International Journal Software Engineering and Computer Science (IJSECS)
6 (1), 2026, 288-300
Published Online April 2026 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v6i1.7046.

RESEARCH ARTICLE Open Access

Design and Feasibility Evaluation of Redux Toolkit-Based State Management in a React Reservation Prototype (CoSpace)

Hussein Nurrokhim [1*], **Alz Danny Wowor** [2]

[1*,2] Department of Informatics Engineering, Faculty of Information Technology, Universitas Kristen Satya Wacana, Salatiga City, Central Java Province, Indonesia.

**Corresponding author*: 672022269@student.uksw.edu.

Received: March 27, 2026; Accepted: April 15, 2026; Published: April 20, 2026.

Abstract: This study examines the design and feasibility evaluation of a React-based coworking space reservation prototype named CoSpace, with Redux Toolkit applied as the state management architecture. The study was motivated by manual reservation processes that tend to produce delayed responses, recording errors, and schedule conflicts. A Research and Development (R&D) approach was employed, covering requirements analysis, system design, prototype implementation, functional testing, and user acceptance evaluation. Black Box Testing confirmed that all functional scenarios operated according to the intended design, including schedule conflict validation and role-based access restriction. User Acceptance Testing (UAT) involving 35 respondents produced an overall feasibility score of 95.4%, placing the prototype in the Very Feasible category. Within the scope of this prototype, Redux Toolkit supported centralized organization of application state and business logic — particularly through slice-based state separation and thunk-based reservation approval. The evaluation, however, was confined to a mock-data environment and did not compare Redux Toolkit against alternative state management approaches. Future work should integrate a real backend service and examine system behavior under multi-user operational conditions.

Keywords: Redux Toolkit; React.js; Reservation System; State Management; User Acceptance Testing.

1. Introduction

The growth of freelancers, university students, and creative communities in Salatiga has contributed to rising demand for flexible working spaces. In the post-pandemic context, Flexible Working Space (FWS) has become an important need because it supports productivity, collaboration, and adaptive work patterns (Driyantini *et al.*, 2020). In practice, however, the increasing need for flexible workspace services is not always accompanied by well-organized reservation management. Room reservations are still handled manually in many cases — through phone calls or instant messaging — which may lead to delayed responses, recording errors, and schedule conflicts (Dewi Sintawati, 2019). A preliminary observation of the reservation workflow further indicated that manual communication made it difficult for administrators to verify room availability quickly, especially when several requests arrived close together. In such situations, the administrator had to recheck previous messages before confirming a booking, increasing the risk of delayed confirmation and overlapping schedules. Although this observation was not intended as a statistical measurement, it provided a practical basis for identifying reservation conflict handling and role-based access control as important requirements in the prototype.

© The Author(s) 2026, corrected publication 2026. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third-party material in this article are included in the article's Creative Commons license unless stated otherwise in a credit line to the material. Suppose the material is not included in the article's Creative Commons license, and your intended use is prohibited by statutory regulation or exceeds the permitted use. In that case, you must obtain permission directly from the copyright holder. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

One relevant approach to addressing this issue is the development of a web-based reservation application using a Single-Page Application (SPA) architecture. Bismoputro *et al.* (2017) explain that React.js is well suited for SPA development because it enables responsive interfaces without requiring a full page reload on every user interaction — a characteristic particularly useful for reservation systems, where users expect immediate feedback when filtering rooms, viewing schedules, and submitting reservations. That said, SPA architecture also introduces technical challenges, especially when application data become increasingly numerous, dynamic, and interdependent. In a reservation system, data such as login status, room information, reservation records, and user notifications are not isolated entities; they interact with one another and must be updated consistently across multiple components. If application state is handled only through props or other simple local approaches, the codebase may become increasingly difficult to maintain as the application grows. This issue becomes more critical when the system must support business rules such as schedule conflict validation and role-based access control. Prior studies indicate that centralized state management is generally more appropriate for React applications with more complex data flows and logic structures (Arrafi & Putra, 2025), yet previous reservation-related studies have mostly focused on conventional web-based booking systems or React-based interfaces without explicitly treating centralized state management as a core architectural concern (Dewi Sintawati, 2019; Triandy & Santoso, 2020; Bismoputro *et al.*, 2017).

Earlier works have also highlighted the growing importance of digital reservation services in shared-space environments. Driyantini *et al.* (2020) and Swarnawati *et al.* (2023) show that the development of coworking and flexible working spaces is closely tied not only to the provision of physical space, but also to the availability of practical and efficient supporting services. From the implementation side, earlier reservation systems were still commonly developed using conventional web approaches — Dewi Sintawati (2019) developed a web-based hotel reservation system using a traditional multi-page model, while Hendry Hermawan and Setiyawati (2025) similarly reflected a more conventional web development pattern. Such approaches remain functional, but they are less suited to interaction scenarios that require responsive page transitions and dynamic data updates. More recent studies have adopted modern front-end technologies: Triandy and Santoso (2020) applied React in a reservation-related application, while Bismoputro *et al.* (2017) emphasized the advantages of SPA development with React.js. Neither study, however, placed centralized state management as a primary architectural concern. A more direct treatment of this issue appears in Arrafi and Putra (2025), who compared Redux and Context API in React/Next.js applications and found that centralized approaches offer measurable benefits when application data flows become more complex — though their study did not focus on a reservation system with explicit business rules such as schedule conflict validation and role-based access control.

