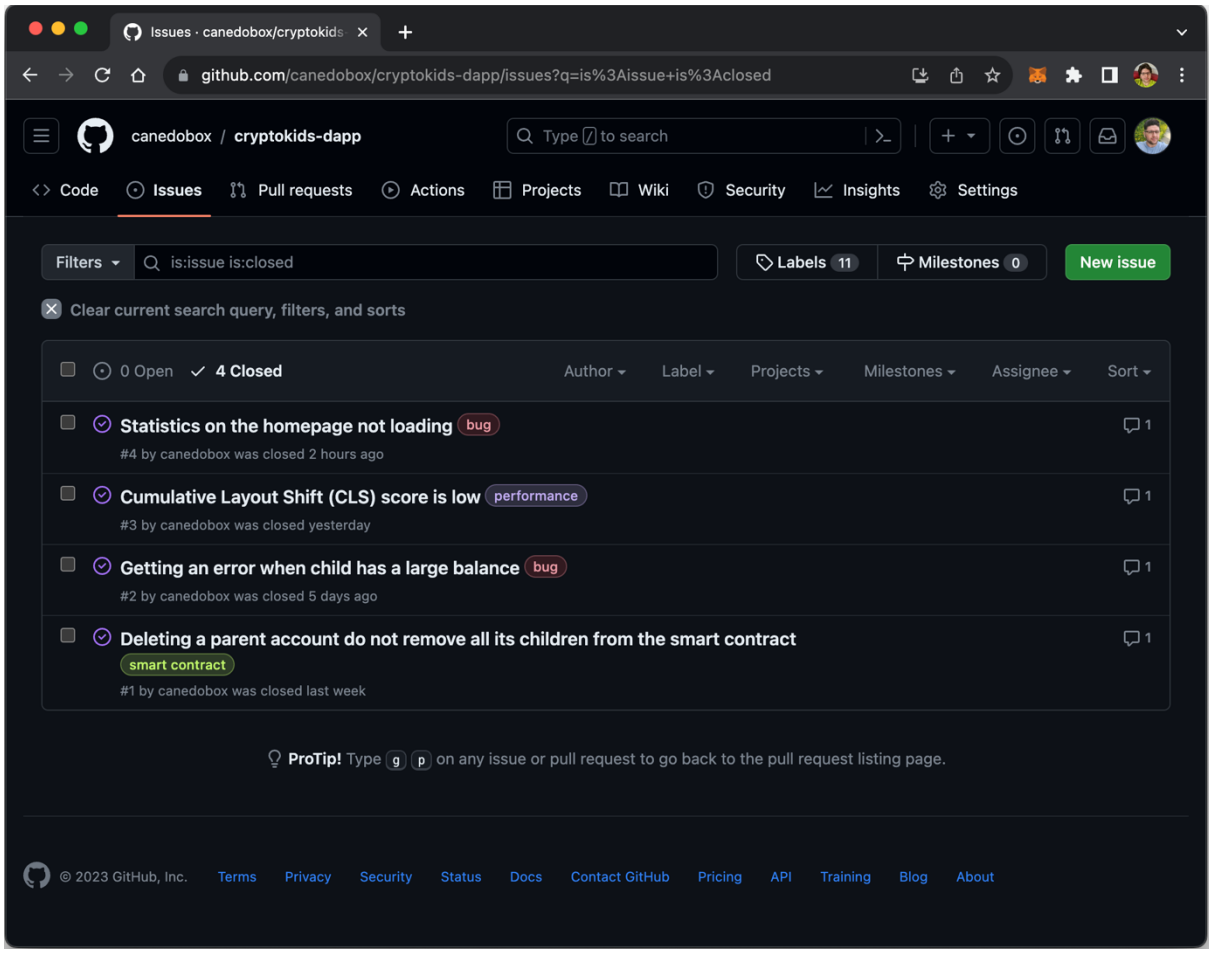



*Figure 62: CryptoKids Dapp "Issues" section on GitHub.*

### 3.6.1. Smart contract testing

Smart contracts are immutable pieces of code, meaning that no changes can be made to a smart contract after deployment. Hence why it is crucial to test a smart contract thoroughly, making sure it is secure and free of errors before it is deployed.

The CryptoKids smart contract testing was fully automated using Hardhat. As you can see below (Figures 63 and 64), there were 120 automated tests implemented for the CryptoKids smart contract, making sure every single function and component of the smart contract was fully tested before deployment.

**CryptoKids smart contract test file source code:**
https://github.com/canedobox/cryptokids-dapp/blob/main/backend/test/CryptoKids.js

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

canedobox@Canedos-iMac cryptokids-dapp % npm run hardhat-test

> cryptokids-dapp@0.1.0 hardhat-test
> cd backend && npm run test


> cryptokids-dapp-backend@0.1.0 test
> hardhat test



  CryptoKids Smart Contract Automated Tests
    DEPLOYMENT TESTS
      ✔ Sets the token name. (2011ms)
      ✔ Sets the token symbol.
    PARENT TESTS
      Register a parent
        ✔ Should fail if caller is already registered. (55ms)
        ✔ Should register a parent.
        ✔ Registering a parent should emit an event.
      Delete a parent
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should delete a parent and its family group. (42ms)
        ✔ Deleting a parent should emit an event.
    FAMILY GROUP TESTS
      Add a child to a family group
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should fail if child is already registered.
        ✔ Should add a child to a family group. (126ms)
        ✔ Adding a child to a family group should emit an event.
      Remove a child from a family group
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should fail if address does not belong to a child.
        ✔ Should fail if child is not part of the caller's family group.
        ✔ Should remove a child from a family group. (69ms)
        ✔ Removing a child from a family group should emit an event.
      Get a family group
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should get a family group. (49ms)
    ACCOUNT TESTS
      Get account type
        ✔ Caller should not be registered.
        ✔ Caller should be registered as a parent.
        ✔ Caller should be registered as a child.
      Edit a profile
        ✔ Should fail if caller is not registered.
        ✔ Should adit a parent's profile.
        ✔ Editing a parent's profile should emit an event.
        ✔ Should edit a child's profile.
        ✔ Editing a child's profile should emit an event.
    TASK TESTS
      Add a task
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should fail if address does not belong to a child.
        ✔ Should fail if child is not part of the caller's family group.
        ✔ Should add a task. (66ms)
        ✔ Adding a task should emit an event.
      Edit a task
        ✔ Should fail if caller is not registered as a parent.
        ✔ Should fail if task ID is not valid.
        ✔ Should fail if child is not part of the caller's family group.
```

*Figure 63: Terminal: smart contract testing using Hardhat (start).*

```
    Cancel task completion
      ✔ Should fail if caller is not registered as a child.
      ✔ Should fail if task ID is not valid.
      ✔ Should fail if task is not assigned to the caller.
      ✔ Should fail if task has not been completed.
      ✔ Should fail if task completion has already been approved.
      ✔ Should cancel task completion.
      ✔ Cancelling task completion should emit an event.
    Approve task completion
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should fail if task ID is not valid.
      ✔ Should fail if child is not part of the caller's family group.
      ✔ Should fail if task has not been completed.
      ✔ Should fail if task completion has already been approved.
      ✔ Should approve task completion.
      ✔ Should send reward tokens to the child's wallet.
      ✔ Approving task completion should emit an event.
    Get child's tasks.
      ✔ Should fail if caller is not registered as a child.
      ✔ Should get child's tasks.
    Get caller's family group tasks.
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should get caller's family group tasks.
  REWARD TESTS
    Add a reward
      ✔ Should fail if caller not registered as a parent.
      ✔ Should fail if address does not belong to a child.
      ✔ Should fail if child is not part of the caller's family group.
      ✔ Should add a reward. (63ms)
      ✔ Adding a reward should emit an event.
    Edit a reward
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if child is not part of the caller's family group.
      ✔ Should fail if reward has already been purchased.
      ✔ Should edit a reward.
      ✔ Editing a reward should emit an event.
    Delete a reward
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if child is not part of the caller's family group.
      ✔ Should fail if reward has already been purchased.
      ✔ Should delete a reward. (57ms)
      ✔ Deleting a reward should emit an event.
    Purchase a reward
      ✔ Should fail if caller is not registered as a child.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if reward is not assigned to the caller.
      ✔ Should fail if reward has already been purchased.
      ✔ Should fail if caller does not have enough tokens.
      ✔ Should purchase a reward. (40ms)
      ✔ Should spend tokens from the child's wallet.
      ✔ Purchasing a reward should emit an event.
    Redeem a reward
      ✔ Should fail if caller is not registered as a child.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if reward is not assigned to the caller.
      ✔ Should fail if reward has not been purchased.
      ✔ Should fail if reward has already been redeemed.
      ✔ Should redeem a reward.
      ✔ Redeeming a reward should emit an event.
    Cancel reward redemption
      ✔ Should fail if caller is not registered as a child.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if reward is not assigned to the caller.
      ✔ Should fail if reward has not been redeemed.
      ✔ Should fail if reward redemption has already been approved.
      ✔ Should cancel reward redemption. (40ms)
      ✔ Cancelling reward redemption should emit an event.
    Approve reward redemption
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should fail if reward ID is not valid.
      ✔ Should fail if child is not part of the caller's family group.
      ✔ Should fail if reward has not been redeemed.
      ✔ Should fail if reward redemption has already been approved.
      ✔ Should approve reward redemption.
      ✔ Approving reward redemption should emit an event.
    Get child's rewards.
      ✔ Should fail if caller is not registered as a child.
      ✔ Should get child's rewards.
    Get caller's family group rewards.
      ✔ Should fail if caller is not registered as a parent.
      ✔ Should get caller's family group rewards.

  120 passing (4s)

canedobox@Canedos-iMac cryptokids-dapp %
```

*Figure 64: Terminal: smart contract testing using Hardhat (end).*

See below (Figure 65) an example of automated tests for the “registerParent” function of the smart contract:

```
/***** PARENT TESTS *****/
describe("PARENT TESTS", () => {
  describe("Register a parent", () => {
    it("Should fail if caller is already registered.", async () => {
      const { contract, parent1, child1 } = await loadFixture(
        deployCryptoKidsFixture
      );
      // Try registering a parent with an account that is already registered as a parent.
      await expect(
        contract.connect(parent1).registerParent("Name")
      ).to.be.revertedWith("You are already registered as a parent.");
      // Try registering a parent with an account that is already registered as a child.
      await expect(
        contract.connect(child1).registerParent("Name")
      ).to.be.revertedWith("You are already registered as a child.");
    });

    it("Should register a parent.", async () => {
      const { contract, notRegistered } = await loadFixture(
        deployCryptoKidsFixture
      );
      // Get current accounts counter.
      const accountsCounter = await contract.accountsCounter();
      // Register a parent.
      await contract.connect(notRegistered).registerParent("Name");
      // Get account type.
      const [accountType] = await contract
        .connect(notRegistered)
        .getProfile();
      // Check account type.
      expect(accountType).to.equal("parent");
      // Get new accounts counter.
      const newAccountsCounter = await contract.accountsCounter();
      // Check number of parents registered.
      expect(newAccountsCounter.parentsRegistered).to.be.greaterThan(
        accountsCounter.parentsRegistered
      );
    });

