IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: Khashanie Barua
Student Reference Number: 10899167

Module Code: PUSL3122	Module Name: HCI, Computer Graphics, and Visualisation
Coursework Title: RoomCrafter: A Web-Based Design Visualization Tool	
Deadline Date: 9th May 2025	Member of staff responsible for coursework: Dr Taimur Bakhshi
Programme: BSc (Hons) Software Engineering	

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

Khashanie Barua	10899167
Osadi Kiriella	10899590
Wijemuni Silva	10899700
Athurugirige Amasha	10899282
Kavithma Samarawickrama	10899192
Godalla Waduge	10820762

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: 

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed :

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.

I *have used/not used translation software.

If used, please state name of software……

Overall mark _____% Assessors Initials _____ Date_____

Table of Contents

Table of Figures

Roles and Responsibilities

Name	Role	Responsibilities
Khashanie Barua	Documentation and Presentation Lead	Report making, Script Making, Data Gathering
Osadi Kiriella	Designer and Developer	Development, Wireframing, UI/UX Desiging
Wijemuni Silva	Documentation and Presentation Lead	Report making, Script Making, Prototyping
Athurugirige Amasha	Developer and Project Lead	Development, Wireframing, UI/UX Desiging
Kavithma Samarawickrama	Usability and Evaluation Lead	Usability Testing, Development
Godalla Waduge	Usability and Evaluation Lead	Usability Testing, Prototyping

Project Links

GitHub repository link:

https://github.com/OWKiriella/Room-Crafter.git

Project demonstration video:

https://youtu.be/DfN9FkOeAkE

1. Introduction

RoomCrafter is a web application that helps users design furniture and prepare their interiors in both 2D and 3D contexts. Users can produce floor plans, arrange furniture, and sample styles and measurements before making changes, or a purchase. The features including real time 3D visualization, and customizable size and color assistance, also allows users to make high-quality decisions about their interior spaces. It is ideal for anyone who is trying to make interior design easier and can help them get closer to experiencing their vision with conviction.

1.1 Application Features

RoomCrafter has the following features:

- User Registration and Login: Users can register as a new user, and login into the application securely as an existing user.
- Dashboard: The Dashboard displays the user's favorite picks, furniture templates, recent designs, and directs the user to other areas in the application.
- Create New Design: When the user is ready to create their design, they will have the ability to start from scratch, set dimensions, decide the room type to design, add furniture, and choose color schemes.
- Template: The user can choose templates based on the room type (i.e. cozy bedroom, modern office).
- 2D / 3D: The user can switch between 2D and 3D views which will show how the furniture is shown in their room.
- Room Lock: The user can lock the room position to have manipulation only of furniture, giving the user greater control while displaying items in more precise positioning.
- Interactive Controls: The user will have a lot of control over items in the room particularly with respect to zooming in and out, rotating, moving, changing color and other functions of the room items under their control.

1.2 Functional and Non-Functional Requirements

Functional Requirements

- User Authentication - Users should be able to sign up, login and manage their account.
- User's Choice - Users should be able to enter the dimensions, colors, shapes and room type for furniture that they would like to customize.
- Design Interface - User should be able to create, view and edit designs using 2D or 3D interactive tools.
- Furniture Interfaces - Users can rotate, move, scale, zoom, and apply color changes to furniture in their virtual room.
- Room Locking - Allows the user to lock the view of the 3D room, so furniture can still be moved.
- Template and catalogue browsing - Users can browse templates or browse furniture catalogue that exists in their model.

Non-Functional Requirements

- Responsiveness - The website must behave appropriately across different screen sizes and browsers (e.g. desktop, tablet and mobile).
- Performance - The rendering of the 3D or 2D user interfaces and any transitions will be smooth and must load within three seconds.
- Usability - The tools will clearly display the quality, usage and user experience while remaining easy to use existing in the interfaces.
- Security - handle user information securely, especially during login and payment.