Based on these prior works, a gap remains: no study has specifically implemented and evaluated Redux Toolkit in an SPA-based reservation system with clearly defined operational logic. Existing studies have shown the importance of digital reservation systems and the advantages of modern web technologies, yet limited attention has been directed toward the practical implementation and evaluation of Redux Toolkit in reservation systems that involve interconnected data, schedule validation, and role-based authorization. This gap matters because reservation systems require not only a responsive interface, but also a reliable mechanism for maintaining data consistency and controlling application behavior across multiple features. The present study addresses that gap through the design and development of CoSpace — a coworking space reservation prototype that applies Redux Toolkit as the state management architecture in a React.js-based application — and evaluates the role of Redux Toolkit in managing application state and supporting complex business logic in a web-based reservation system.

2. Literature Review

2.1 Digital Reservation Systems

Digital reservation systems have been developed across various domains to reduce the limitations of manual booking processes — delayed confirmation, recording errors, and schedule conflicts among them. Dewi Sintawati (2019) developed a web-based hotel room reservation system using the Rational Unified Process method to improve the organization of reservation data and booking transactions, while Triandy and Santoso (2020) developed a web-based tour package reservation application using the MERN stack, showing that reservation services can be supported through integrated web technologies. More recent reservation-system studies also commonly employ Black Box Testing and User Acceptance Testing (UAT) to evaluate whether the developed system works according to functional requirements and is acceptable to users (Sutjiadi *et al.*, 2025). Most reservation-related studies, however, still emphasize functional availability — booking submission, data storage, report generation — without treating the architectural question of how application state is managed across multiple interacting components as a central concern. In the context of a coworking space reservation system, this question becomes important because room availability, reservation status, user roles,

Copyright © 2026 IJSECS

notifications, and approval decisions are interconnected. The present study extends previous reservation-system work by focusing not only on reservation features, but also on the state management architecture that supports the reservation workflow.

2.2 Single-Page Applications and React-Based Interfaces

SPA architecture is widely used in modern web development because it enables dynamic interaction without requiring a full page reload for every user action. In reservation systems, this characteristic is useful because users often expect immediate feedback when selecting dates, filtering available rooms, calculating costs, and submitting booking requests. Bismoputro *et al.* (2017) showed that React.js can be used to develop SPA-based applications with responsive interface behavior. SPA development also introduces challenges in managing data that changes across multiple components, however. React documentation explains that as applications grow, developers need to be more intentional in organizing state and data flow, because redundant or duplicated state can become a source of bugs. This issue is directly relevant to reservation systems: interface responsiveness alone is not sufficient, and the application must also ensure that user interactions, reservation data, and administrative decisions remain synchronized across the system.

2.3 State Management in React Applications

In React applications, state can initially be managed within local components or shared by lifting state to a common parent component. This approach suits simple interactions, but it may become difficult to maintain when many components depend on the same data. React documentation explains that shared state is commonly moved to the closest common parent and passed down through props, while context can reduce prop drilling in deeper component trees. More complex applications, however, often require a more structured approach. For larger applications, centralized state management can organize data flow more predictably. Redux is described as a library for predictable and maintainable global state management, while Redux Toolkit is the recommended approach for writing Redux logic because it simplifies store setup, slice creation, and common Redux patterns. Redux Toolkit also provides createAsyncThunk, which structures asynchronous or process-based logic through promise lifecycle actions — directly relevant to reservation approval, where conflict validation must be completed before a reservation status is updated. Prior research has also addressed the suitability of centralized state management for applications with more complex and interdependent data flows. Arrafi and Putra (2025) compared Redux and Context API in a React/Next.js application and showed that centralized approaches are more appropriate when state interactions become more complex. Brekalo *et al.* (2025) compared several global state management tools in React applications — Context API, Zustand, Redux, and MobX — and emphasized that the suitability of any tool depends on the size, complexity, and requirements of the application. Sharma *et al.* (2025) compared Redux, Zustand, and Context API from the perspective of performance, scalability, and developer experience. Based on this body of work, Redux Toolkit is not treated in the present study as a universally superior solution, but as a suitable architectural option for a reservation prototype that requires centralized control over authentication, reservation data, notifications, and schedule conflict validation.

2.4 Research Gap

Three points emerge from the reviewed literature. First, digital reservation systems have been widely developed to improve booking processes and reduce manual administrative problems. Second, React-based SPA development has been used to support responsive and interactive web interfaces. Third, centralized state management has been discussed as a relevant approach for React applications with complex and interdependent data flows. These areas, however, are often discussed in isolation. Limited attention has been given to the design and feasibility evaluation of Redux Toolkit in a reservation prototype that involves explicit operational rules — schedule conflict validation and role-based access control among them. Existing reservation studies tend to focus on feature implementation, while state management studies often discuss general application architecture rather than reservation-specific business workflows. The present study addresses this gap by designing and evaluating the feasibility of CoSpace, a React-based reservation prototype that applies Redux Toolkit to organize application state and business logic within a controlled mock-data environment.

3. Methodology

A Research and Development (R&D) approach was employed in this study. R&D was selected because the study did not stop at identifying problems in the existing reservation process, but also produced a functional software prototype and evaluated its feasibility and effectiveness — connecting conceptual problem analysis with direct software implementation and testing (Sidik, 2019; Okpatrioka, 2023). The research workflow

Copyright © 2026 IJSECS

adapted a simplified R&D model into five main stages: requirements analysis, system design, implementation, testing, and evaluation, as illustrated in Figure 1. Although the stages are presented sequentially, the development process was iterative in practice, since findings identified during testing could lead to revisions in the design or implementation stages before the final evaluation was conducted.