    it("Registering a parent should emit an event.", async () => {
      const { contract, notRegistered } = await loadFixture(
        deployCryptoKidsFixture
      );
      // Register a parent and check event emitted.
      await expect(contract.connect(notRegistered).registerParent("Name"))
        .to.emit(contract, "ParentRegistered")
        .withArgs(notRegistered.address);
    });
  });
```

*Figure 65: CryptoKids.js: smart contract tests for the “registerParent” function.*

### 3.6.2. User testing

The user testing was designed to cover the key functionalities of the CryptoKids Dapp:

- Navigate to the CryptoKids Dapp homepage: https://cryptokids-dapp.vercel.app/
- Connect to the application using MetaMask.
- Sign up as a parent.
- As a parent: add a child to a family group.
- As a parent: add a task.
- As a child: complete a task.
- As a parent: approve task completion.
- As a parent: add a reward.
- As a child: purchase a reward.
- As a child: redeem a reward.
- As a parent: approve reward redemption.

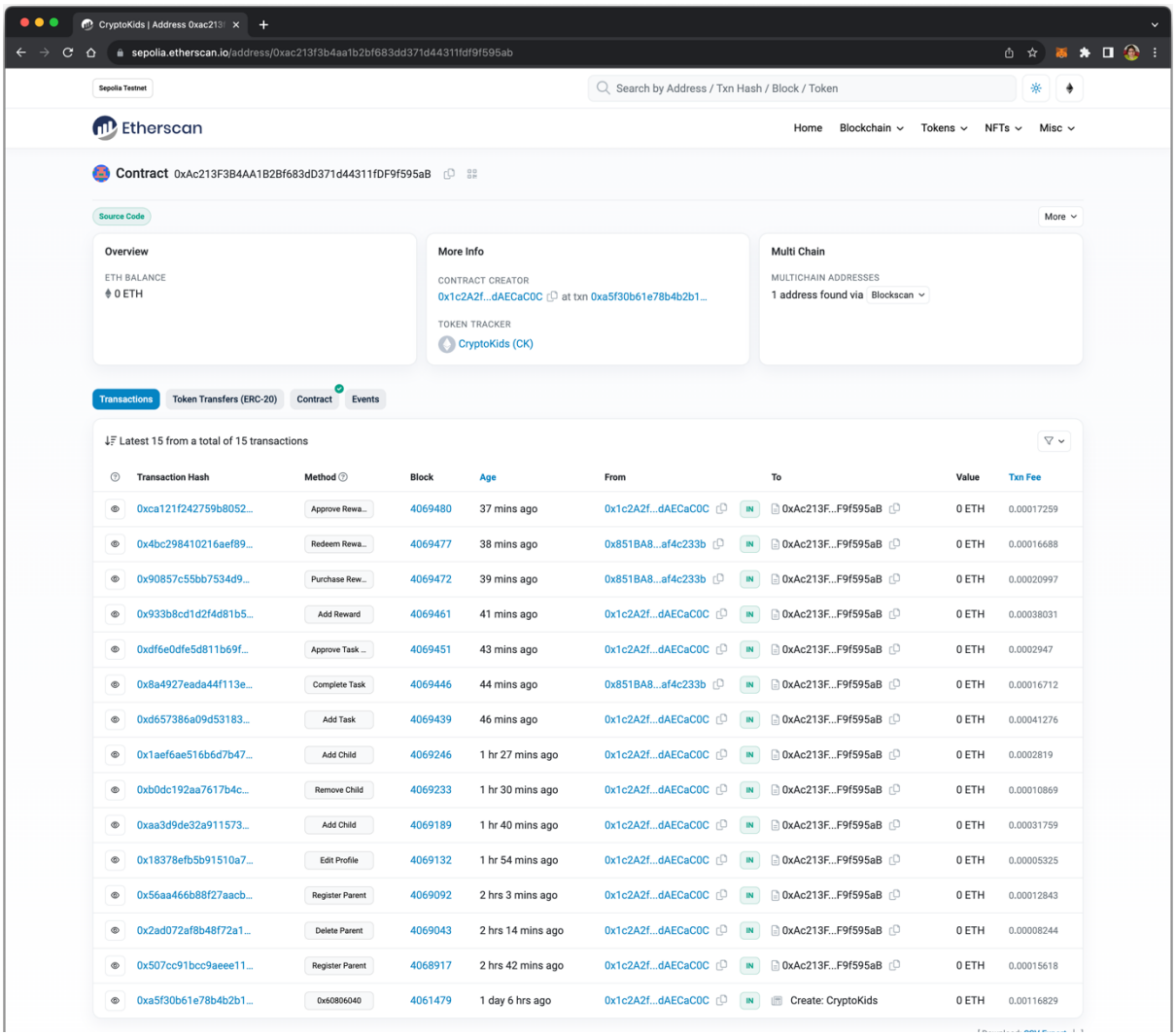



*Figure 66: CryptoKids smart contract transactions after user testing.*

All those tests were executed successfully multiple times, and without any errors, during the development process and after the deployment. See below a detailed step-by-step of how each of those tests was performed.

### 3.6.2.1. Connect to the application using MetaMask:

- **Step 1:** user clicks on the “Connect Wallet” button.
- **Step 2:** user selects the account(s) to connect with MetaMask and clicks on the “Next” button.
- **Step 3:** user clicks on the “Connect” button on MetaMask.
- **Step 4:** the user is connected to the application.