1.3 Prototype

Low-Fidelity

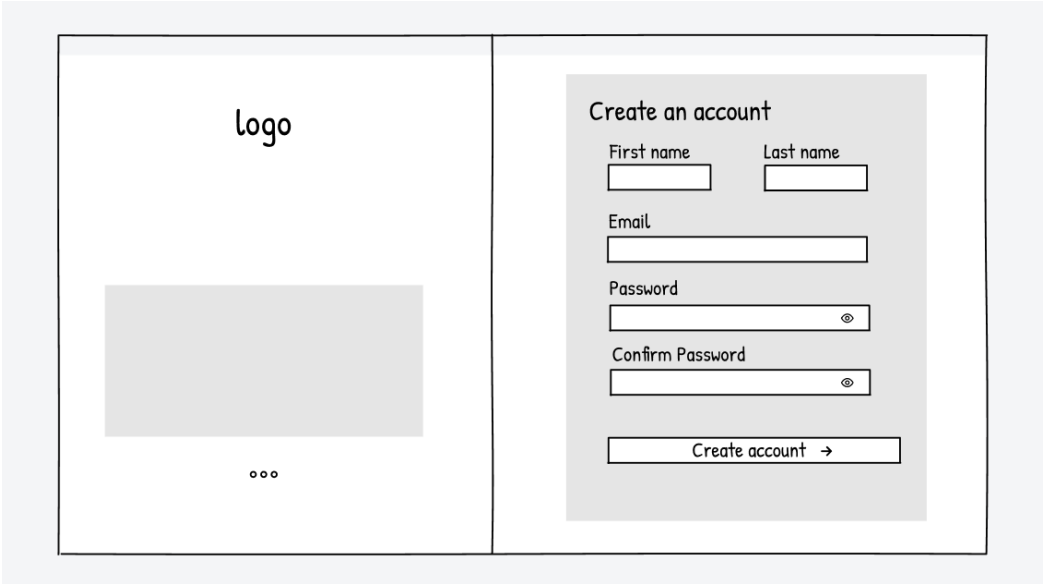

Figure 1 Sign Up - low fidelity

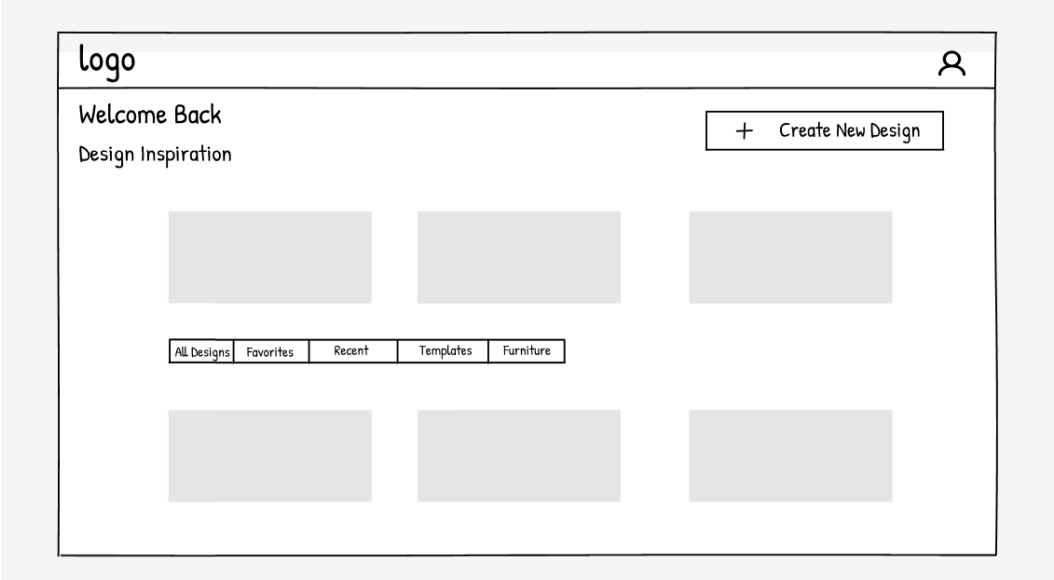

Figure 2 Welcome - low fidelity

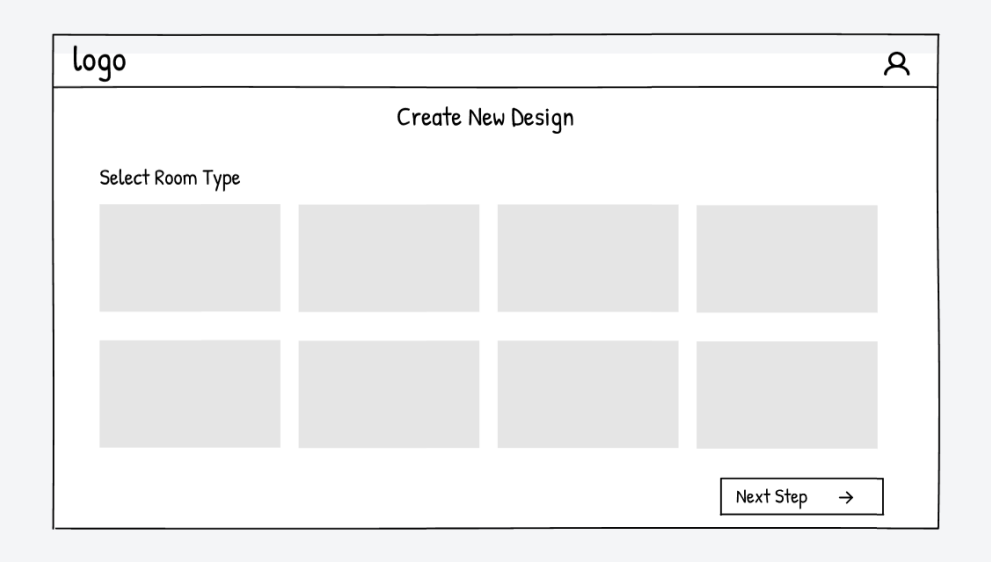

Figure 4 Create new design 1 - low fidelity

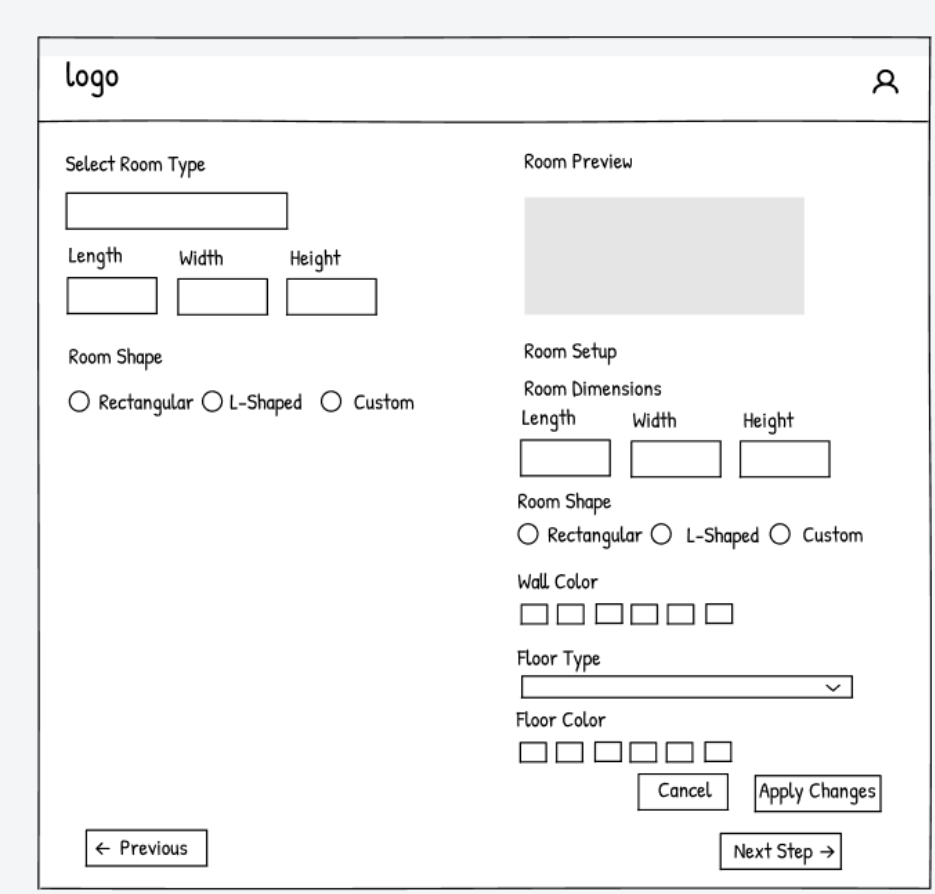

Figure 3 Create new design 2 - low fidelity

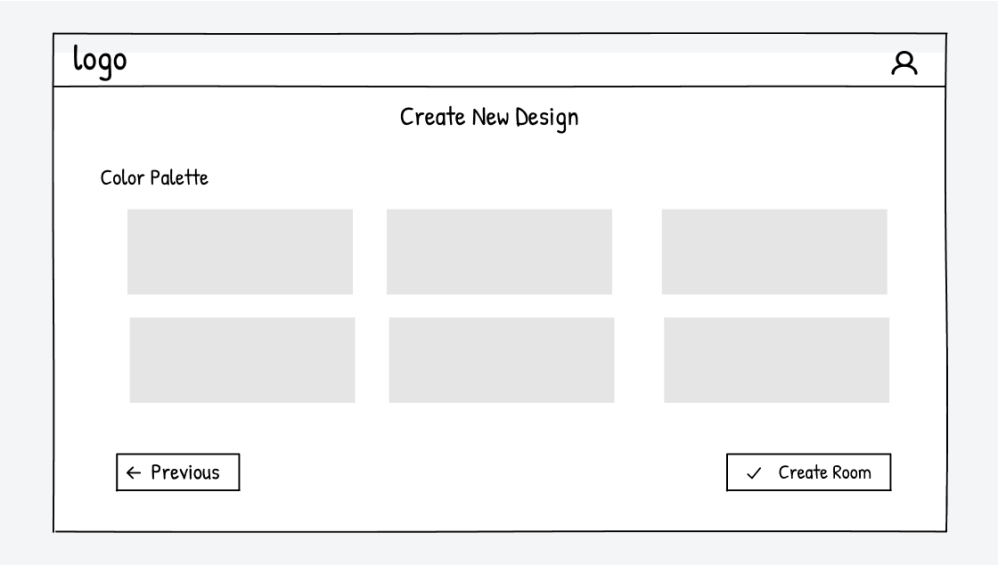

Figure 5 Create new design 3 - low fidelity

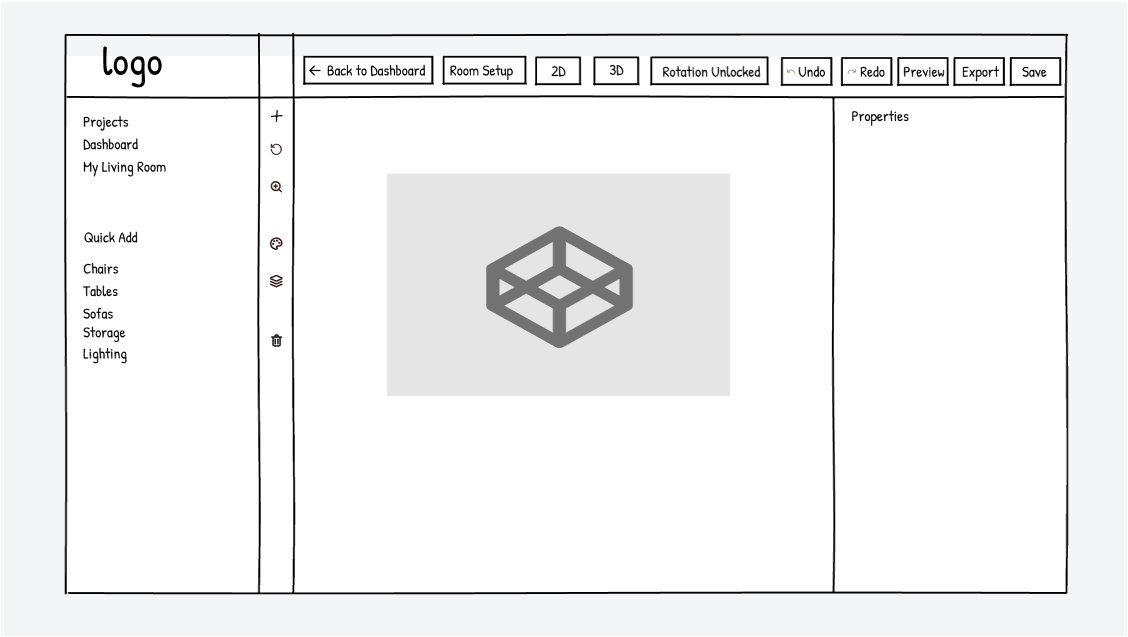

Figure 6 Customize furniture - low fidelity

High – fidelity

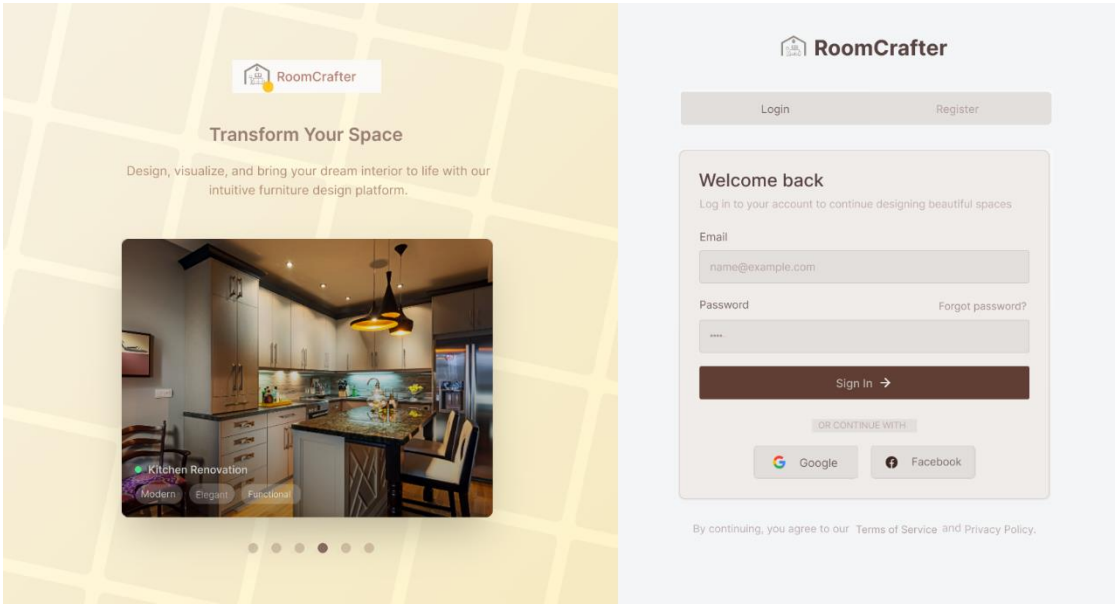

Figure 7 Sign Up - high fidelity

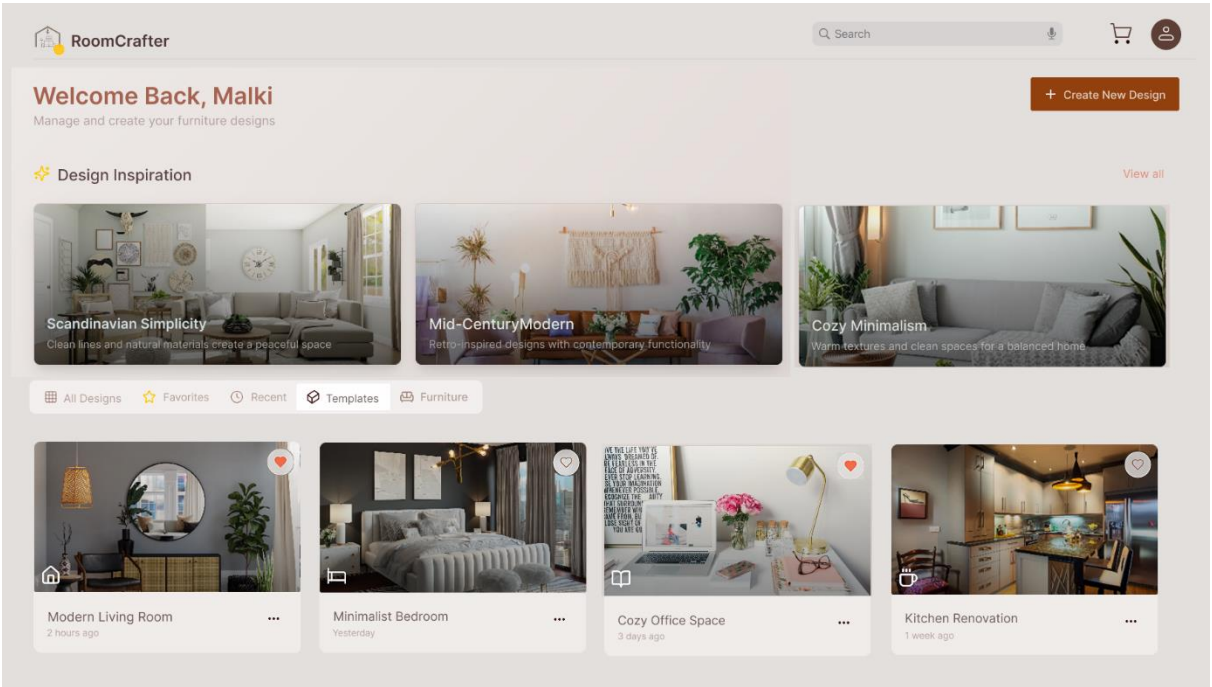