Figure 1. Stages of the Research and Development (R&D) Method

3.1 Requirements Analysis

The requirements analysis stage was carried out to identify the main problems in the manual reservation process and to define the system requirements that had to be fulfilled by the proposed prototype. The analysis showed that the system needed to support both functional and non-functional requirements. Functional requirements included user login, room data management, reservation creation, schedule conflict validation, notifications, and basic report export. Non-functional requirements included a responsive interface, an understandable interaction flow, and data consistency within an SPA environment. These requirements served as the basis for the subsequent design and implementation stages.

3.2 System Design

At this stage, the results of the requirements analysis were translated into technical system design using the Unified Modeling Language (UML) approach (Umar *et al.*, 2021; Nilawati & Martin, 2023). The design stage was divided into three main parts: functional design, system architecture design, and Redux architecture design. The functional design described how users interact with the system, represented through a use case diagram and several activity diagrams. The use case diagram identified the main actors and their access rights — Tenant and Admin — while the activity diagrams modeled the main business processes, including login, reservation creation, room filtering, reservation approval, room management, and report export. These activity diagrams were later used as the basis for constructing the functional testing scenarios. The system architecture, meanwhile, was designed as a React.js-based SPA following a unidirectional data flow pattern, in which user interactions on the interface trigger specific actions, after which the latest data are processed and returned to the components that require them — an approach selected to support responsiveness at the interface level while maintaining a clearer data flow across the application. For the Redux architecture, the global application state was structured into three main slices: authSlice for authentication and user-role data, roomsSlice for room and reservation data, and tasksSlice for notifications and administrative tasks. This modular division was intended to keep the data structure maintainable and to avoid concentrating all business logic in a single file or component, directly supporting the study's focus on evaluating centralized state management in a reservation system with interconnected data and business rules.

Copyright © 2026 IJSECS

3.3 Implementation

The implementation stage translated the previously prepared design into a working software prototype (Hendry Hermawan & Setiyawati, 2025). The user interface was developed using React.js, while application state was managed using Redux Toolkit. All data operations at this stage were still based on mock data — an intentional limitation so that the study could focus on evaluating the effectiveness of state management architecture on the client side, rather than on network latency, backend services, or external system integration. The implemented prototype included the main features identified during the analysis stage: authentication, room management, reservation handling, schedule conflict validation, notifications, and basic report export, serving as the practical realization of the conceptual design developed earlier.

3.4 Product Testing

After the prototype had been completed, two main forms of testing were conducted: Black Box Testing and UAT (Hendry Hermawan & Setiyawati, 2025; Syafiuddin *et al.*, 2017). The combination of these two approaches was intended to evaluate the system from both technical and user perspectives. The overall testing and evaluation framework is presented in Figure 2.

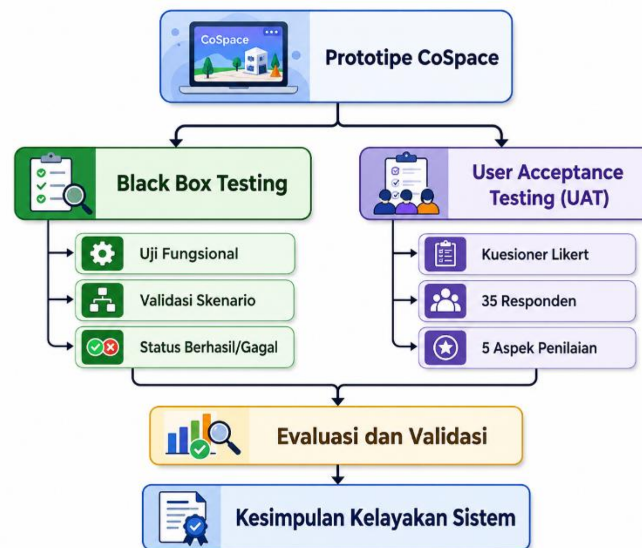

Figure 2. Testing and Evaluation Framework of the CoSpace Prototype

Black Box Testing was conducted to verify whether the main system functions operated according to the intended design, focusing on the relationship between input, process, and output — including specific scenarios such as schedule conflict validation and access restriction to protected pages. Test scenarios were derived from the process models designed in the previous stage. UAT, meanwhile, was conducted to measure the level of user acceptance of the developed prototype using an online questionnaire consisting of 10 statements measured on a five-point Likert scale, covering five evaluation aspects: usability, functionality, interface, performance, and user satisfaction. These aspects were selected to capture not only whether the system worked technically, but also how it was perceived by end users in actual use. Data were collected using purposive sampling involving 35 respondents: 30 students from the Faculty of Information Technology, Universitas Kristen Satya Wacana, representing tenant-type users, and 5 laboratory staff or assistants, representing admin-type users. These two groups were chosen because they reflect the two principal roles within the system — users who make reservations and users who manage and verify them — ensuring that the UAT results reflected both the end-user perspective and the operational perspective of system administrators. Each respondent was asked to use the prototype for approximately 15–20 minutes before completing the questionnaire.