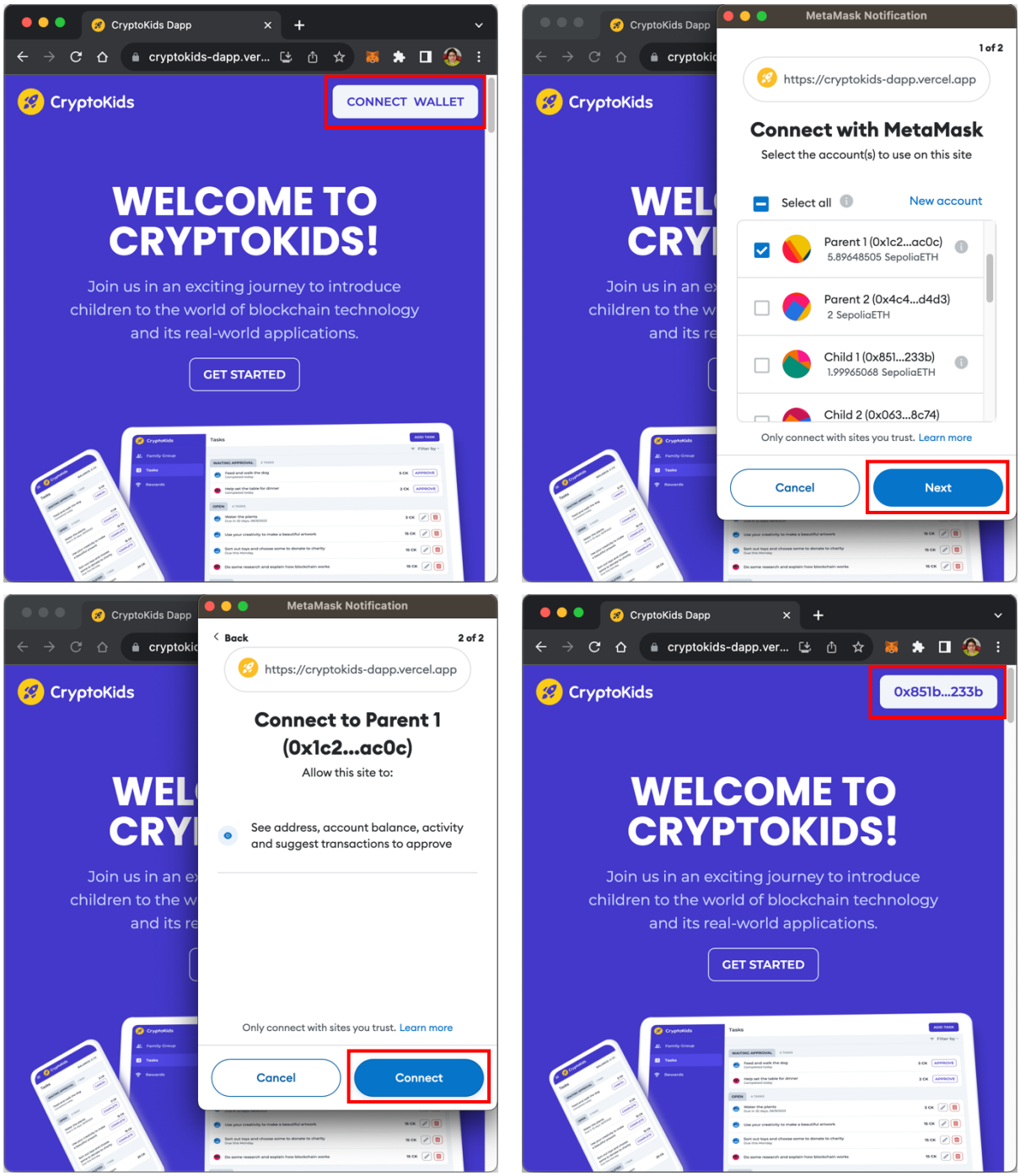



*Figure 67: User testing: connect to the application using MetaMask.*

### 3.6.2.2. Sign up as a parent:

- **Step 1:** user connects to the application using MetaMask.
- **Step 2:** user enters a name and clicks on the “Sing Up” button.
- **Step 3:** user confirms the transaction using MetaMask.
- **Step 4:** the user is registered and has access to the parent dashboard.

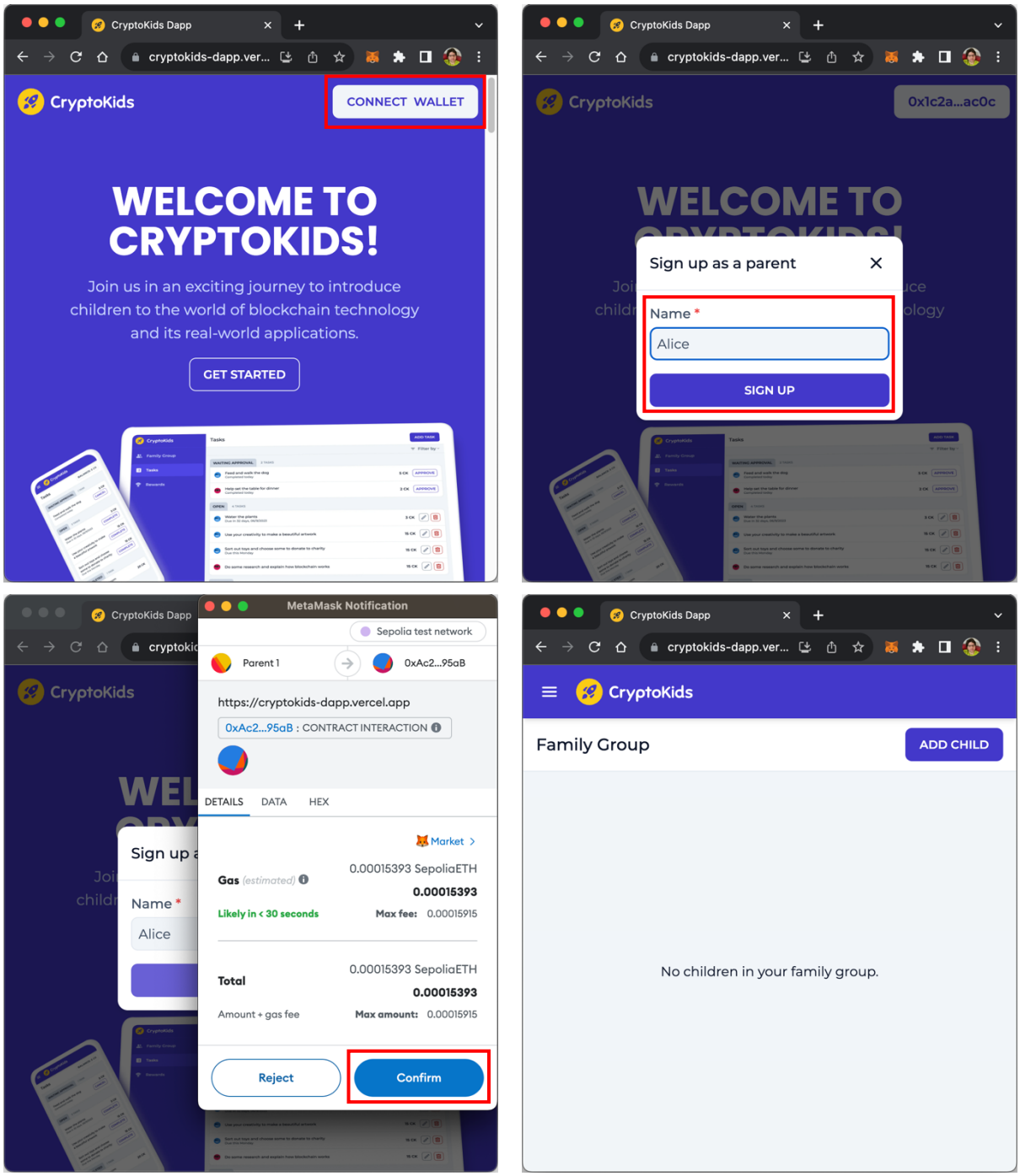



*Figure 68: User testing: sign up as a parent.*

### 3.6.2.3. Add a child to a family group:

- **Step 1:** parent clicks on the “Add Child” button on the “Family Group” page.
- **Step 2:** parent enters the child’s address and name, and clicks on the “Add Child” button.
- **Step 3:** parent confirms the transaction using MetaMask.
- **Step 4:** the child is added to the family group.

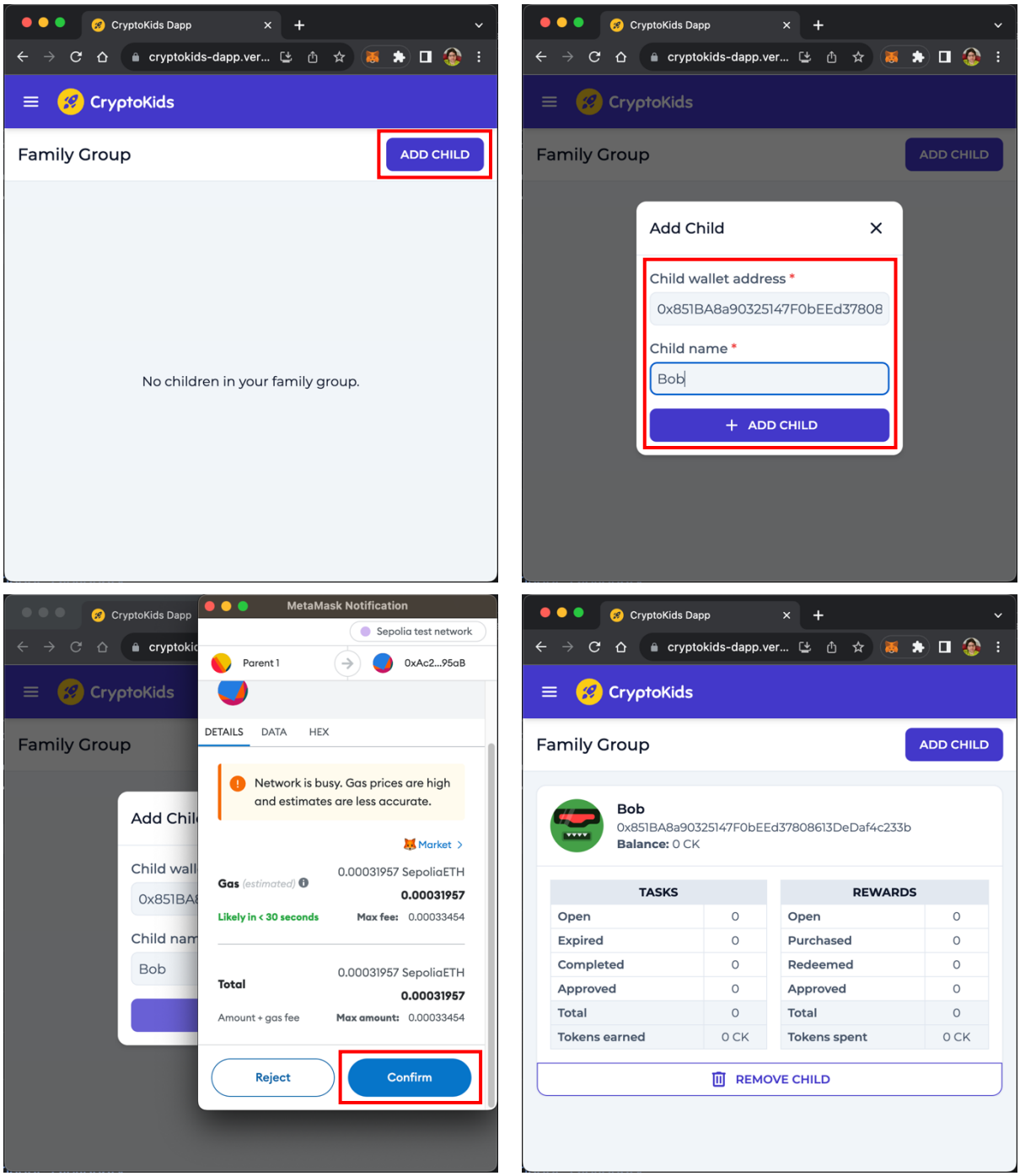



*Figure 69: User testing: add a child to a family group.*

### 3.6.2.4. Add a task:

- **Step 1:** parent clicks on the “Add Task” button on the “Tasks” page.
- **Step 2:** parent selects a child, enters the task’s details, and clicks on the “Add Task” button.
- **Step 3:** parent confirms the transaction using MetaMask.
- **Step 4:** the task is added.

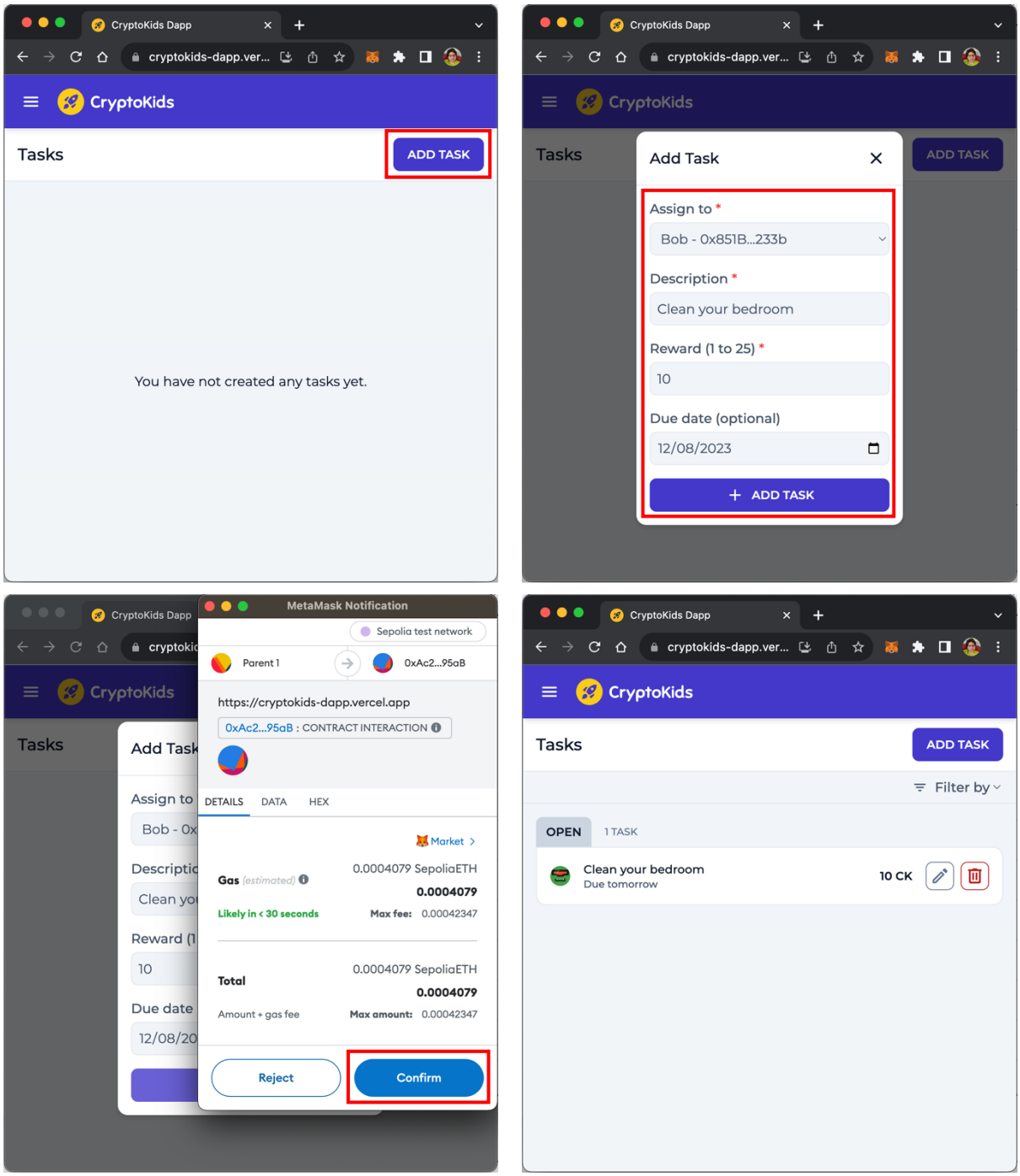



*Figure 70: User testing: add a task.*

### 3.6.2.5. Complete a task:

- **Step 1:** child clicks on the “Complete” button on the “Tasks” page.
- **Step 2:** child confirms the transaction using MetaMask.
- **Step 3:** child waits for the transaction to be confirmed.
- **Step 4:** the task is marked as completed and is now waiting for the parent’s approval.

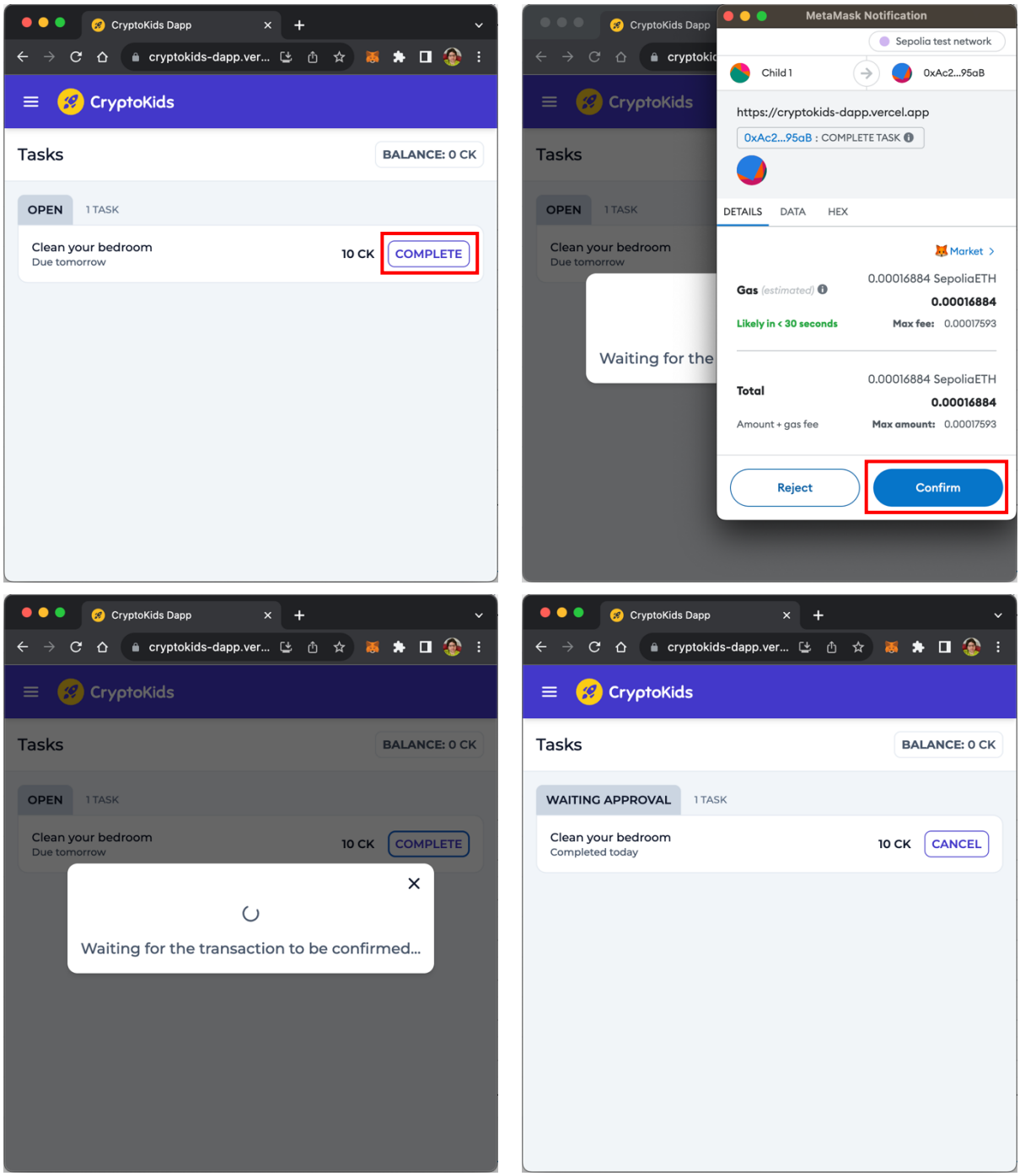