Figure 8 Home - high fidelity

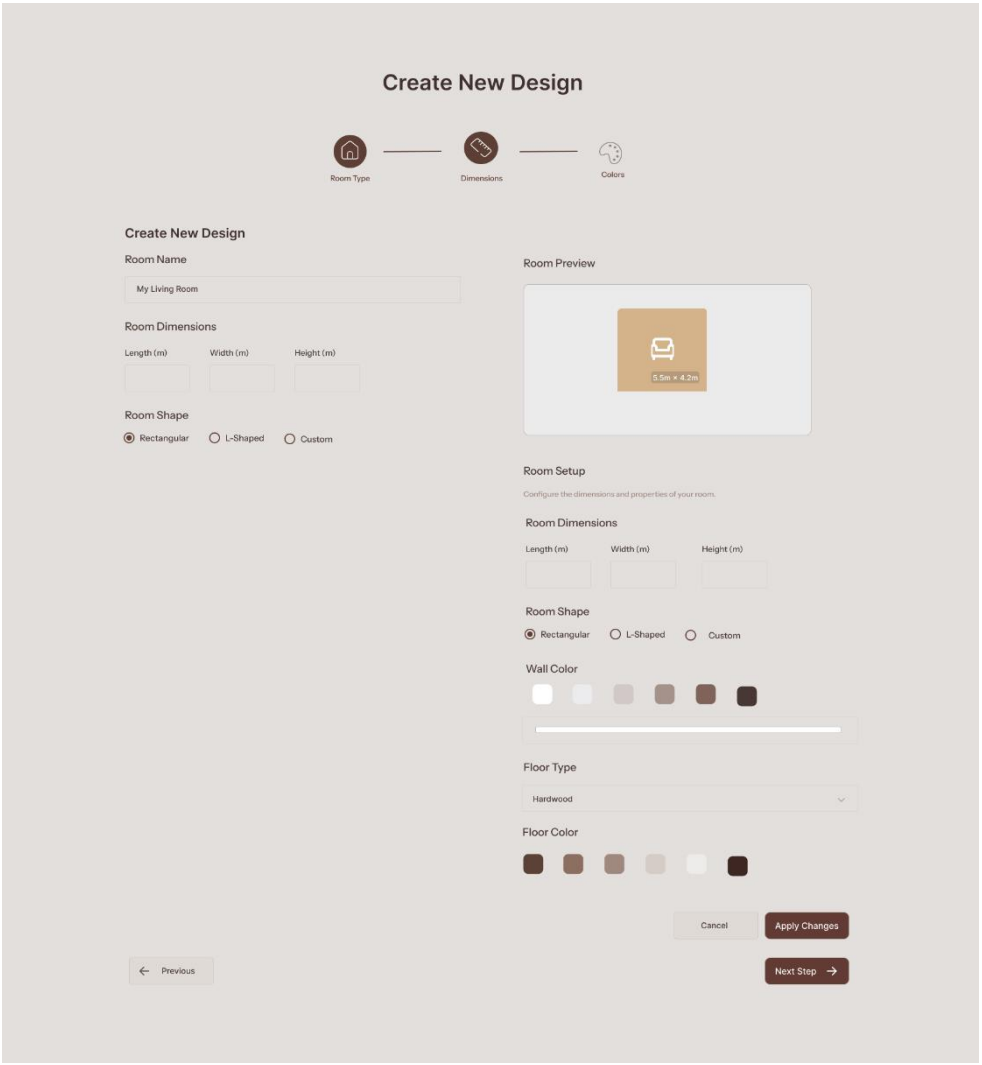

Figure 9 Create New Design - high fidelity

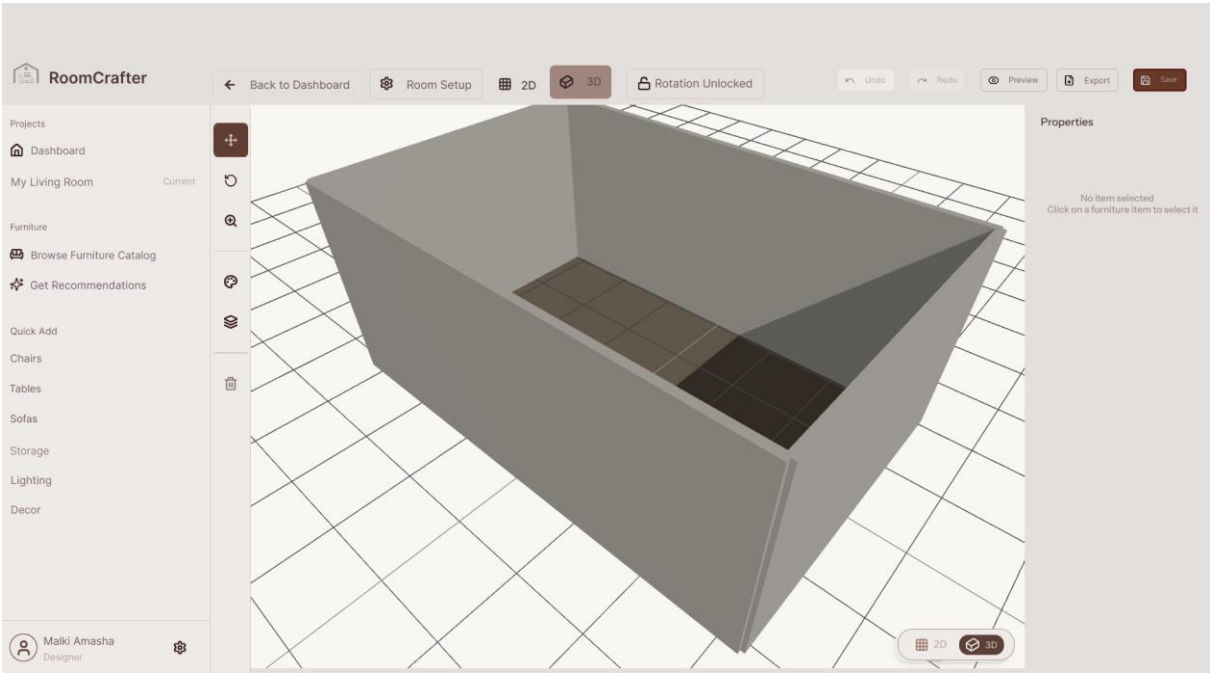

Figure 10 3D View - high fidelity

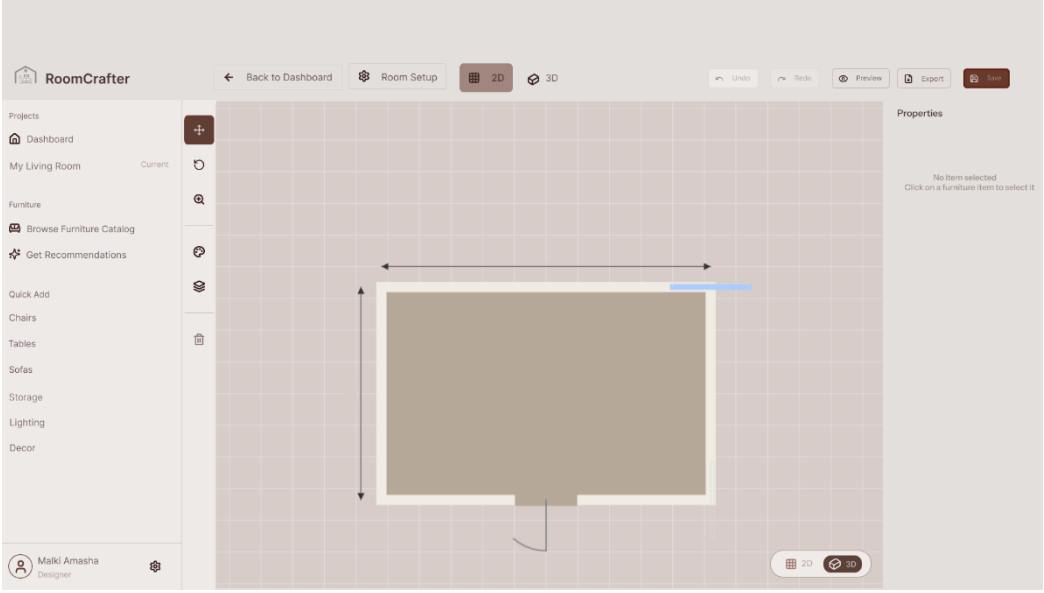

Figure 11 2D view - high fidelity

1.4 Bringing Requirements to Life

1) User Persona 1

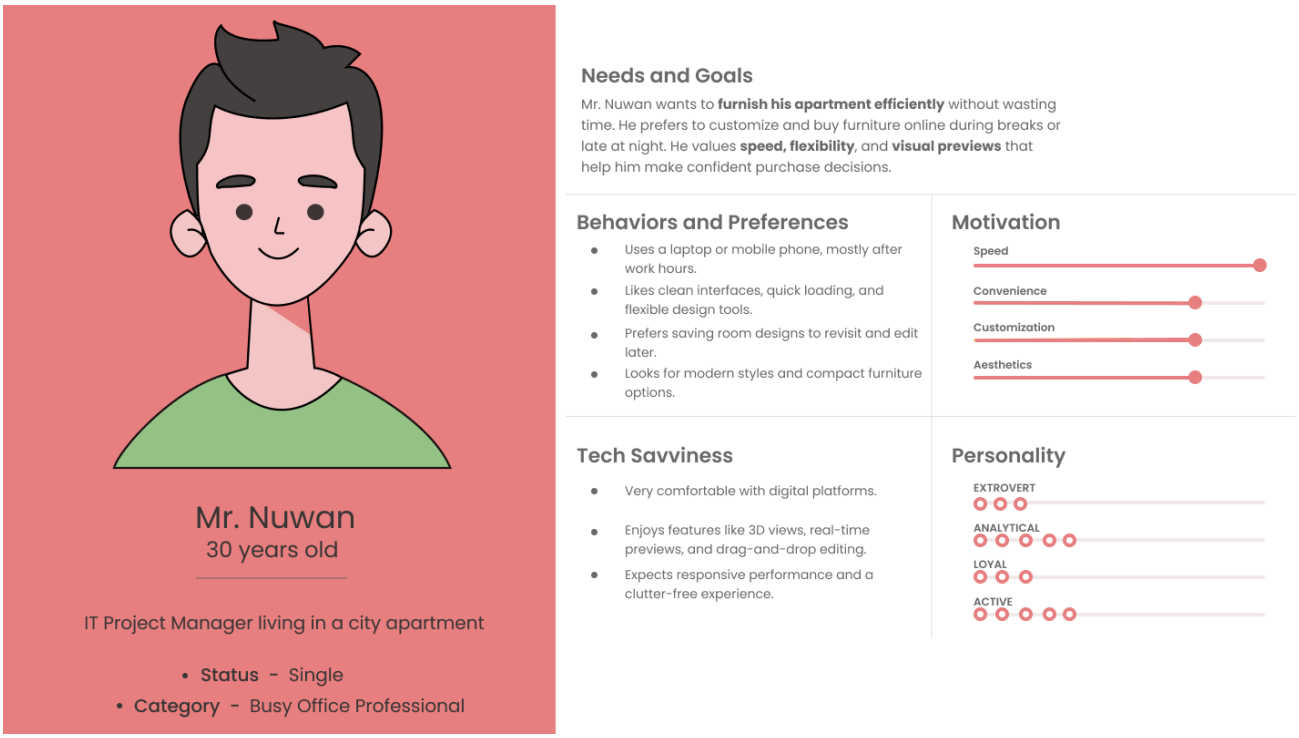

Figure 12 User persona 1

2)User Persona 2

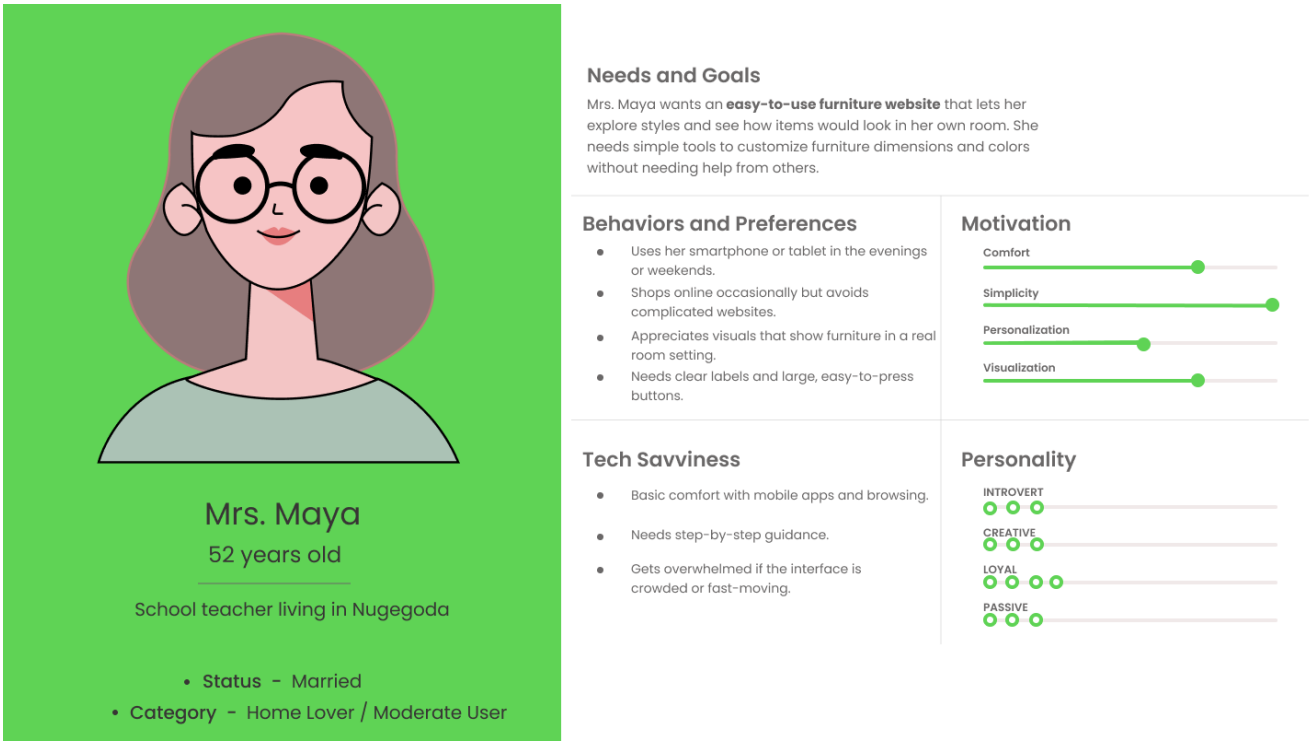

Figure 13 User persona 2

3)User Persona 3

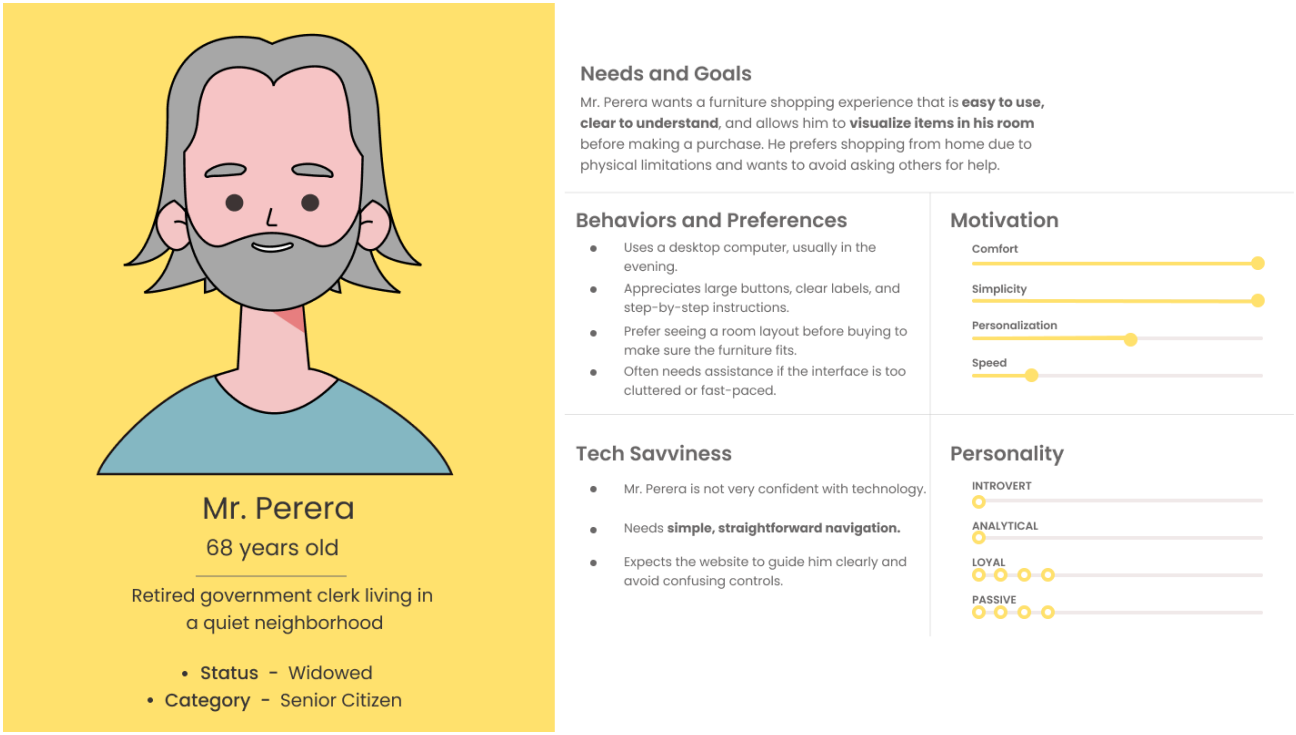

Figure 14 User persona 3

1.4 Storyboards

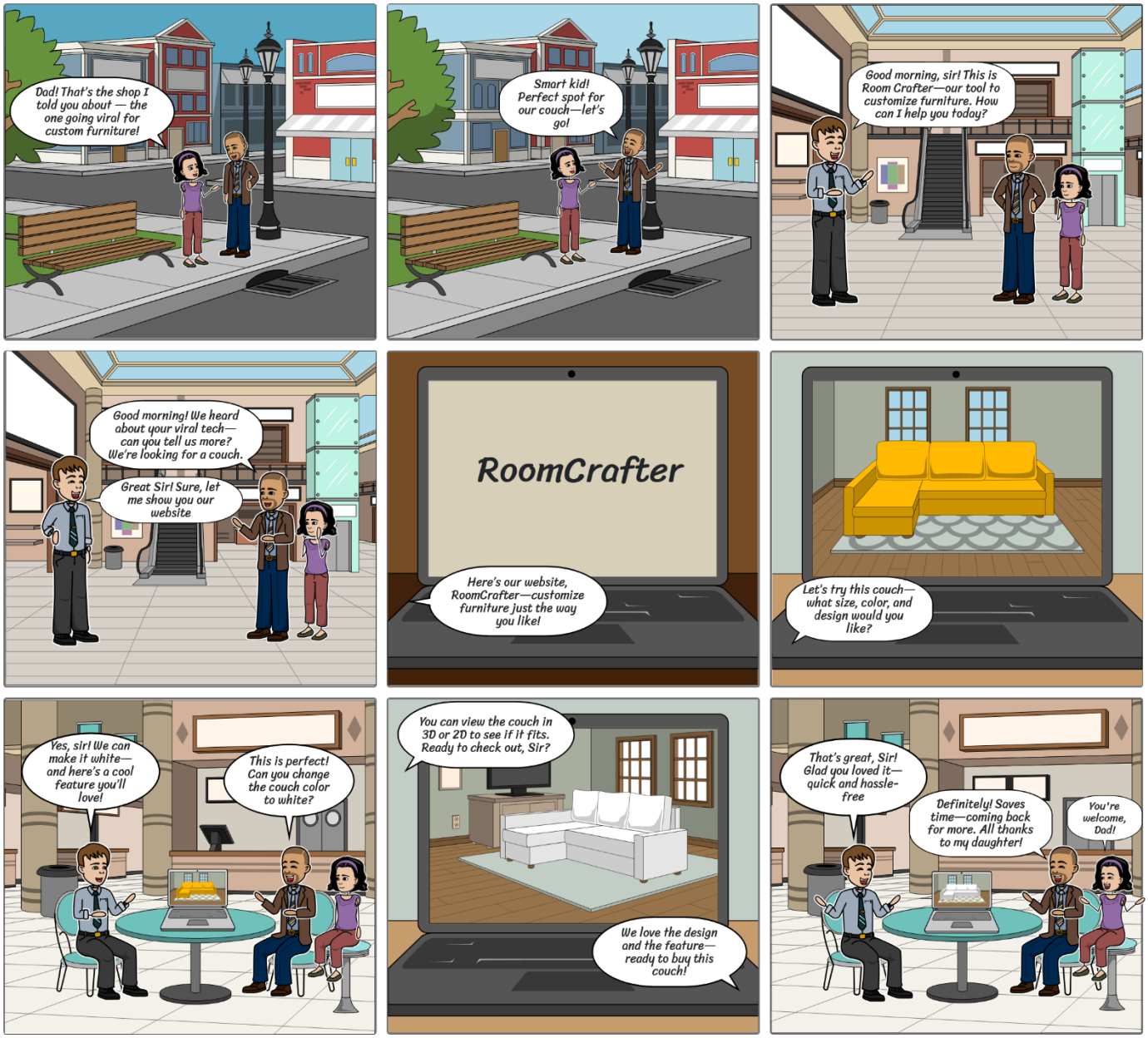

Figure 15 Storyboard example

1.5 Mock Evaluations

Prior to formal user testing, a set of mock testing was conducted, which involved usability and design of the RoomCrafter website. In essence, these mock tests were conducted to assess design flaws and validate foundation usability principles (e.g., clarity, consistency, and navigability). The design team conducted internal task walkthroughs and heuristic evaluations from low fidelity sketches through to hi-fidelity mockups. Additionally, user actions were evaluated in terms of intuitiveness. In particular, selecting furniture, changing attributes, drag and drop of furniture, and saving the room design.