3.5 Evaluation and Validation

The final stage involved evaluating and validating the results of the testing process. Black Box Testing data were analyzed based on the success status of each test scenario, while UAT data were analyzed using the Percentage Index formula to determine the feasibility level of the system based on user responses (Sugiyono, 2018). The results of these two testing stages were then used as the basis for drawing conclusions regarding the feasibility of the CoSpace prototype and the role of Redux Toolkit in supporting state consistency, business logic organization, and the overall reservation workflow. The evaluation in this study is positioned as a feasibility evaluation of the developed prototype rather than a comparative evaluation of Redux Toolkit

Copyright © 2026 IJSECS

against other state management approaches. No benchmarking against Context API, local component state, or other state management libraries was conducted, nor were code complexity, load performance, or multi-user concurrency measured. Findings are therefore interpreted within the scope of functional feasibility, user acceptance, and client-side architectural suitability in a controlled mock-data environment.

4. Result and Discussion

This section presents the results of the design, implementation, and evaluation of the CoSpace system, with particular attention to the relationship between system design and implementation outcomes — and how Redux Toolkit supports the reservation workflow and application data management.

4.1 Results

The design stage produced a set of visual artifacts that served as the basis for system implementation, including a use case diagram, a system architecture design, and representative activity diagrams for the most critical reservation processes. The use case diagram shown in Figure 3 illustrates two main actors: Tenant and Admin. The Admin actor has broader access rights, including the ability to approve reservations, manage room data, and export reports. The arrangement of actors and permissions indicates that role control requirements had already been considered during the design stage — important because the implementation of Role-Based Access Control (RBAC) in the subsequent stage was not introduced independently, but was directly derived from the functional design of the system.

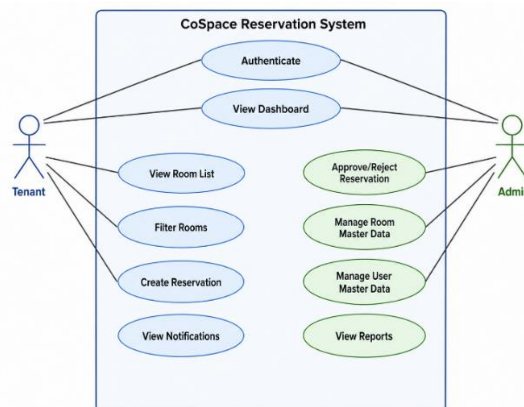

Figure 3. Use Case Diagram of the CoSpace System

The technical system architecture is shown in Figure 4. The system adopts the unidirectional data flow pattern characteristic of React and Redux, in which user interactions on interface components trigger dispatch actions to the store, after which the state is updated and returned to the components that require the latest data. This pattern helps maintain a clear data flow and reduces the possibility of inconsistency across pages — directly relevant to the study's objective of evaluating whether Redux Toolkit can manage interrelated reservation data flows in a more centralized and maintainable manner.

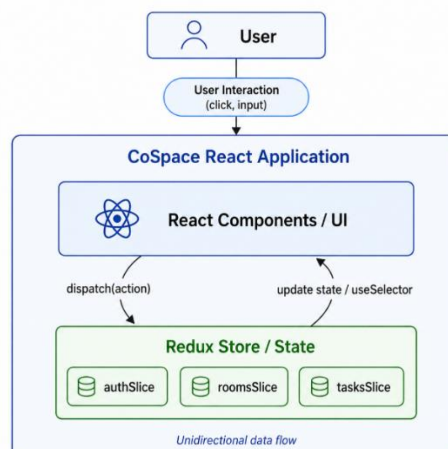

Figure 4. System Architecture of CoSpace (React + Redux)

Copyright © 2026 IJSECS

Two activity diagrams are presented in this paper — reservation creation and reservation approval — because both directly reflect the core user flow and the central business logic evaluated in this study. Figure 5 presents the reservation creation process performed by the tenant, from form completion to the storage of reservation data with an initial Pending status, showing that reservation data are treated from the beginning as part of a structured process that allows later status changes and validations to be traced more clearly. Figure 6 describes the reservation approval process by the admin: before the status is changed to Confirmed, the system first checks for possible schedule conflicts, positioning conflict validation as the main mechanism for maintaining reservation data consistency and preventing approval for overlapping time slots.