*Figure 71: User testing: complete a task.*

### 3.6.2.6. Approve task completion:

- **Step 1:** parent clicks on the “Approve” button on the “Tasks” page.
- **Step 2:** parent confirms the transaction using MetaMask.
- **Step 3:** task completion is approved.
- **Step 4:** task reward is sent to the child’s wallet.

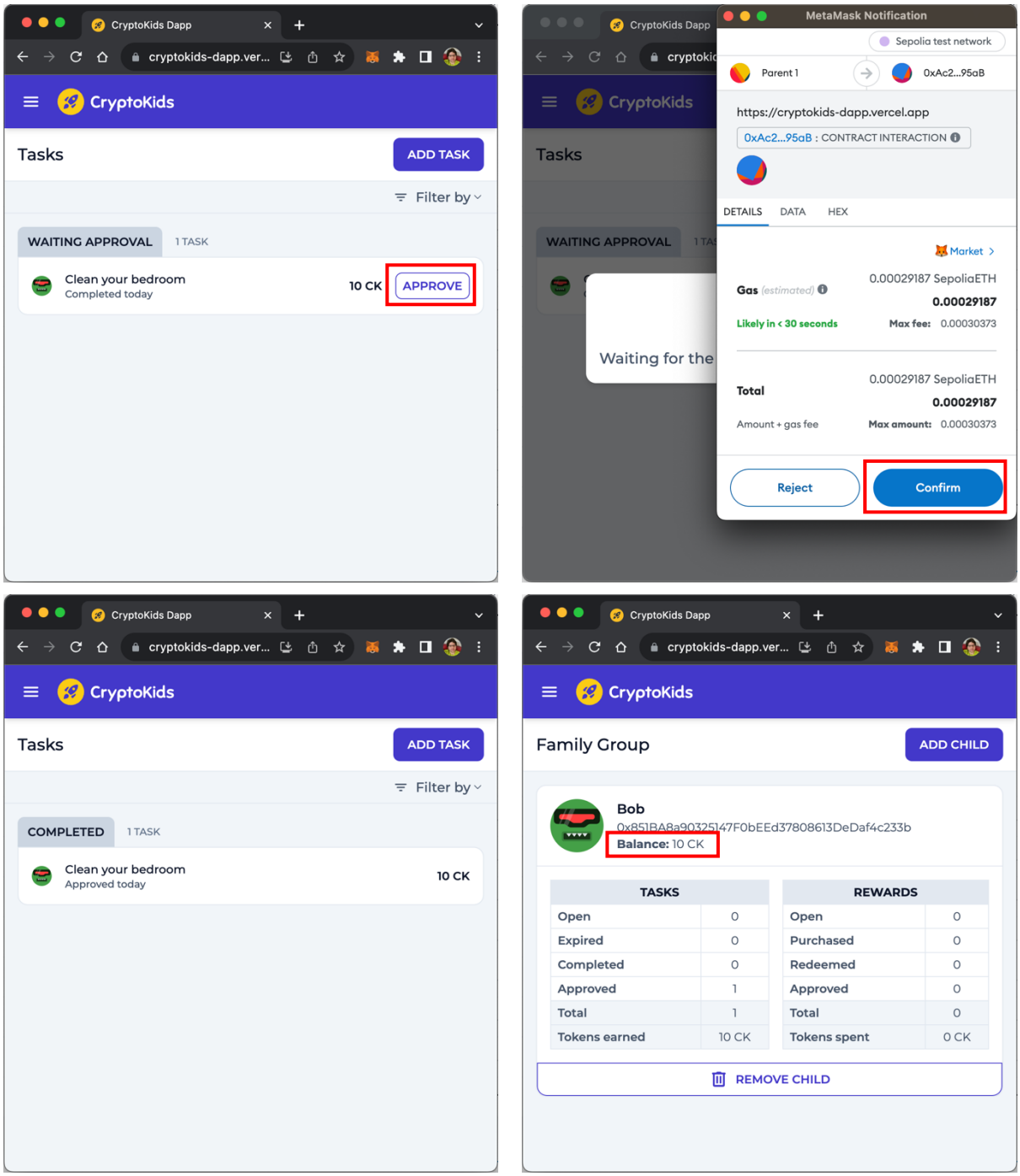



*Figure 72: User testing: approve task completion.*

### 3.6.2.7. Add a reward:

- **Step 1:** parent clicks on the “Add Reward” button on the “Rewards” page.
- **Step 2:** parent selects a child, enters the reward’s details, and clicks on the “Add Reward” button.
- **Step 3:** parent confirms the transaction using MetaMask.
- **Step 4:** the reward is added.

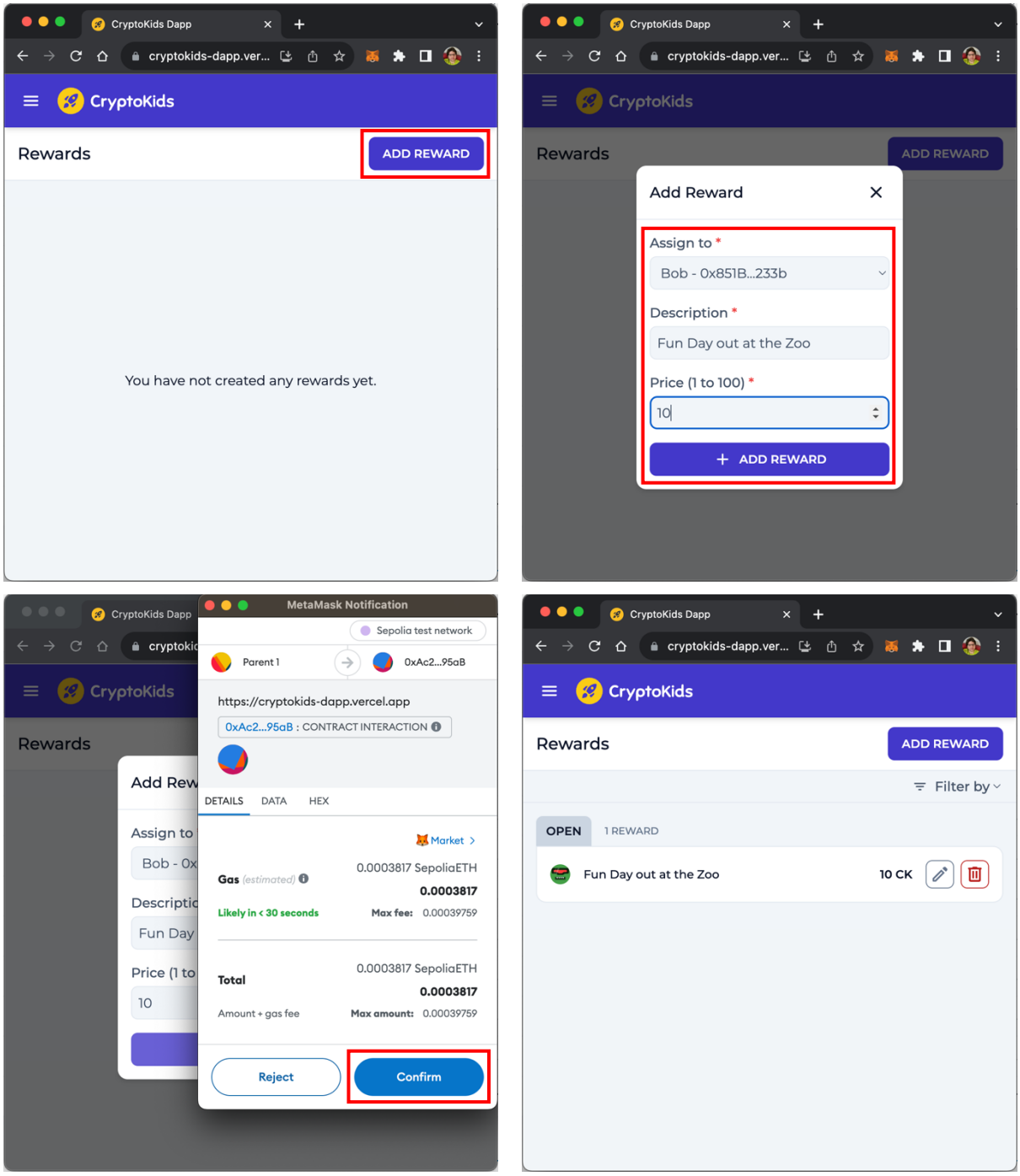



*Figure 73: User testing: add a reward.*

### 3.6.2.8. Purchase a reward:

- **Step 1:** child clicks on the “Buy” button on the “Marketplace” page.
- **Step 2:** child confirms the transaction using MetaMask.
- **Step 3:** child waits for the transaction to be confirmed.
- **Step 4:** the reward is purchased and child’s balance is updated.

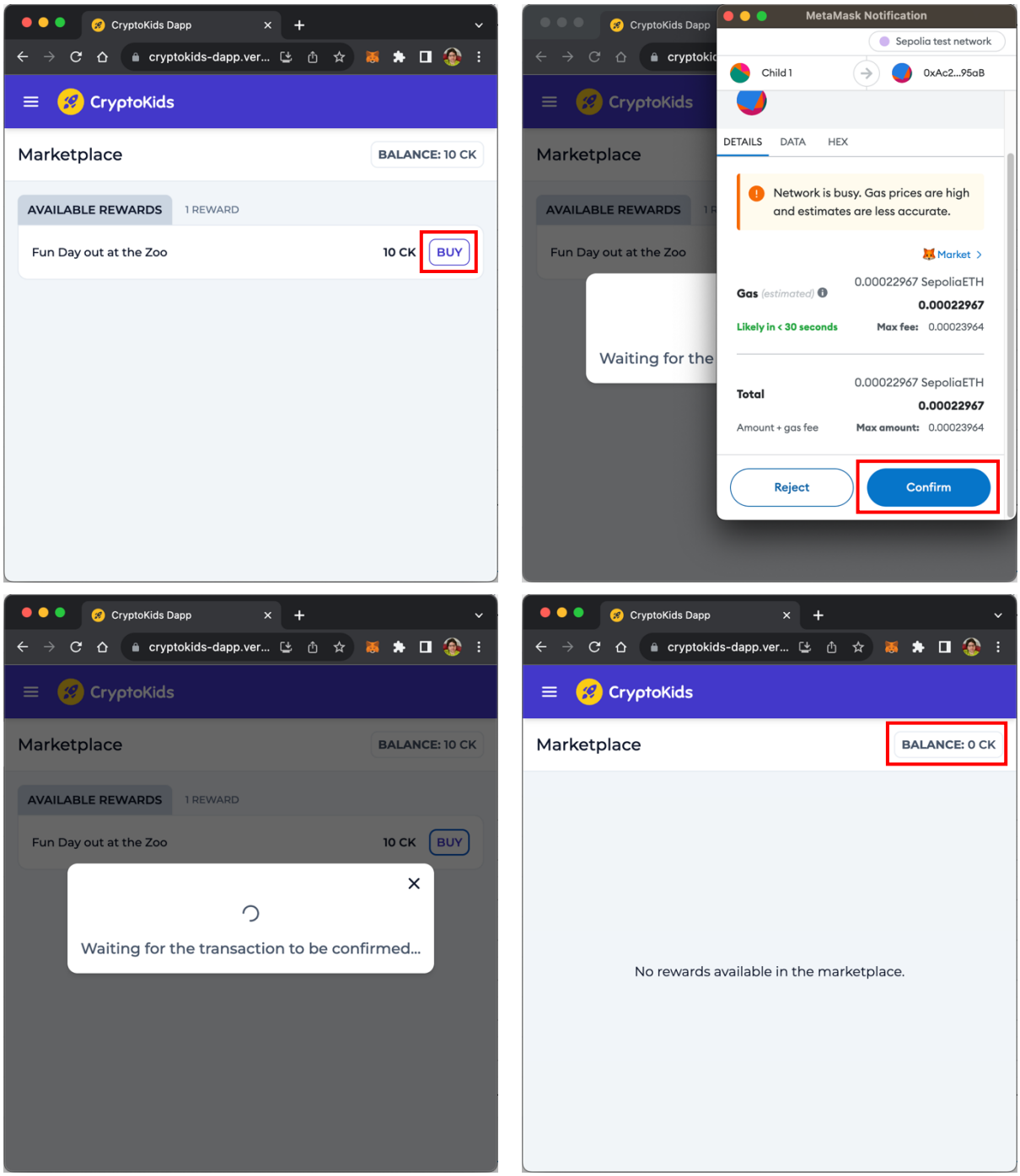



*Figure 74: User testing: purchase a reward.*

### 3.6.2.9. Redeem a reward:

- **Step 1:** child clicks on the “Redeem” button on the “Rewards” page.
- **Step 2:** child confirms the transaction using MetaMask.
- **Step 3:** child waits for the transaction to be confirmed.
- **Step 4:** the reward is marked as redeemed and is now waiting for the parent’s approval.

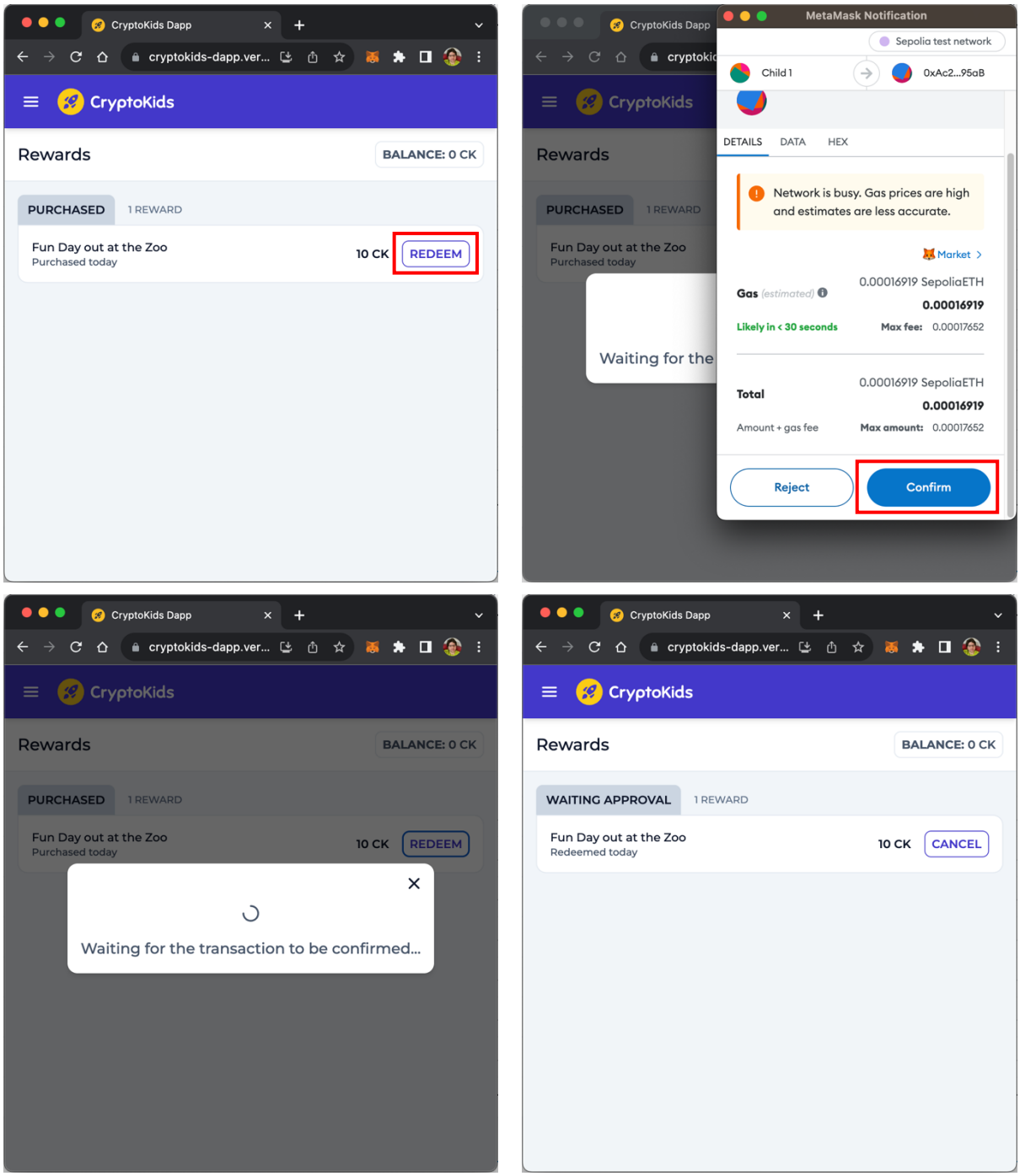



*Figure 75: User testing: redeem a reward.*

### 3.6.2.10. Approve reward redemption:

- **Step 1:** parent clicks on the “Approve” button on the “Rewards” page.