The heuristic evaluation used principles developed by Nielsen and addressed the following principles: feedback from the system, consistency of labels and layout, visibility of options for the user, and avoiding errors. The mock testing revealed a couple of minor usability errors - wrong labelled tools and inconsistent layout when viewed in the preview area. In response to the evaluation, the RoomsCrafter team would address the label on the icons, improve visual spacing, and provide better feedback cues.

The mock evaluations evaluated an important part of the overall design process. They both made sure the design interface was improved, while allowing the design team to rectify a usability issue(s) prior to engaging link users, while trying to ensure, to the best extent possible, the users overall experience would be seamless and polished.

1.6 User Feedback

We conducted an online survey to capture user feedback on how usable RoomCrafter is. The survey assisted us in gathering feedback on usability elements, which allowed us to make minor design changes.

Here is the feedback we received:

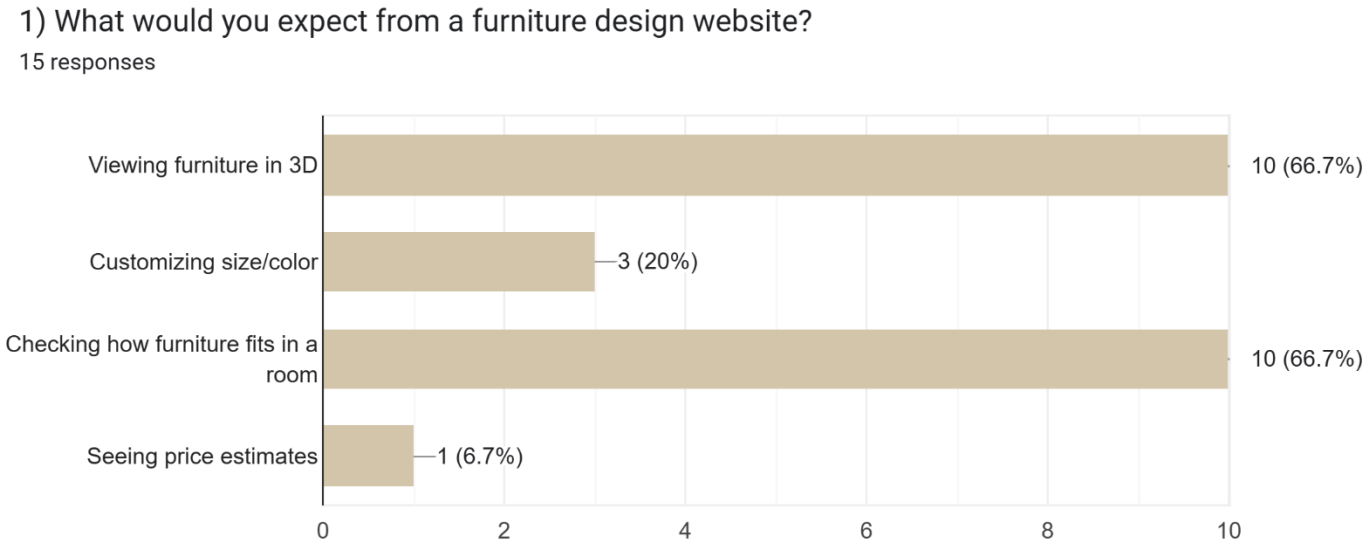

Figure 16 Survey question 1

Figure 17 Survey question 2

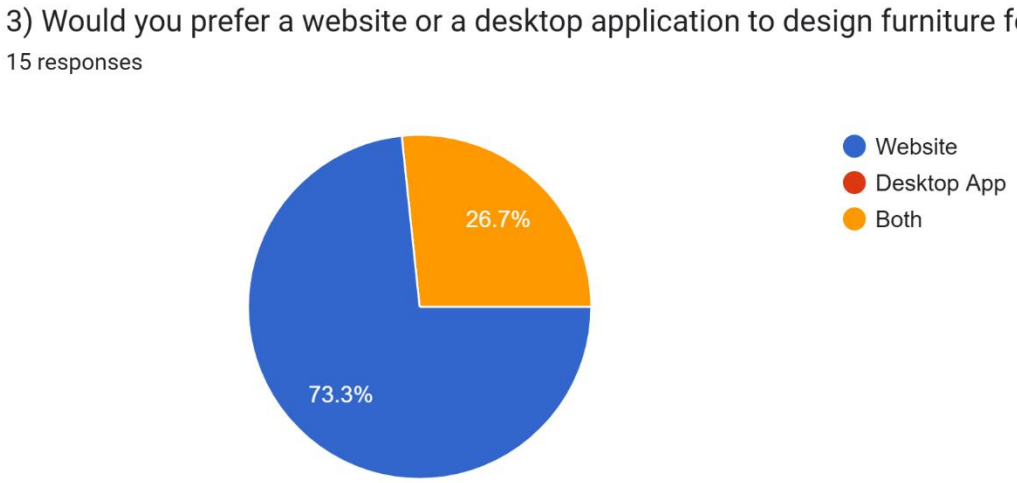

Figure 18 Survey question 3

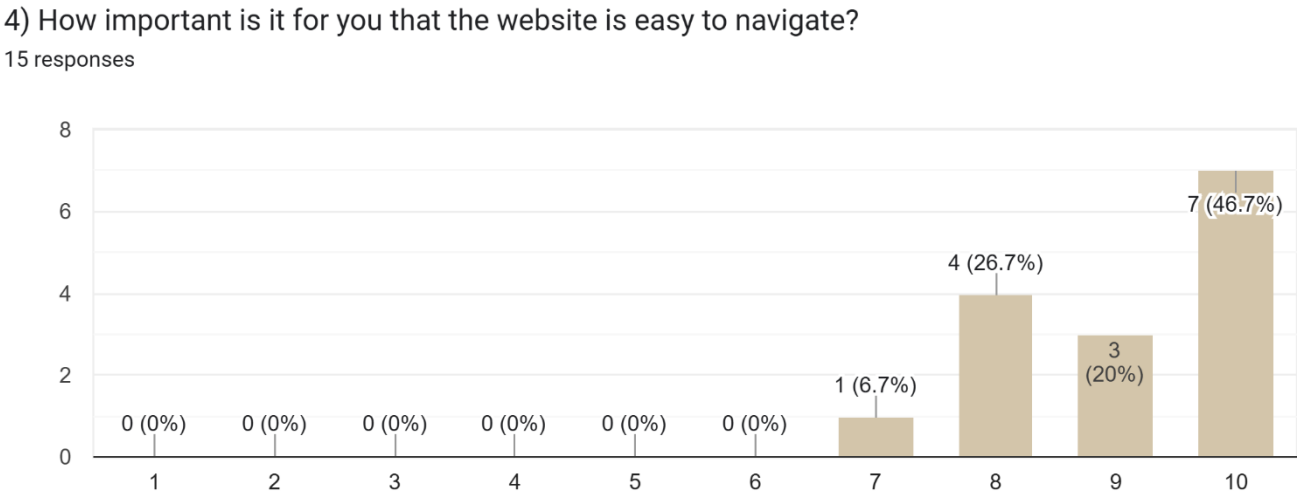

Figure 19 Survey question 4

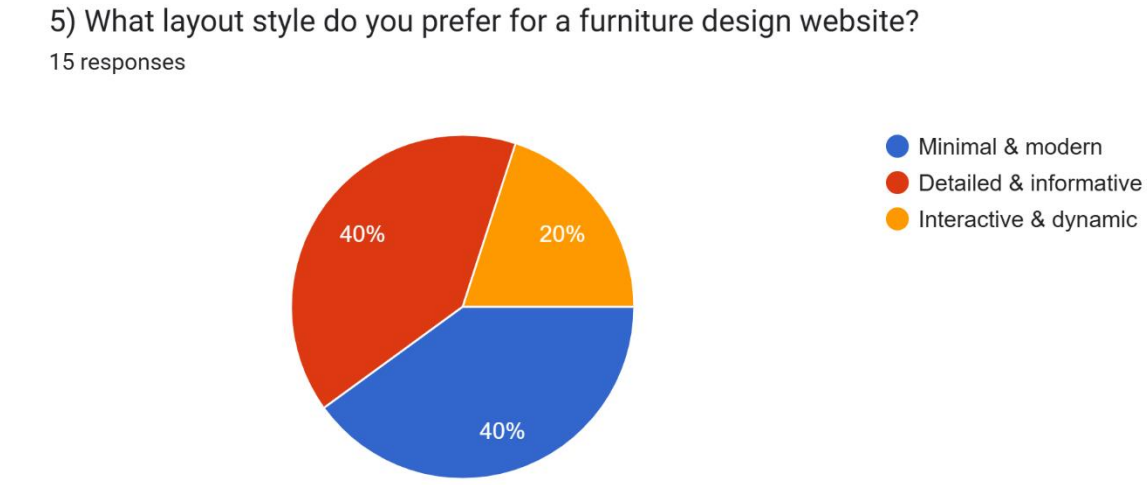

Figure 20 Survey question 5

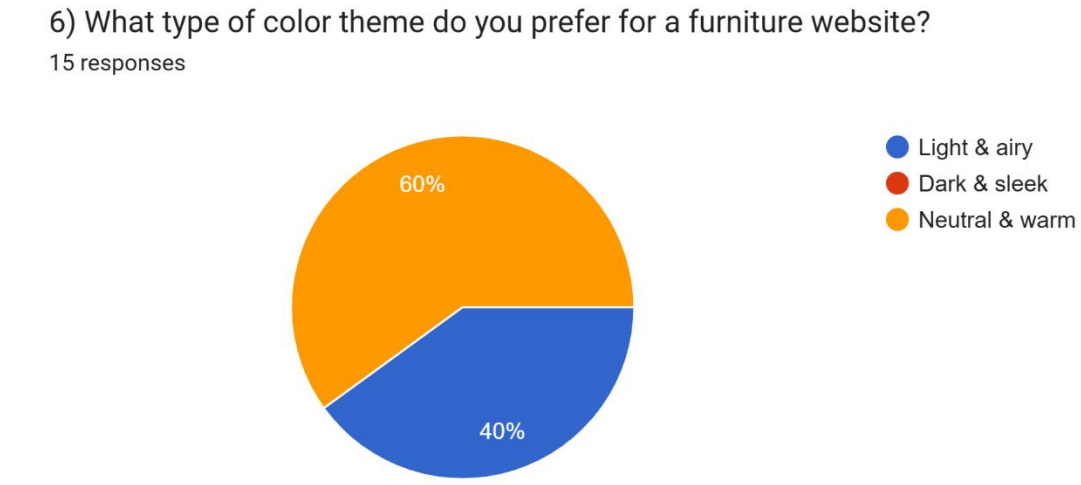

Figure 21 Survey question 6

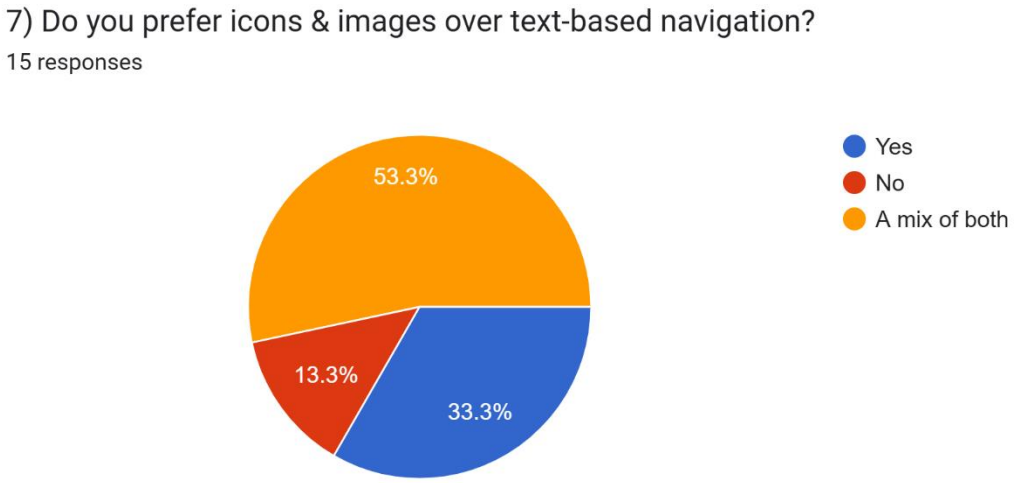

Figure 22 Survey question 7

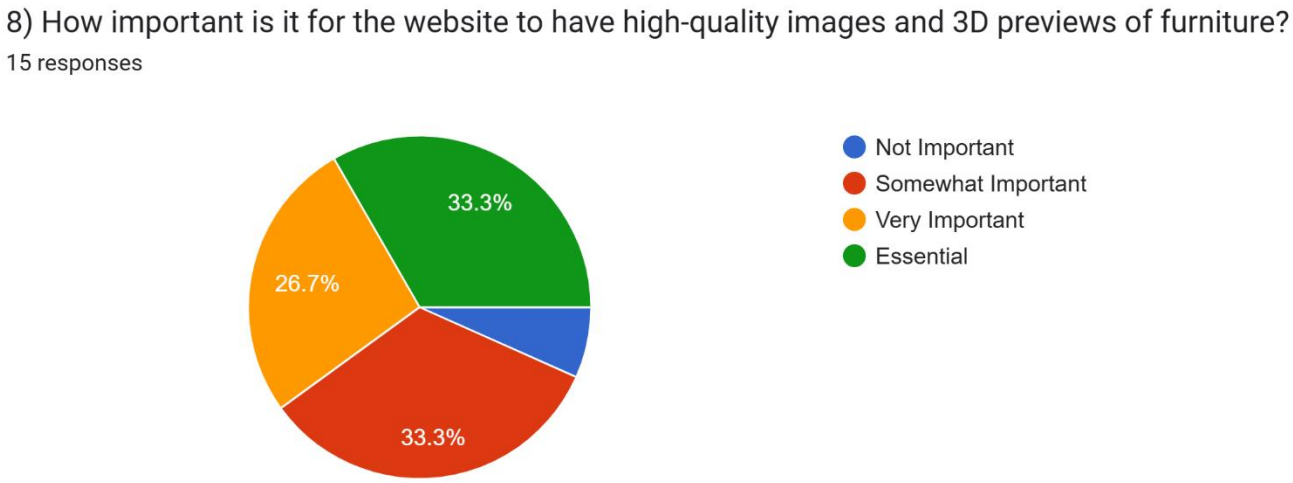

Figure 23 Survey question 8

9) Would you find it helpful to visually design a room layout using interactive tools?

15 responses

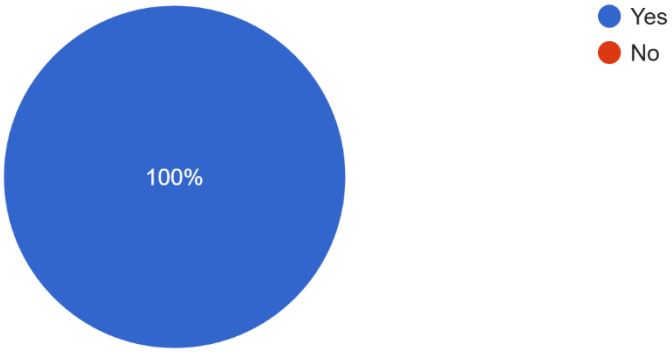

Figure 24 Survey question 9

10) What are some UI features that would improve your experience on the website?

15 responses

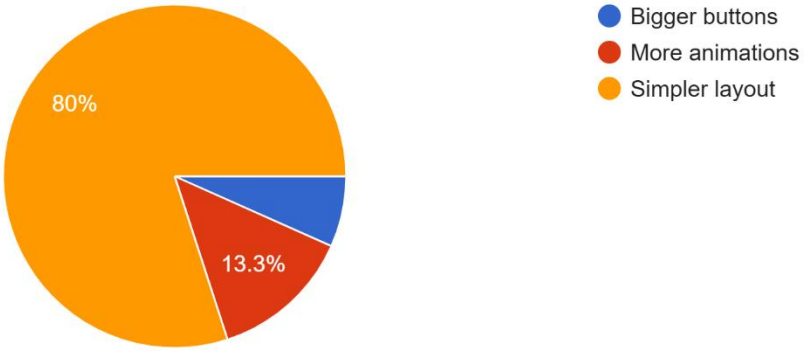

Figure 25 Survey question 10

1.8 Feedback and Updates

Regarding the survey user feedback we used to mold RoomCrafter, we made adjustments to define the user experience. In considering user requests, we added the ability to see furniture in 3D, to change size, to change colors and to see how furniture occupied a room layout.

The platform was designed as a website based on the preference of the majority of users. The site design is clean, minimalistic, modern and encourages user exploration. We used a neutral and warm color scheme to meet user suggestions in the site. Navigation on the site includes both icon and text features for user access and we developed the site for user exploration.