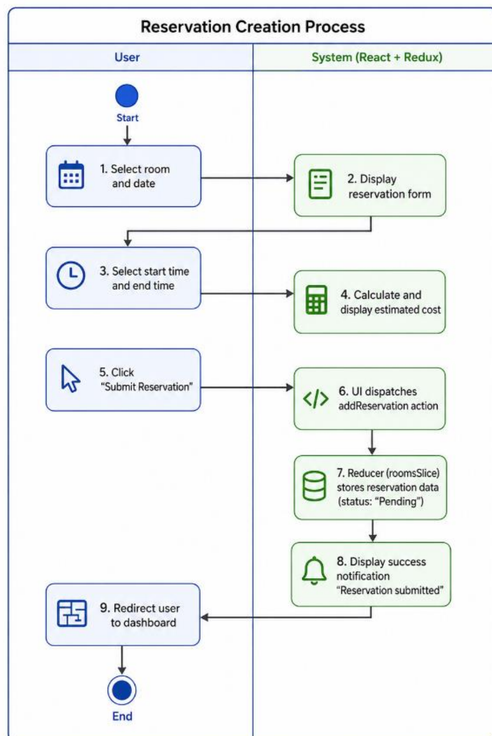

Figure 5. Activity Diagram of the Reservation Creation Process

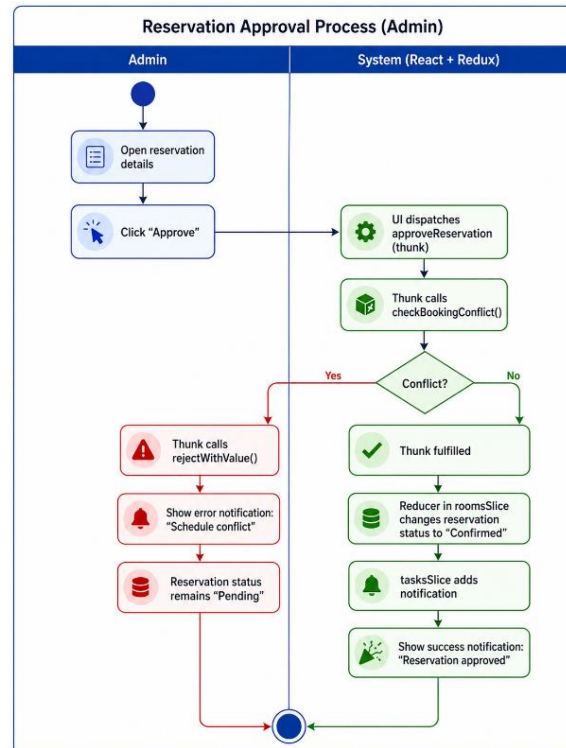

Figure 6. Activity Diagram of the Reservation Approval Process (Admin)

Taken together, the activity diagrams indicate that the system design does not merely focus on interface flows, but also on process sequencing, validation, and access control. The design stage therefore provided a strong foundation for implementation and made it easier to trace the relationship between system requirements, business logic, and testing outcomes. The effectiveness of Redux Toolkit can only be assessed meaningfully if the business processes it supports have been clearly defined from the outset — which is precisely what these design artifacts establish. They serve not only as documentation, but as a basis for examining the consistency between process design, state implementation, and system evaluation results.

4.1.1 System Implementation

The implementation stage translated the entire design into an executable prototype, discussed here from the perspectives of user interface realization and Redux Toolkit architecture. The user interface was developed using React.js and Tailwind CSS. On the reservation form page shown in Figure 7, users can select the date, time, duration, and additional requirements, with changes in the input affecting cost calculation automatically. This indicates that the interface not only accepts input, but also displays processing results dynamically based on the current state — and that the calculation logic follows a structured data flow rather than being implemented as isolated behavior across multiple interface elements. A responsive user experience, in other words, need not come at the expense of business-logic consistency.

Copyright © 2026 IJSECS

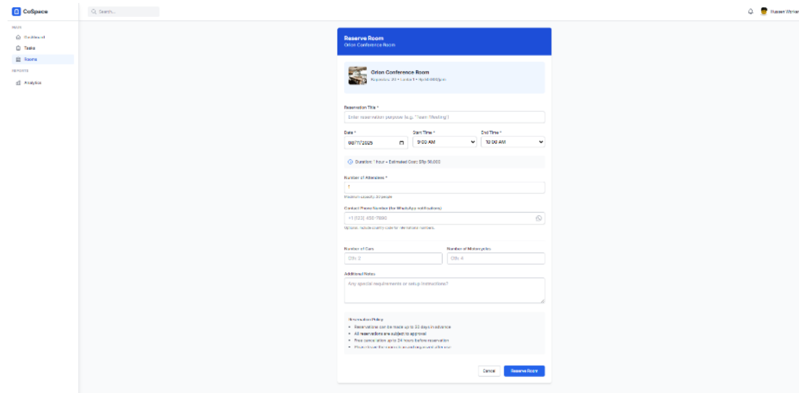

Figure 7. Reservation Form Page Implementation

Figure 8 shows the reservation management page used by the admin to review and approve reservations. The Approve button on this page is directly connected to the conflict-validation logic implemented through Redux Toolkit, indicating that decisions in the admin interface are not made solely at the presentation level but still pass through centralized business-logic validation. From the perspective of architectural evaluation, this page is one of the clearest examples of how centralized business logic helps preserve decision consistency when reservation data change — Redux Toolkit functioning here not only as a data container, but as a controller of decision rules in a critical process.

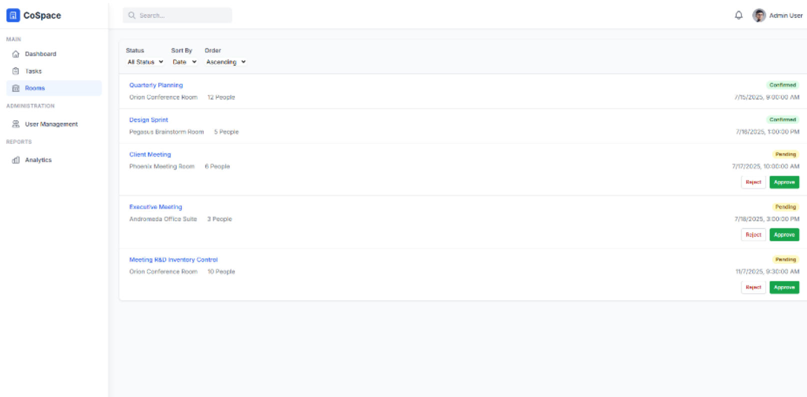

Figure 8. Reservation Management Page Implementation (Admin)

4.1.2 Redux Toolkit Architecture Implementation

This section explains the technical core of the study: how Redux Toolkit was used to manage application state in a centralized manner.

All major application data are stored in a single global store, the root configuration of which is shown in Listing 1.