- **Step 2:** parent confirms the transaction using MetaMask.
- **Step 3:** parent waits for the transaction to be confirmed.
- **Step 4:** reward redemption is approved.

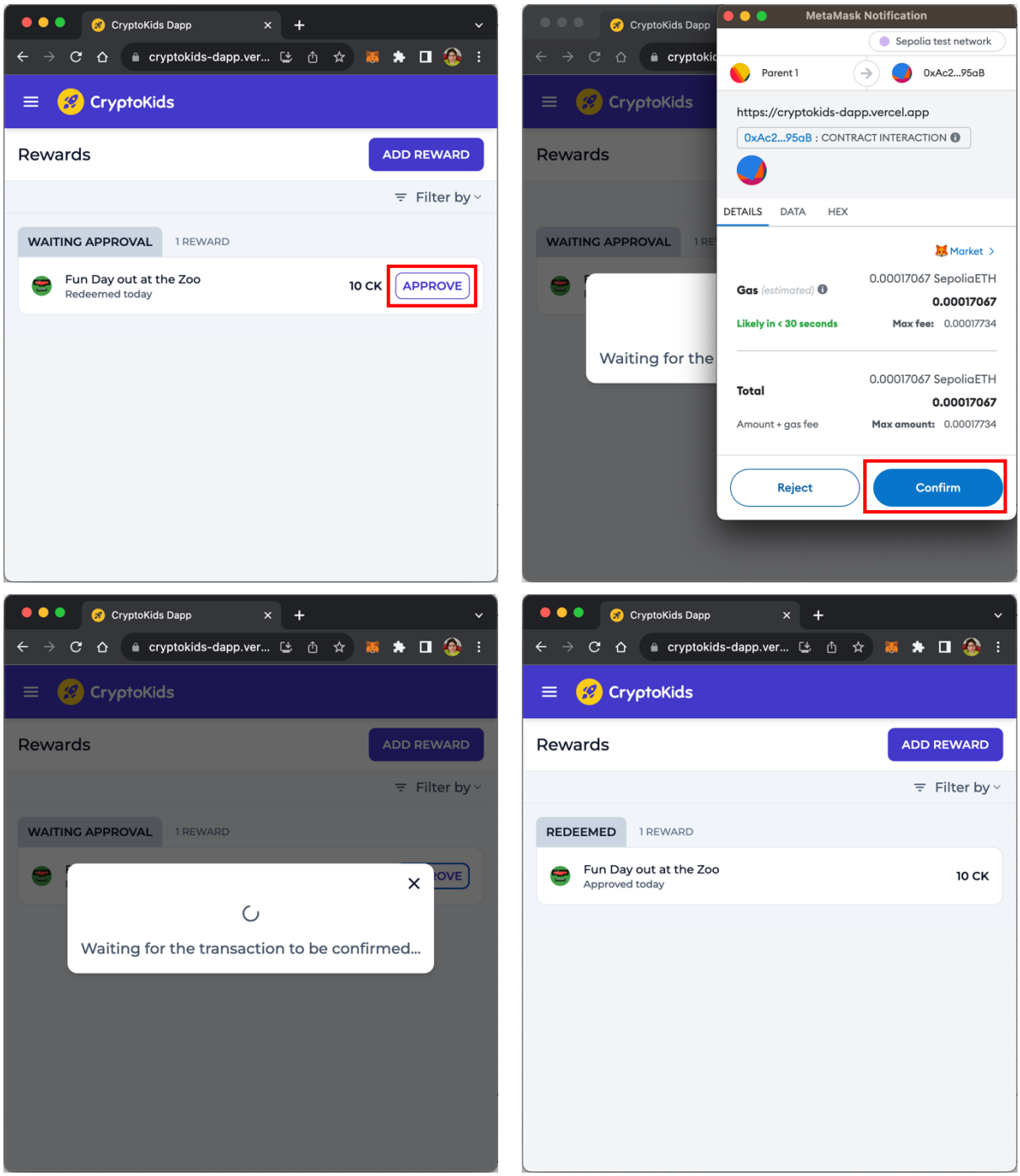



*Figure 76: User testing: approve reward redemption.*

### 3.6.3. Compatibility testing

The CryptoKids Dapp was tested by multiple users across different devices (smartphones, tablets, and computers) and web browsers (Chrome, Firefox, Brave, Edge, Opera, and the MetaMask mobile application browser).

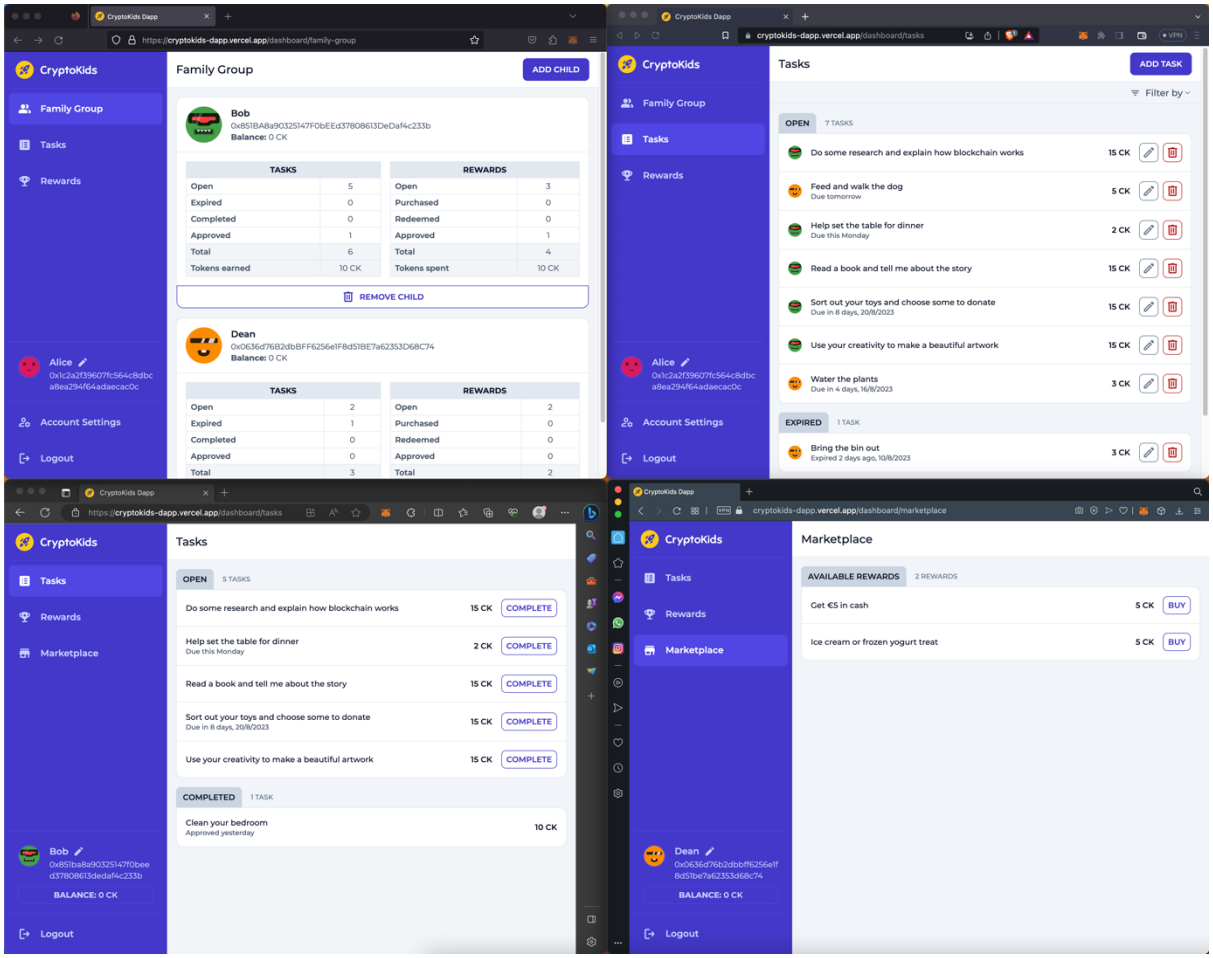



*Figure 77: Compatibility testing: web browsers (Firefox, Brave, Edge, and Opera).*

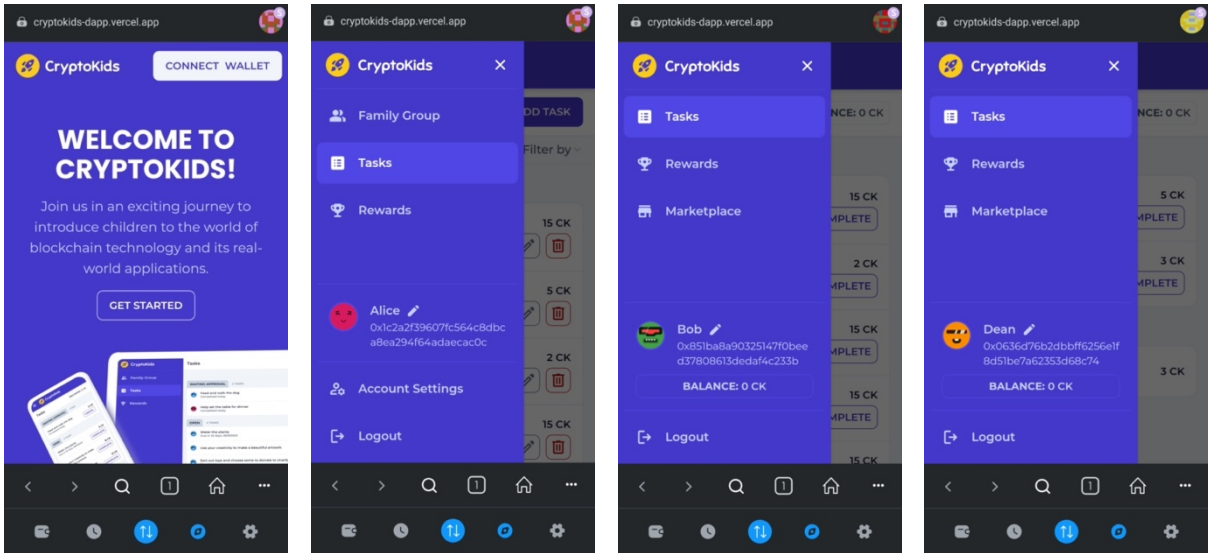



*Figure 78: Compatibility testing: MetaMask mobile application browser.*

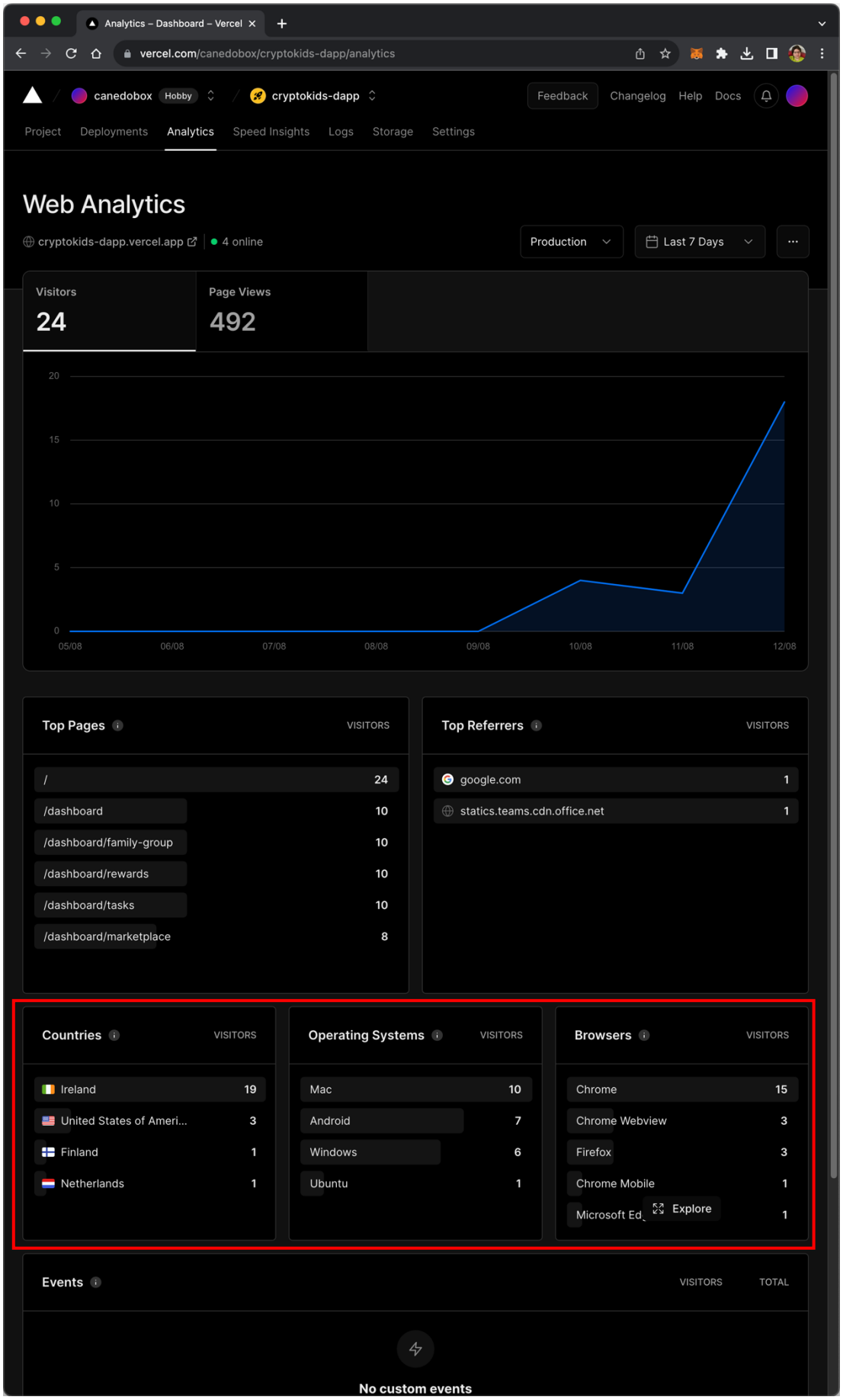



*Figure 79: Compatibility testing: Vercel web analytics dashboard.*

### 3.6.4. Responsive testing

The CryptoKids Dapp is fully responsive and provides a smooth user experience on any device, such as smartphones, tablets, and computers.

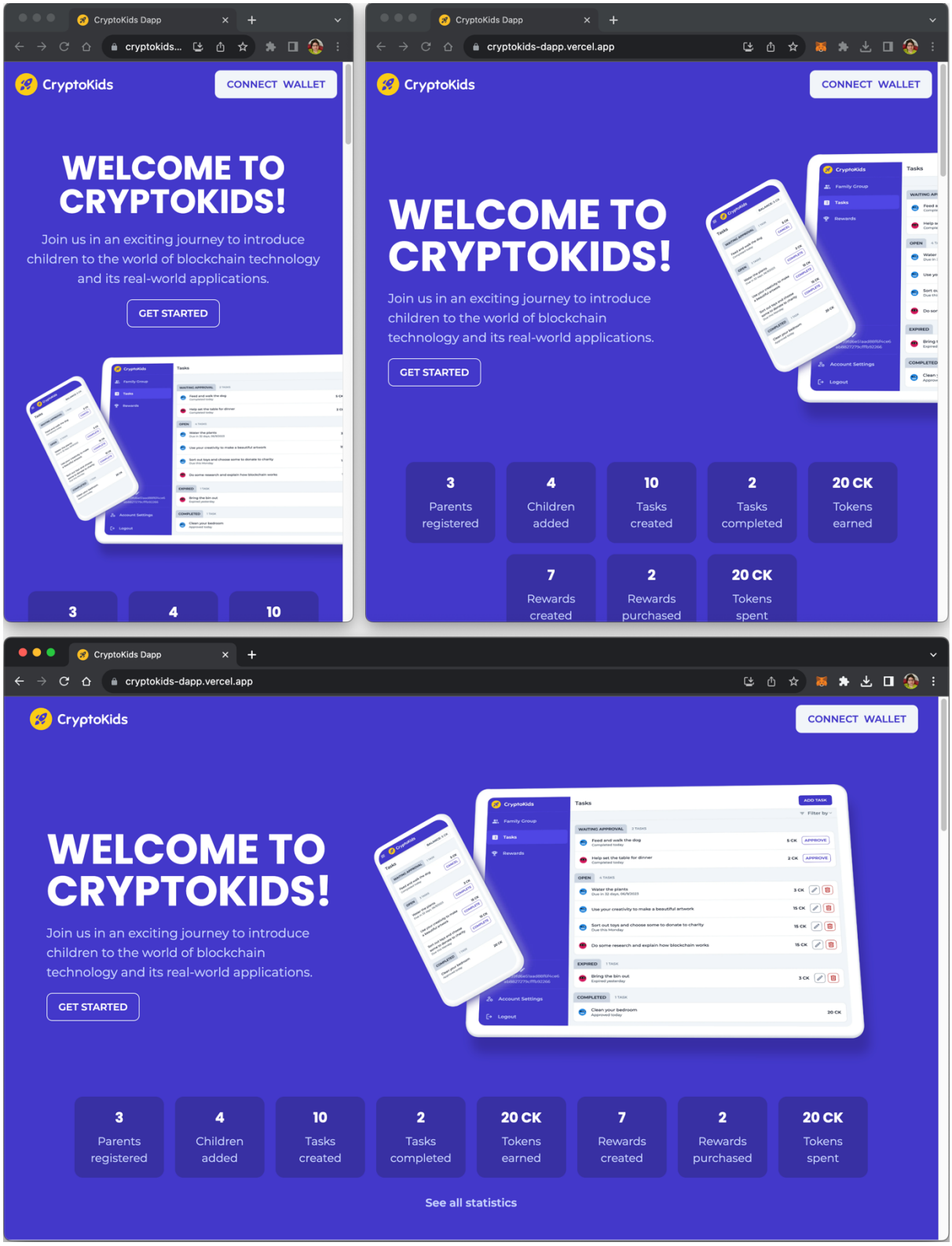



*Figure 80: Responsive testing: homepage.*

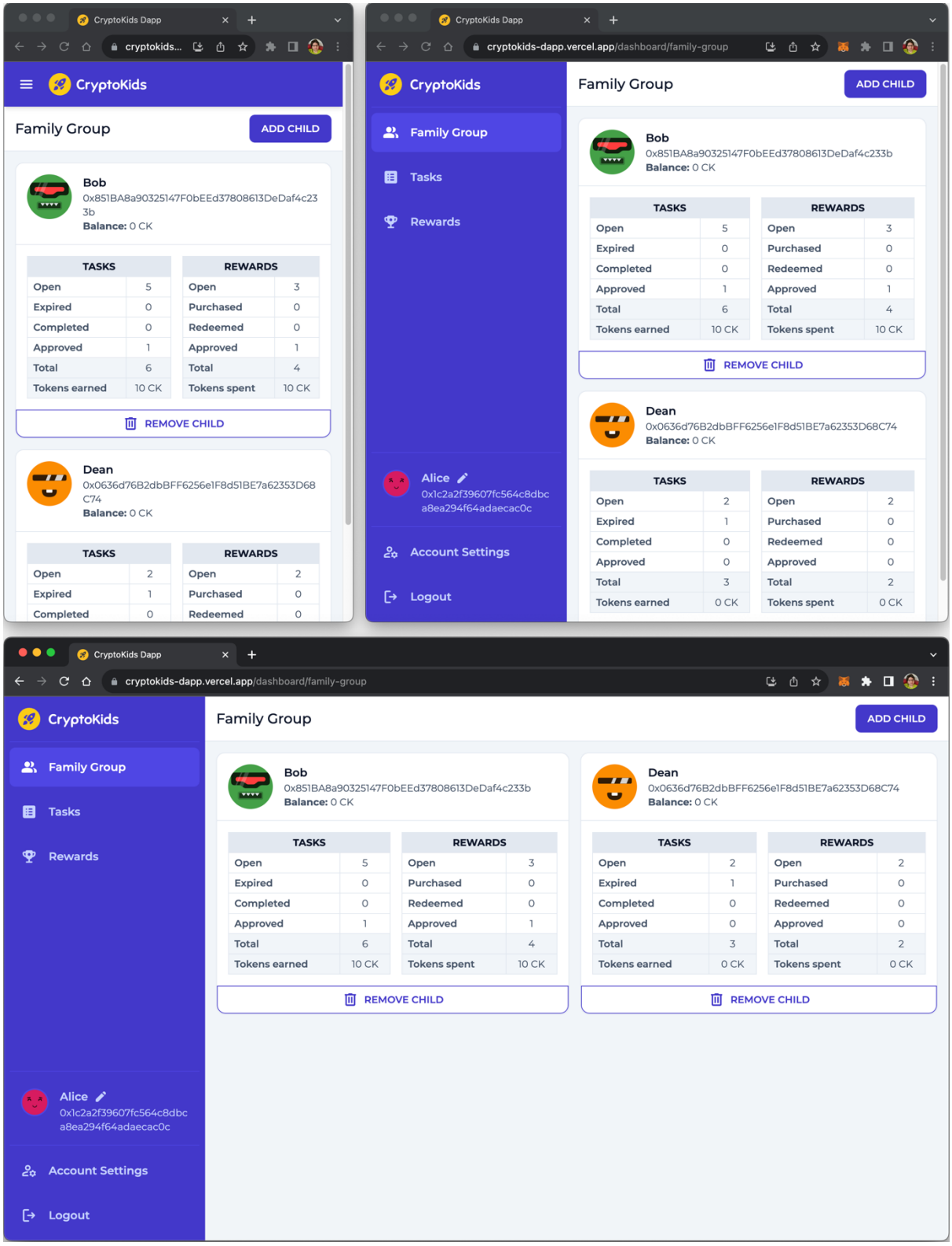



*Figure 81: Responsive testing: parent dashboard.*

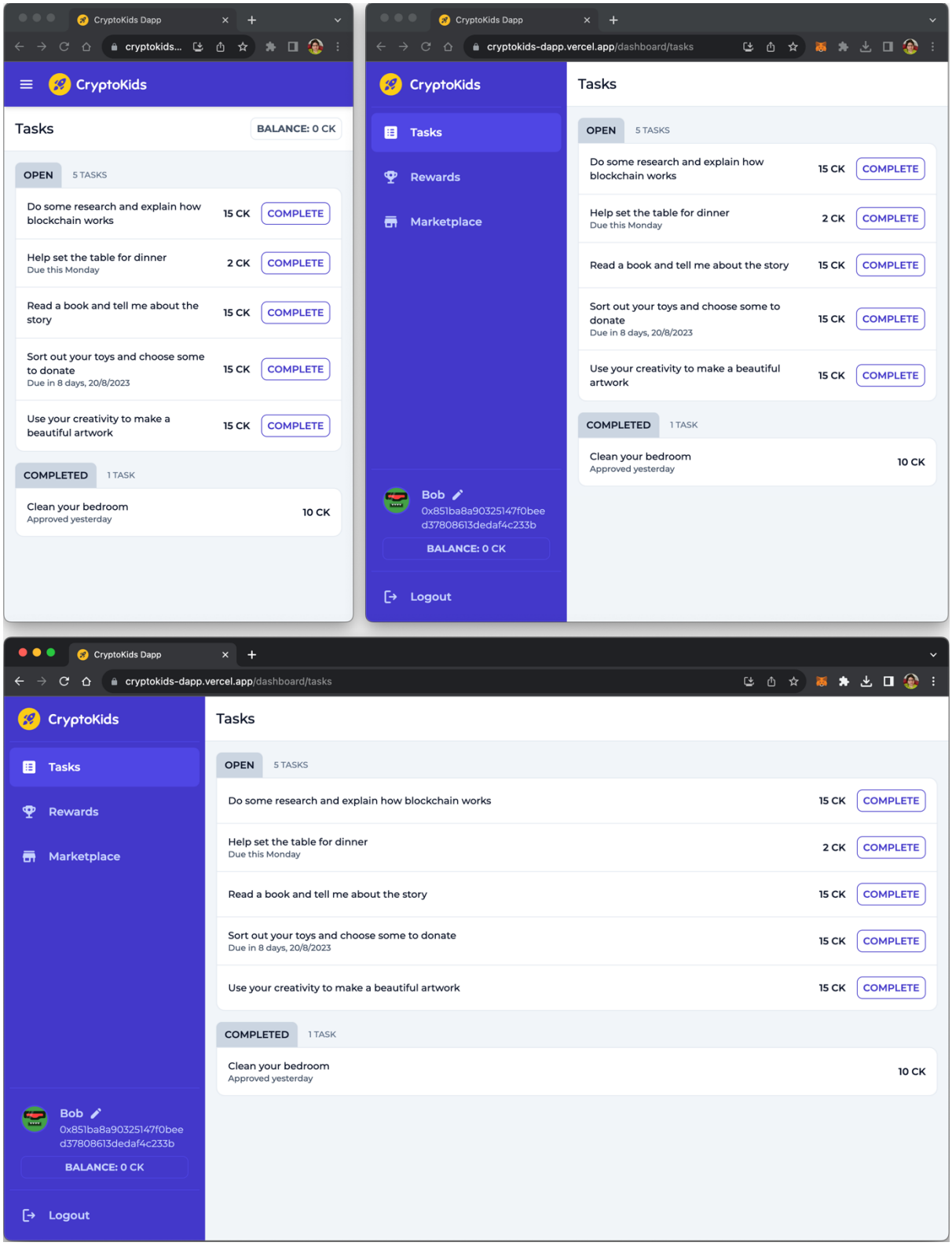



*Figure 82: Responsive testing: child dashboard.*

### 3.6.5. Performance testing

Vercel speed insights dashboard provides a view of the application's performance metrics based on visitors' data. This tool was used to monitor the application performance for each type of device (smartphones, tablets, and computers) and page individually.