We used professional quality images with interactive 3D previews of furniture fixtures. We added tools for users to see visually how the furniture used might orient in a room.

2. Methods and Technology

Platform

RoomCrafter is a web-based furniture customization and visualization platform that enables a very smooth and responsive experience across all devices. The Next.js framework is used to develop RoomCrafter and has a hybrid rendering strategy that utilizes server-side rendering to enhance page load speed and SEO, while providing client-side interactivity for real-time layout updates and seamless transitions.

The platform follows a mobile-first responsive design, allowing users to access full features whether on desktop, tablet, or smartphone. This flexibility supports designing spaces from home, showrooms, or while on the move.

As a web-native solution, it requires no installations—users can customize and manage layouts directly in the browser. Cloud deployment ensures high uptime, scalability, and rapid feature rollouts. Whether for personal projects or professional design work, RoomCrafter offers an accessible and future-ready tool that simplifies interior design while enhancing creativity and confidence in furniture planning decisions.

Technology

RoomCrafter is developed using Next.js and TypeScript, ensuring efficient, type-safe development. Tailwind CSS enables rapid, utility-first styling for consistent, responsive UI design. For 3D visualization, the app integrates Three.js through @react-three/fiber, allowing real-time previews of furniture in a room, enhanced with @react-three/drei for simplified controls and lighting. UI components are built with shadcn/ui, leveraging Radix UI and Tailwind for accessibility and reusability. Icons are rendered using Lucide React. Lightweight data such as user preferences and saved layouts are stored in localStorage, enabling session continuity without server-side dependencies, making RoomCrafter both interactive and user-centric.

Architecture

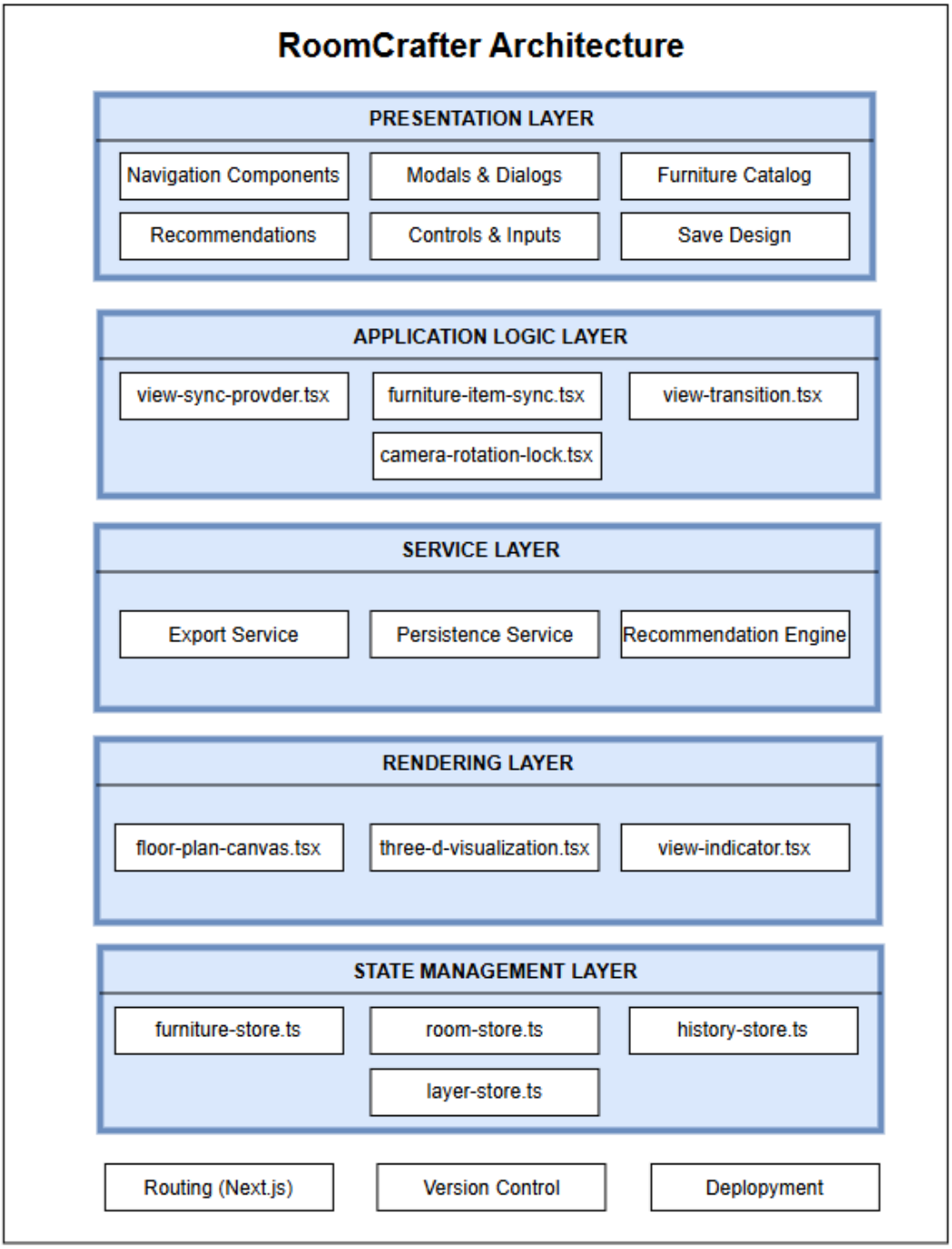

Figure 26 Architecture

Coding details

1) **Modules for Managing State**
   Tracks and manages the furniture items chosen by a user. It includes functions to add, change, or delete items, while keeping track of their sizes, locations, and styles applied.