Listing 1. Root Store Configuration (store.js)

```
import { configureStore } from '@reduxjs/toolkit';
import authReducer from '../features/auth/authSlice';
import roomsReducer from '../features/rooms/roomsSlice';
import tasksReducer from '../features/tasks/tasksSlice';

export const store = configureStore({
  reducer: {
    auth: authReducer,   // stores login status and user roles
    rooms: roomsReducer, // stores room and reservation data
    tasks: tasksReducer, // stores notifications or administrative tasks
  },
});
```

Dividing the store into three slices makes the application logic easier to follow. authSlice is responsible for login status and user roles, roomsSlice manages rooms and reservation data, and tasksSlice handles notifications relevant to administrative actions. This organization prevents application logic from becoming concentrated in a single large file. From the viewpoint of architectural evaluation, this structure also shows that logic which could otherwise be scattered across many components can be centralized within the store and thunk layer — making data updates, process validation, and access control traceable through a shared

Copyright © 2026 IJSECS

flow. To ensure that certain pages can be accessed only by authorized users, the system uses the ProtectedRoute component. The access-right check is placed in a wrapper component so that access control can be handled centrally: the component reads the authentication state and user role from the global store, then redirects unauthenticated users to the login page and unauthorized users away from protected routes, preventing sensitive pages from being briefly displayed to users who lack the required permissions. One of the key business logics in this system is reservation schedule conflict validation. This logic is placed in an async thunk so that the validation process is not scattered across interface components, as shown in Listing 2.
Listing 2. Conflict Validation in Thunk

```
export const approveReservation = createAsyncThunk(
  'rooms/approve',
  async (reservationId, { getState, rejectWithValue }) => {
    const state = getState(); // retrieve the latest state from the store

    // locate the reservation being processed
    const target = state.rooms.reservations.find(
      (r) => r.id === reservationId
    );

    // check whether the schedule conflicts with another reservation
    const isConflict = checkConflict(state.rooms.reservations, target);

    // if a conflict is found, stop the process and return an error
    if (isConflict) {
      return rejectWithValue('Failed: the schedule conflicts with another reservation.');
    }

    // if there is no conflict, return the reservation id for reducer processing
    return reservationId;
  }
);
```

Placing validation logic inside a thunk provides several advantages. Interface components become simpler because they do not need to contain long business rules. Validation is performed against the latest application state through getState(), making decisions more consistent with the current data. The same logic can also be reused in other parts of the system without rewriting identical rules. Within the implemented prototype, Redux Toolkit supported the organization of business logic in a centralized manner — particularly for the reservation approval flow — and in the controlled mock-data setting, this structure supported more traceable state updates and helped maintain consistency in the client-side reservation workflow.

4.1.3 Testing and Evaluation Results

System evaluation was conducted using two approaches: functional testing through Black Box Testing and user acceptance testing through UAT. Black Box Testing was used to ensure that the major system functions operated according to the design, covering both normal workflows and edge cases that could potentially lead to errors. The results are presented in Table 1.

Table 1. Functional Testing Results (Black Box)

No	Test Scenario	Expected Result	Actual Result	Status
1	Tenant logs in with invalid credentials	The system rejects the login and displays an error notification	The system rejected the login and displayed an error notification	Passed
2	Tenant logs in with valid credentials	The system accepts the login and redirects the user to the dashboard	The system successfully logged in and redirected the user to the dashboard	Passed
3	Tenant creates a new reservation	Reservation data are stored with the initial status Pending	The reservation was stored in the state with Pending status	Passed
4	User filters rooms based on capacity	The room list adjusts to the selected filter	The room list was reactively updated according to the filter	Passed
5	Admin approves a reservation with no conflict	The reservation status changes to Confirmed	The status changed to Confirmed and a success notification appeared	Passed
6	Admin approves a reservation with a schedule conflict	The system rejects the approval and displays an error notification	The action was rejected by the thunk, an error notification	Passed

Copyright © 2026 IJSECS

			appeared, and the status remained Pending	
7	Admin exports a reservation report	The report file is successfully downloaded	The report file was successfully downloaded through the browser	Passed
8	User submits a reservation form with empty required fields	The system rejects the submission and shows validation messages in the form	The form was not submitted and validation messages appeared on the empty fields	Passed
9	A tenant forces access to the /admin URL	The system rejects access and redirects the user to the dashboard	ProtectedRoute detected an invalid role and redirected the user to the dashboard	Passed
10	An unauthenticated user forces access to the /admin URL	The system rejects access and redirects the user to the login page	ProtectedRoute detected the unauthenticated status and redirected the user to the login page	Passed

All ten testing scenarios obtained a Passed status, indicating that the main functions of the prototype operated according to the intended design. The successful result in the schedule conflict scenario is particularly telling: the logic implemented through Redux Toolkit — specifically through thunk — was able to prevent reservation status changes when conflicting data were detected. The route protection results, meanwhile, show that access restriction can be consistently enforced at the application level because authentication data and user roles are managed centrally in the store. Within the controlled prototype environment, the Redux Toolkit-based workflow supported traceable data-flow handling in the implemented client-side reservation process. After the system functions had been validated, UAT was conducted to examine how the prototype was received by users. The 35 respondents were selected through purposive sampling: 30 students from the Faculty of Information Technology, Universitas Kristen Satya Wacana, representing tenant-type users, and 5 laboratory staff members, representing admin-type users. This composition ensured that the evaluation reflected both the end-user perspective and the operational perspective of system administrators. Each respondent was asked to use the prototype for approximately 15–20 minutes before completing the questionnaire. The UAT instrument consisted of 10 questionnaire items measured on a five-point Likert scale, covering five evaluation aspects: usability, functionality, interface, performance, and user satisfaction. The findings are summarized in Table 2.