The CryptoKids Dapp performed very well during testing and scored 100 in most of the performance tests across all devices and pages (see Figures 83, 84, 85 and 86).

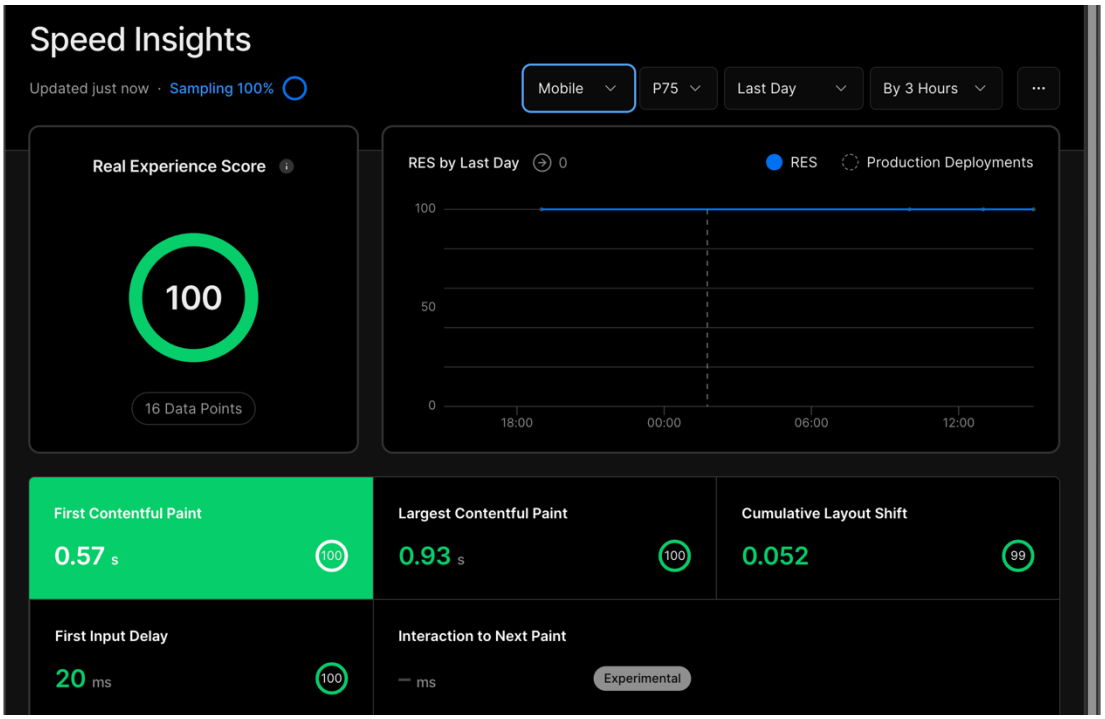



*Figure 83: Performance testing: Vercel speed insights dashboard (mobile).*

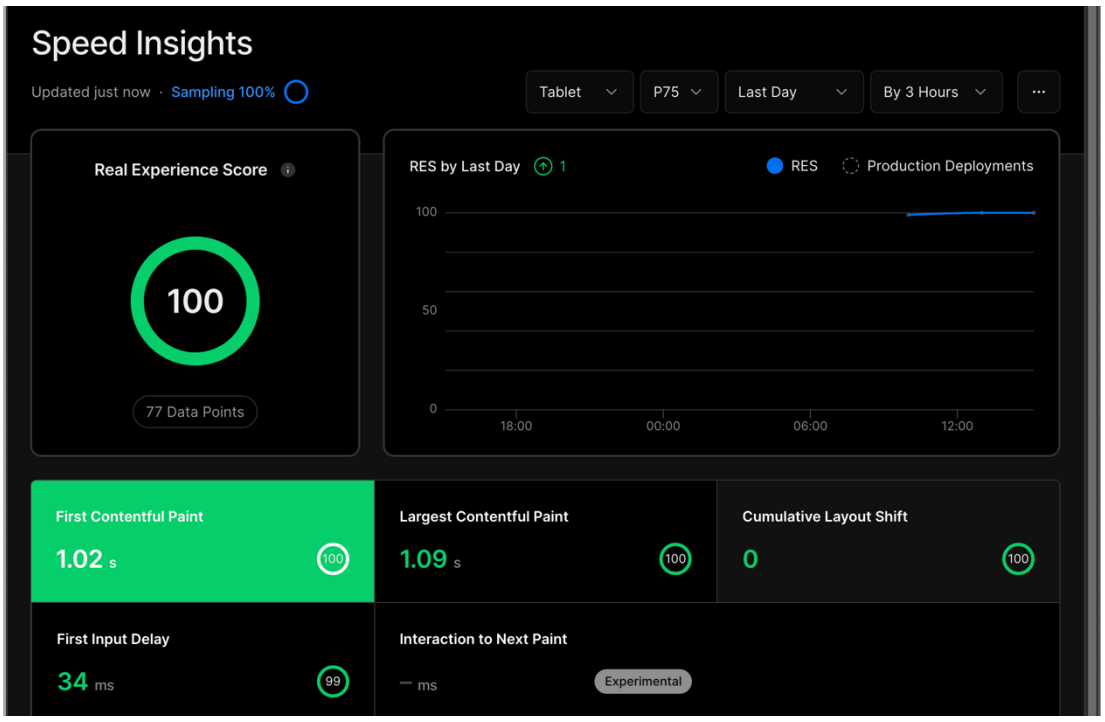



*Figure 84: Performance testing: Vercel speed insights dashboard (tablet).*

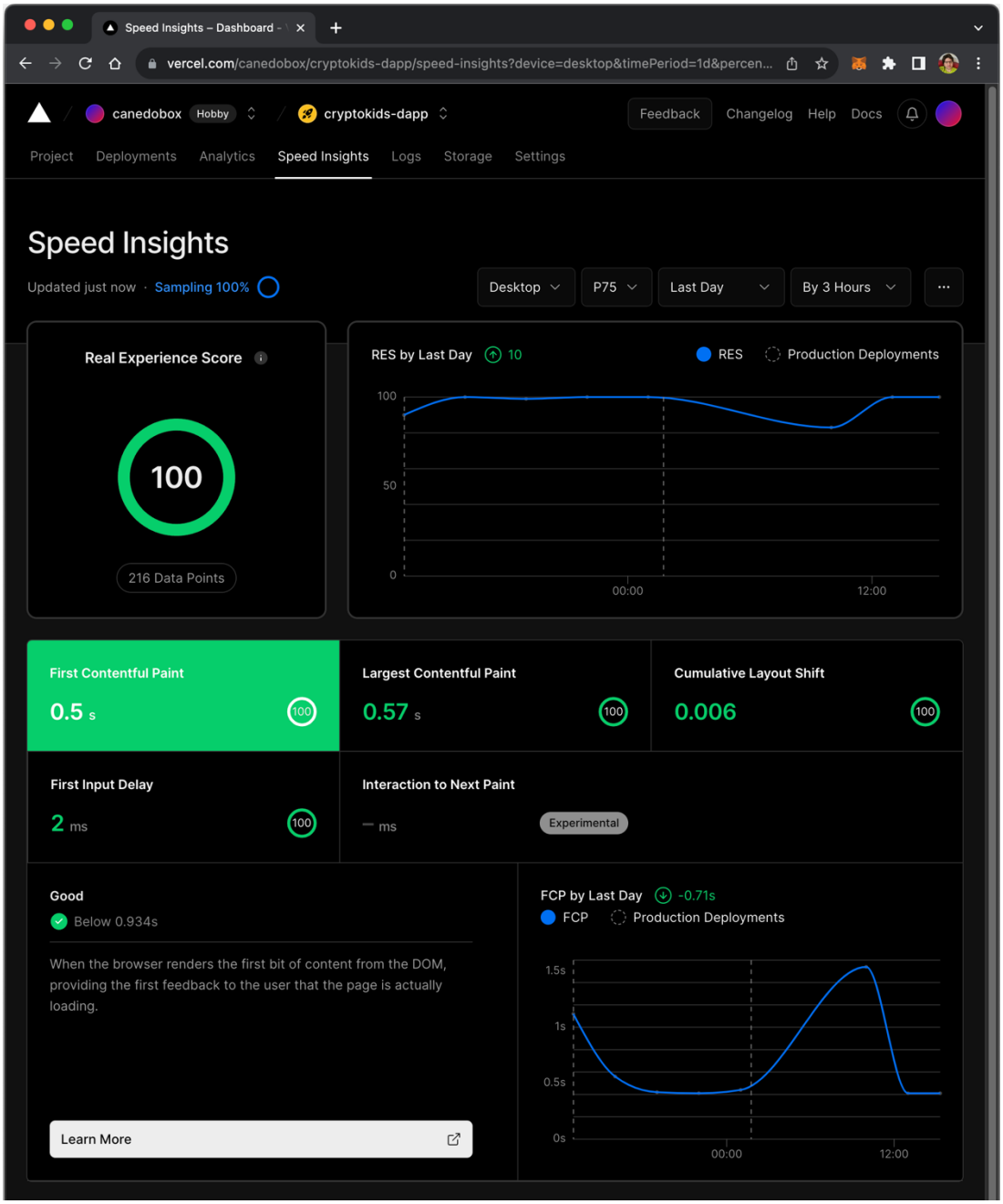



*Figure 85: Performance testing: Vercel speed insights dashboard (desktop).*

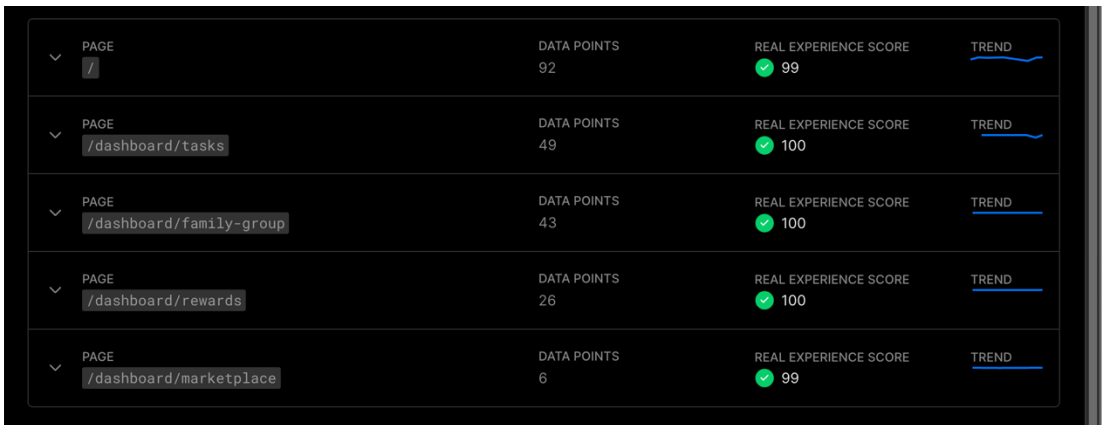



*Figure 86: Performance testing: Vercel speed insights dashboard (application pages).*

# 4. Conclusions

The CryptoKids project's main goals were to introduce children to blockchain technology and its applications with practical activities and in a safe, fun, and engaging manner, while also promoting financial literacy, curiosity for learning, problem-solving skills, and critical thinking by offering a gamified experience, rewards system, safe learning environment, and parental involvement. All of those objectives were successfully met.

The application provides a clean, modern, and fully responsive user-friendly interface suitable for children, with intuitive navigation and smooth user experience on any device (smartphones, tablets, and computers) with an internet browser that supports MetaMask installed (Chrome, Firefox, Brave, Edge, Opera, and the MetaMask mobile application browser).

The thorough testing has also proven that the application performed better than expected, all its requirements were met, all its functionalities were implemented, and that both the smart contract and frontend application are secure, error-free, and that they work as intended.