- furniture-store.ts

```
export const useFurnitureStore = create<FurnitureState>((set, get) => ({
  furnitureItems: [],

  addFurnitureItem: (item) =>
    set((state) => ({
      furnitureItems: [...state.furnitureItems, item],
    })),

  updateFurnitureItem: (id, updates) =>
    set((state) => ({
      furnitureItems: state.furnitureItems.map((item) =>
        (item.id === id ? { ...item, ...updates } : item)),
    })),

  removeFurnitureItem: (id) =>
    set((state) => ({
      furnitureItems: state.furnitureItems.filter((item) => item.id !== id)
    })),

  getFurnitureItem: (id) => {
    return get().furnitureItems.find((item) => item.id === id)
  },

  setFurnitureItems: (items) => set({ furnitureItems: items }),
}))
```

Figure 27 code 1

- room-store.ts

```
interface RoomState {
  roomDimensions: RoomDimensions
  wallColor: string
  floorColor: string
  floorType: string
  roomShape: RoomShape
  updateRoomDimensions: (dimensions: RoomDimensions) => void
  updateWallColor: (color: string) => void
  updateFloorColor: (color: string) => void
  updateFloorType: (type: string) => void
  updateRoomShape: (shape: RoomShape) => void
}

export const useRoomStore = create<RoomState>((set) => ({
  roomDimensions: {
    length: 6.5,
    width: 4.2,
    height: 2.8,
  },
  wallColor: "#f0ece3",
  floorColor: "#b5a898",
  floorType: "hardwood",
  roomShape: "rectangular",

  updateRoomDimensions: (dimensions) => {
    // Validate dimensions to ensure they're positive numbers
    const validatedDimensions = {
      length: Math.max(0.1, dimensions.length),
      width: Math.max(0.1, dimensions.width),
      height: Math.max(0.1, dimensions.height),
    }
```

Figure 28 code 2

```
    set({ roomDimensions: validatedDimensions })
  },
  updateWallColor: (color) => set({ wallColor: color }),
  updateFloorColor: (color) => set({ floorColor: color }),
  updateFloorType: (type) => set({ floorType: type }),
  updateRoomShape: (shape) => set({ roomShape: shape }),
}))
```

Figure 29 code 3

- history-store.ts

```
// Save the current state to history
saveState: (state) => {
  // Create a deep copy of the state to avoid reference issues
  const stateCopy = JSON.parse(JSON.stringify(state))

  set((prev) => {
    // Only add to history if the state is different from the last one
    if (prev.past.length > 0) {
      const lastState = prev.past[prev.past.length - 1]
      if (JSON.stringify(lastState) === JSON.stringify(stateCopy)) {
        return prev // No change, return the current state
      }
    }

    return {
      past: [...prev.past, stateCopy],
      future: [], // Clear future when a new state is added
      canUndo: true,
      canRedo: false,
    }
  })
},

undo: () => {
  const { past } = get()

  if (past.length <= 1) {
    return null // Nothing to undo or only initial state left
  }
```

Figure 30 code 4

- layer-store.ts

```
export const useLayerStore = create<LayerState>((set) => ({
  layers: [
    { id: "walls-floor", name: "Walls & Floor", visible: true },
    { id: "furniture", name: "Furniture", visible: true },
    { id: "lighting", name: "Lighting", visible: true },
    { id: "decor", name: "Decor", visible: true },
    { id: "measurements", name: "Measurements", visible: true },
  ],

  toggleLayerVisibility: (id) => {
    set((state) => ({
      layers: state.layers.map((layer) =>
        (layer.id === id ? { ...layer, visible: !layer.visible } : layer)),
    }))
  },

  addLayer: (layer) => {
    set((state) => ({
      layers: [...state.layers, layer],
    }))
  },

  removeLayer: (id) => {
    set((state) => ({
      layers: state.layers.filter((layer) => layer.id !== id),
    }))
  },
}))
```

Figure 31 code 5

2) Visualization Components

```
// Convert world coordinates to screen coordinates
const worldToScreen = (x: number, z: number) => {
  return {
    x: (x * scale + offset.x + canvasSize.width / 2) * window.devicePixel
    y: (z * scale + offset.y + canvasSize.height / 2) * window.devicePixe
  }
}

// Convert screen coordinates to world coordinates
const screenToWorld = (x: number, y: number) => {
  return {
    x: (x / window.devicePixelRatio - offset.x - canvasSize.width / 2) /
    z: (y / window.devicePixelRatio - offset.y - canvasSize.height / 2) /
  }
}

// Draw the room and furniture
const draw = () => {
  const canvas = canvasRef.current
  if (!canvas) return

  const ctx = canvas.getContext("2d")
  if (!ctx) return

  // Clear canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height)

  // Draw room, walls, furniture, etc.
  // ...
}
```

Figure 32 code 6

three-d-visualization.tsx

```
export function ThreeDVisualization({
  roomDimensions,
  roomShape,
  wallColor,
  floorColor,
  floorType,
  furnitureItems,
  selectedItemId,
  setSelectedItemId,
  activeTool,
  updateFurnitureItem,
  layers = [],
  glossiness = 0.5,
  textureScale = 1,
  isPreviewMode = false,
  isCameraRotationLocked = false,
}: ThreeDVisualizationProps) {
  // Get the view sync context
  const { syncViewTransform, currentTransform } = useViewSync()

  return (
    <Canvas shadows camera={{ position: [5, 5, 5], fov: 50 }}>
      <color attach="background" args={["#f8f7f4"]} />
      <ambientLight intensity={0.5} />
      <directionalLight
        position={[10, 10, 5]}
        intensity={1}
        castShadow
        shadow-mapSize-width={2048}
        shadow-mapSize-height={2048}
      />
```

Figure 33 code 7

view-transition.tsx

```tsx
export function ViewTransition({ children }: ViewTransitionProps) {
  const { viewMode } = useViewSync()
  const [prevViewMode, setPrevViewMode] = useState(viewMode)
  const [isTransitioning, setIsTransitioning] = useState(false)

  useEffect(() => {
    if (viewMode !== prevViewMode) {
      setIsTransitioning(true)
      const timer = setTimeout(() => {
        setIsTransitioning(false)
        setPrevViewMode(viewMode)
      }, 500) // Match this with the animation duration
      return () => clearTimeout(timer)
    }
  }, [viewMode, prevViewMode])

  return (
    <AnimatePresence mode="wait">
      <motion.div
        key={viewMode}
        initial={{ opacity: 0 }}
        animate={{ opacity: 1 }}
        exit={{ opacity: 0 }}
        transition={{ duration: 0.5 }}
        className="w-full h-full"
      >
        {children}
      </motion.div>
    </AnimatePresence>
  )
}
```

Figure 34 code 8

3) UI Components

A group of interactive tools makes up the user interface pieces used to navigate, configure, and interact within RoomCrafter:

```
// Handle adding selected furniture to the room
const handleAddToRoom = () => {
  // For each selected item, prepare it for adding to the room
  const furnitureToAdd = Object.entries(selectedItems).map(([itemId, custo
    const item = furnitureItems.find((f) => f.id === itemId)!
    return {
      id: `furniture-${Date.now()}-${itemId}`,
      type: item.category.replace(/s$/, ""), // Remove trailing 's' from c
      name: item.name,
      position: { x: 0, y: 0, z: 0 }, // Default position, will be adjuste
      dimensions: item.dimensions,
      rotation: { x: 0, y: 0, z: 0 },
      color: customization.color,
      material: customization.material,
      originalItem: item, // Keep reference to original catalog item
    }
  })

  // Pass the selected furniture back to the parent component
  onSelectFurniture(furnitureToAdd)
}

return (
  <div className="flex flex-col h-full bg-light-sand-beige">
```

Figure 35 code 9

furniture-recommendations.tsx

```tsx
export function FurnitureRecommendations({
  roomDimensions,
  onSelectRecommendation,
  onClose,
}: FurnitureRecommendationsProps) {
  const [selectedSet, setSelectedSet] = useState<string | null>(null)
  const [likedSets, setLikedSets] = useState<string[]>([])
  const [dislikedSets, setDislikedSets] = useState<string[]>([])
  const [isClient, setIsClient] = useState(false)

  // Handle liking a recommendation set
  const handleLike = (setId: string) => {
    setLikedSets((prev) => [...prev.filter((id) => id !== setId), setId])
    setDislikedSets((prev) => prev.filter((id) => id !== setId))
  }

  // Handle applying the selected recommendation set
  const handleApplyRecommendation = () => {
    if (!selectedSet) return

    const set = recommendationSets.find((s) => s.id === selectedSet)
    if (!set) return

    // Convert recommendation items to furniture items
    const furnitureItems = set.items.map((item, index) => {
      // Calculate positions to spread items across the room
      const xPos = roomDimensions.length / 2 + (index % 2 === 0 ? -1 : 1) *
      const zPos = roomDimensions.width / 2 + (index % 3 === 0 ? -1 : 1) *
```

Figure 36 code 10

save-design-dialog.tsx

```tsx
"use client"

import { useState } from "react"
import { Button } from "@/components/ui/button"
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
} from "@/components/ui/dialog"
import { Input } from "@/components/ui/input"
import { Label } from "@/components/ui/label"
import { GlossyCloseButton } from "@/components/ui/glossy-close-button"

interface SaveDesignDialogProps {
  open: boolean
  onOpenChange: (open: boolean) => void
  onSave: (name: string) => void
  defaultName?: string
}

export function SaveDesignDialog({ open, onOpenChange, onSave, defaultName
  const [designName, setDesignName] = useState(defaultName || "")

  const handleSave = () => {
    onSave(designName || "Untitled Design")
    onOpenChange(false)
  }
```