Table 2. Summary of UAT Results by Evaluation Aspect

Aspect	Score Obtained	Maximum Score	Percentage Index	Category
Usability	333	350	95.1%	Very Feasible
Functionality	509	525	96.9%	Very Feasible
Interface (UI)	335	350	95.7%	Very Feasible
Performance	155	175	88.5%	Very Feasible
Satisfaction	338	350	96.5%	Very Feasible
Total	1670	1750	95.4%	Very Feasible

The total score obtained was 1670 out of a maximum possible score of 1750. UAT was analyzed using the Percentage Index (PI), calculated as follows:

$$PI = \frac{\text{Score Obtained}}{\text{Maximum Score}} \times 100\%$$

Where Maximum Score = $Y \times m \times N$, with Yas the highest Likert-scale score (5), $m$as the number of items in each aspect, and Nas the number of respondents (35). All evaluation aspects fall within the Very Feasible category. Functionality achieved the highest score at 96.9%, indicating that respondents perceived the core features as useful and understandable. Satisfaction followed at 96.5%, the interface aspect reached 95.7%, and usability obtained 95.1%. Performance received the lowest score at 88.5% — still within the Very Feasible category, though the gap from the other aspects deserves attention. This result may reflect characteristics of Single-Page Applications, which require JavaScript assets to be loaded during initial access; under certain device or network conditions, this initial loading process may make the system feel slightly slower than subsequent interactions after the application has fully loaded. The prototype also operated entirely on the client side using mock data, so performance optimization for broader deployment scenarios was not the main focus of this study. A visual comparison of the UAT aspect scores is provided in Figure 9.

Copyright © 2026 IJSECS

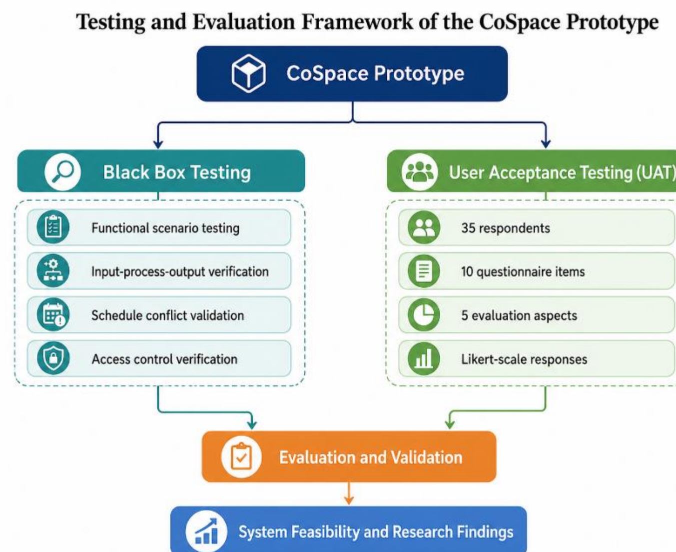

Figure 9. Comparison of UAT Aspect Scores

The system obtained a total Percentage Index of 95.4% (1670/1750), placing the CoSpace prototype in the Very Feasible category. The developed prototype can therefore be considered well accepted by users within the context of the conducted evaluation.

4.2 Discussion

The findings of this study indicate that Redux Toolkit can support the development of a reservation prototype with interrelated data flows and business rules. From the design stage through implementation, the centralized state structure made it possible to organize authentication, room data, reservation records, notifications, and access control within a coherent application flow — reflected in the successful implementation of the root store, route protection, and conflict-validation logic in thunk, all of which contributed to more traceable and consistent behavior across the implemented prototype features. The use of Redux Toolkit also introduced a trade-off worth acknowledging directly. On one hand, it required additional architectural structure: store configuration, slice separation, action dispatching, selector usage, and thunk-based business logic — a structure that may be more complex than local component state for small and isolated interface interactions. On the other hand, the reservation workflow in CoSpace involved interconnected data, role-based access, notifications, and conflict validation. In this context, the additional structure was considered justified because it allowed critical business rules to be placed outside individual UI components and executed consistently within the implemented client-side workflow before state changes were committed. Redux Toolkit was not treated as a universally superior solution, but as a suitable approach for this prototype because the system required centralized control over interdependent reservation data. The design artifacts also played an important role in supporting implementation quality — the use case diagram and activity diagrams served as a structural basis for the implementation of business logic and for the preparation of testing scenarios, confirming that the effectiveness of centralized state management is not determined only by the technology selected, but also by the clarity of the process design that precedes it.

The testing results reinforce this interpretation. Functional testing showed that the system successfully executed all ten scenarios, including normal flows and edge cases. The conflict-validation scenario is particularly significant: it demonstrates that Redux Toolkit was not used solely for storing data, but also for controlling business rules in a critical operational process. The UAT results further strengthen the practical relevance of the prototype — the overall score of 95.4% indicates positive acceptance from both user groups, while the relatively lower score on performance points to an important limitation. The system still uses mock data and has not been integrated with a real backend service, which means the present findings primarily reflect the effectiveness of the client-side state architecture rather than the performance of a production-ready deployment. The results should not be read as evidence of production-level consistency under real reservation traffic, but as evidence of feasibility within the implemented client-side prototype.