In conclusion, apart from a small delay and some mirror issues, which were dealt with early and had little impact on the overall development of the project, the CryptoKids project was a success with better outcomes than originally anticipated.

# 5. Further Development

All the CryptoKids Dapp functionalities and requirements were implemented successfully, and the application works as intended. Although that is the case, the application has room for expansion and improvement. Below are a few ideas to improve the application and user experience.

- **Level Up System:**
  A gamification feature where children can earn experience points (XP) by completing tasks, lessons, and quizzes. For each level reached, children would be able to mint a respective NFT, which unlocks extra benefits or rewards. For example, after completing enough tasks, lessons, and quizzes to reach level 5, a child would be able to mint a "Level 5" NFT that gives one extra CK token for each task the child completes. This system encourages children's engagement and participation.

- **Lessons:**
  Educational content designed to introduce children to the concepts of blockchain technology and its applications, such as decentralised applications, smart contracts, cryptocurrencies, and digital transactions. By completing lessons, children would be rewarded with CK tokens and XP. This would provide the children with a clear understanding of blockchain technology in a fun and engaging manner.

- **Quizzes:**
  Interactive quizzes designed to test children's understanding of the content covered in the lessons. Quizzes would be based on the child's learning progress, offering multiple-choice and true/false questions. By completing quizzes and answering the questions correctly, children would be rewarded with CK tokens and XP. This would provide kids with the opportunity to reinforce their knowledge and receive instant feedback on their understanding.

# 6. References


Ethereum. (2023). Intro to Ethereum. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/intro-to-ethereum/ [Accessed 9 August 2023].

Ethereum. (2023). Introduction to dapps. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/dapps/ [Accessed 9 August 2023].

Ethereum. (2023). ERC-20 Token Standard. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/ [Accessed 9 August 2023].

OpenZeppelin. (n.d.). ERC20 - OpenZeppelin Docs. [online] openzeppelin.com. Available at: https://docs.openzeppelin.com/contracts/4.x/erc20 [Accessed 9 August 2023].

Ethereum. (2023). Nodes and clients. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/nodes-and-clients/ [Accessed 10 August 2023].

Szabo, N. (1994). Smart Contracts. [online] www.fon.hum.uva.nl. Available at: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html [Accessed 10 August 2023].

Szabo, N. (1996). Nick Szabo -- Smart Contracts: Building Blocks for Digital Markets. [online] www.fon.hum.uva.nl. Available at: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html [Accessed 10 August 2023].

Ethereum. (2023). Smart contracts. [online] ethereum.org. Available at: https://ethereum.org/en/smart-contracts/ [Accessed 10 August 2023].

Ethereum. (2023). Smart contract security. [online] ethereum.org. Available at: https://ethereum.org/en/developers/docs/smart-contracts/security/ [Accessed 10 August 2023].

# 7. Table of Figures