Compared with previous studies, the contribution of this work lies in its more explicit focus on Redux Toolkit as a state-management architecture in a reservation context. Prior studies showed the usefulness of React.js for SPA development and reservation-related interfaces (Triandy & Santoso, 2020; Bismoputro *et al.*, 2017), while others emphasized the importance of centralized state management for complex React-based applications (Arrafi & Putra, 2025). The present study connects these strands in a single implementation setting — a coworking-space reservation prototype in which state consistency, role-based access, and schedule

Copyright © 2026 IJSECS

validation are treated as core architectural concerns. The prototype relies on mock data and does not yet involve real-time backend integration, however, meaning the present evaluation does not fully capture data synchronization issues, multi-user concurrency, or production-scale performance. Future work may focus on integrating a real backend service, improving performance optimization, and examining RTK Query as an enhancement for more efficient data synchronization and retrieval.

5. Conclusion

This study designed and evaluated the feasibility of CoSpace, a React-based coworking space reservation prototype that applies Redux Toolkit for state management. The prototype fulfilled the main functional requirements — authentication, reservation handling, schedule conflict validation, room data management, notifications, and basic report export. Black Box Testing confirmed that all tested scenarios operated according to the intended design, while UAT involving 35 respondents produced a feasibility score of 95.4%, placing the prototype in the Very Feasible category. Within the scope of the developed prototype, Redux Toolkit supported centralized organization of application state and business logic, particularly through slice-based state separation and thunk-based reservation approval. The study did not, however, compare Redux Toolkit with alternative state management approaches, nor did it test the system under real backend, load, or multi-user concurrency conditions. Future work should integrate a production backend, evaluate synchronization under real reservation traffic, and consider RTK Query for server-state fetching, caching, and synchronization.

References

Arrafi, W. S., & Putra, R. E. (2025). Implementasi dan manajemen state pada website Next.js: Perbandingan Context API dan Redux pada website Mangaice. *Journal of Informatics and Computer Science (JINACS)*, *6*(04), 1131–1144. https://doi.org/10.26740/jinacs.v6n04.p1131-1144

Bismoputro, I., Al Huda, F., & Brata, A. H. (2024). Pengembangan single page application berbasis ReactJS untuk usaha percetakan online (Studi kasus: Global Grafika). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *8*(7).

Brekalo, S., Pap, K., & Kos, F. (2025). Characteristics and comparison of global state management tools in React applications. *Acta Graphica: Znanstveni Časopis za Tiskarstvo i Grafičke Komunikacije*, *33*(1), 23–33. https://hrcak.srce.hr/329694

Dewi, I., & Suminten, S. (2019). Perancangan sistem informasi reservasi kamar hotel berbasis web dengan metode RUP (Rational Unified Process). *Journal of Information System, Informatics and Computing (JISICOM)*, *3*(2), 16–22.

Driyantini, E., Pramukaningtiyas, H. R. P., & Agustiani, Y. K. (2020). Flexible working space, budaya kerja baru untuk tingkatkan produktivitas dan kinerja organisasi. *Jurnal Ilmu Administrasi: Media Pengembangan Ilmu dan Praktek Administrasi*, *17*(2), 206–220. https://doi.org/10.31113/jia.v17i2.584

Hermawan, R. H., & Setiyawati, N. (2025). Pengembangan sistem informasi pencatatan hasil susu pada peternakan sapi perah berbasis web. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *9*(5), 7375–7383. https://doi.org/10.36040/jati.v9i5.14703

Nilawati, L., & Martin, M. (2023). Penerapan metode RAD pada perancangan sistem informasi permohonan data aduan Smartmaps berbasis web. *JURIKOM (Jurnal Riset Komputer)*, *10*(2), 648. https://doi.org/10.30865/jurikom.v10i2.6041

Okpatrioka, O. (2023). Research and development (R&D) penelitian yang inovatif dalam pendidikan. *Dharma Acariya Nusantara: Jurnal Pendidikan, Bahasa dan Budaya*, *1*(1), 86–100. https://doi.org/10.47861/jdan.v1i1.154

Sharma, N., & Charan, S. (2025). Performance and developer experience comparison of Redux, Zustand, and Context API in React applications. *IJSAT: International Journal on Science and Technology*, *16*(2).

Copyright © 2026 IJSECS

Sidik, M. (2019). Perancangan dan pengembangan e-commerce dengan metode research and development. *Jurnal Teknik Informatika UNIKA Santo Thomas*, *4*(1), 99–107. https://doi.org/10.17605/jti.v4i1.516

Sugiyono. (2017). *Metode penelitian bisnis: Pendekatan kuantitatif, kualitatif, kombinasi, dan R&D*. Alfabeta.

Sutjiadi, R., Rahmawati, T., Kristianto, A., & Kanessa, F. T. (2025). Designing a web-based restaurant reservation information system with requirement prototyping method. *Jurnal Techno Nusa Mandiri*, *22*(1), 9–17. https://doi.org/10.33480/techno.v22i1.6110

Syafiuddin, M., Rachmadi, A., & Herlambang, A. D. (2024). Pengembangan sistem informasi akademik berbasis web pada Sekolah Dasar Nahdlatul Ulama' Kepanjen. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *8*(5).

Triandy, R., & Santoso, N. (2020). Pengembangan aplikasi web reservasi paket wisata menggunakan MERN stack (Studi kasus: Zona Tamasya Tour Organizer). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *4*(6), 1616–1624.

Umar, R., Sarjimin, S., Nugroho, A. S., Dito, A., & Gunawan, I. (2021). Perancangan sistem informasi keuangan berbasis web multi user dengan UML. *Jurnal Algoritma*, *17*(2), 204–211. https://doi.org/10.33364/algoritma/v.17-2.204.

Copyright © 2026 IJSECS