Figure 37 code 11

4) Main Logic and Coordination

These files help keep interactions and views consistent even in tricky scenarios like switching modes or running complex processes:

view-sync-provider.tsx

```tsx
"use client"

import type React from "react"
import { createContext, useContext, useState, useEffect } from "react"
import { useFurnitureStore } from "@/lib/furniture-store"
import { useRoomStore } from "@/lib/room-store"

interface ViewSyncContextType {
  viewMode: "2d" | "3d"
  setViewMode: (mode: "2d" | "3d") => void
  selectedItemId: string | null
  setSelectedItemId: (id: string | null) => void
  syncViewTransform: (transform: ViewTransform) => void
  currentTransform: ViewTransform
  convertCoordinates: (
    position: { x: number; y?: number; z: number },
    from: "2d" | "3d",
    to: "2d" | "3d",
  ) => { x: number; y: number; z: number }
}

interface ViewTransform {
  position?: { x: number; y: number }
  scale?: number
  rotation?: number
}
```

Figure 38 code 12

furniture-item-sync.tsx

```
      // Check if position needs to be constrained
      if (item.position.x > maxX || item.position.x < minX || item.position
        updates.position = {
          x: Math.max(minX, Math.min(maxX, item.position.x)),
          y: viewMode === "2d" ? 0 : item.position.y,
          z: Math.max(minZ, Math.min(maxZ, item.position.z)),
        }
        needsUpdate = true
      } else if (viewMode === "2d" && item.position.y !== 0) {
        // In 2D view, ensure y is always 0
        updates.position = {
          ...item.position,
          y: 0,
        }
        needsUpdate = true
      }

      // Apply updates if needed
      if (needsUpdate) {
        updateFurnitureItem(item.id, updates)
      }
    })
  }, [viewMode, furnitureItems, roomDimensions, updateFurnitureItem])

  return null
}
```

Figure 39 code 13

Implementation and Testing

Implementation

1) Home Page

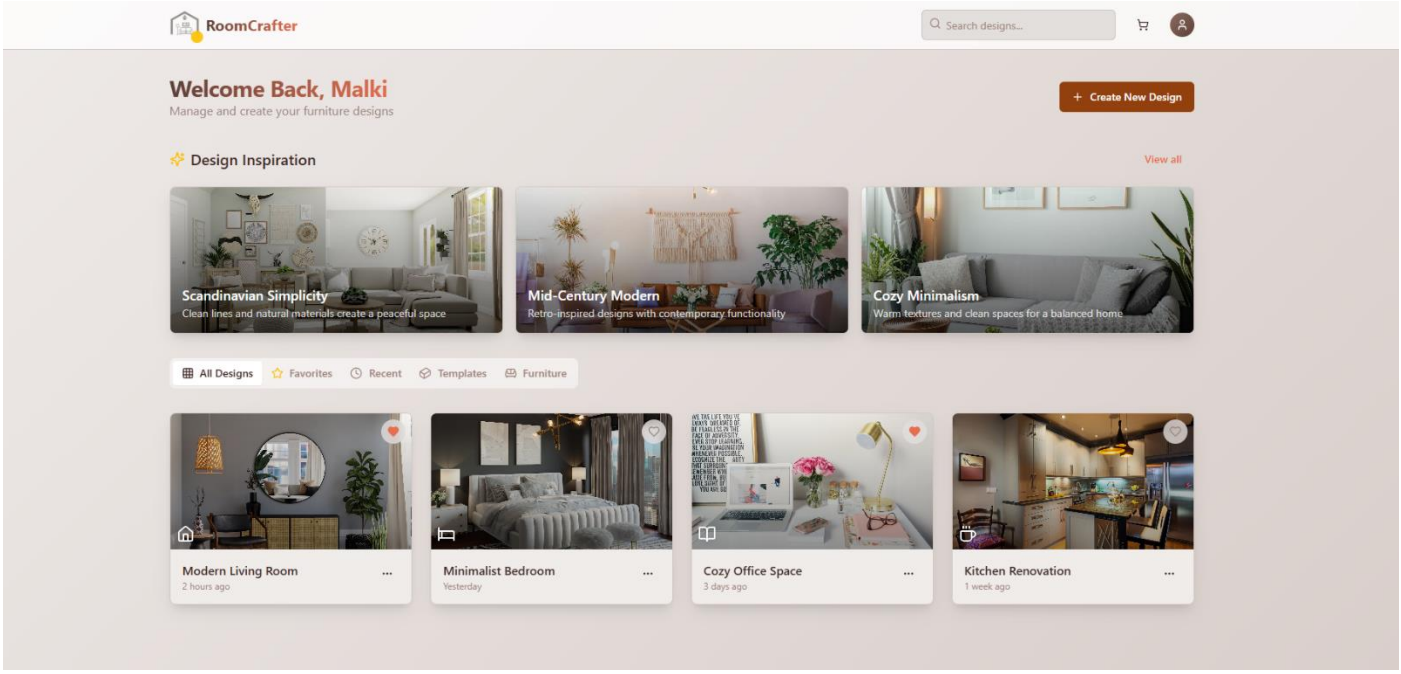

Figure 40 Home Screen

2) Favourites Page

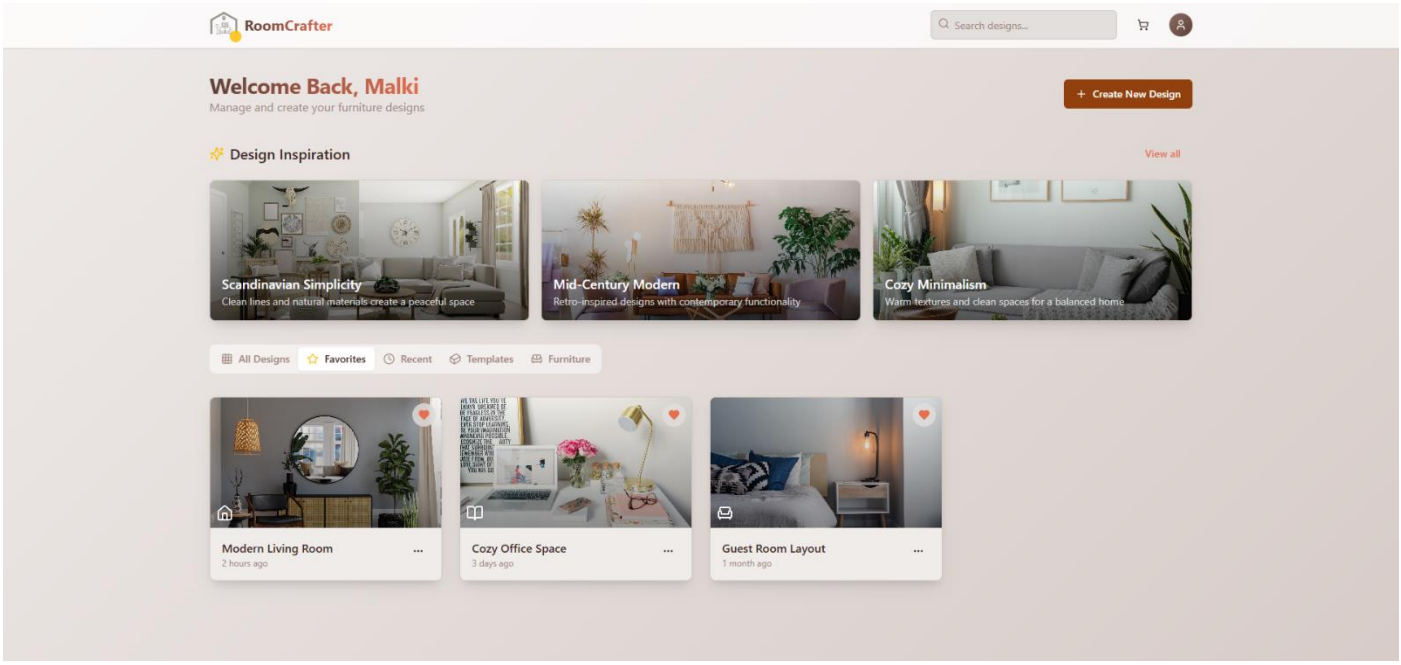

Figure 41 Favorites Section

3) Recent Page

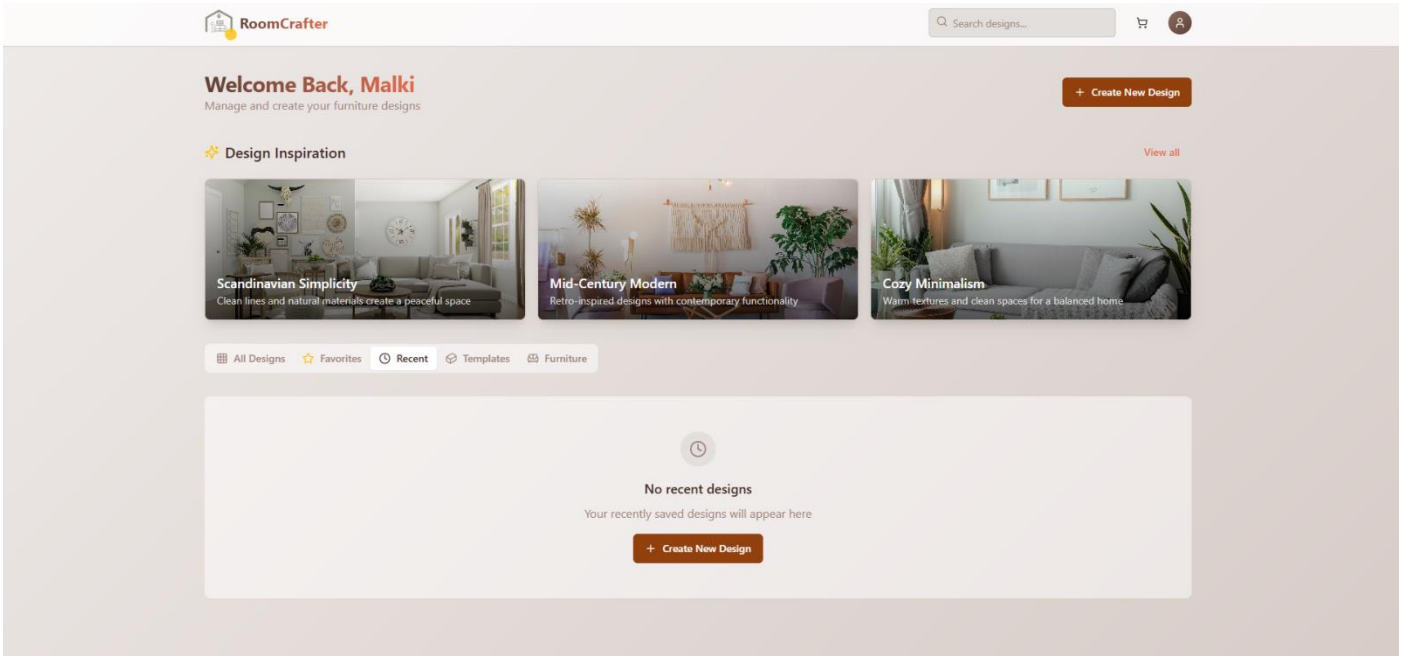

Figure 42 Recents Section

4) Templates Page

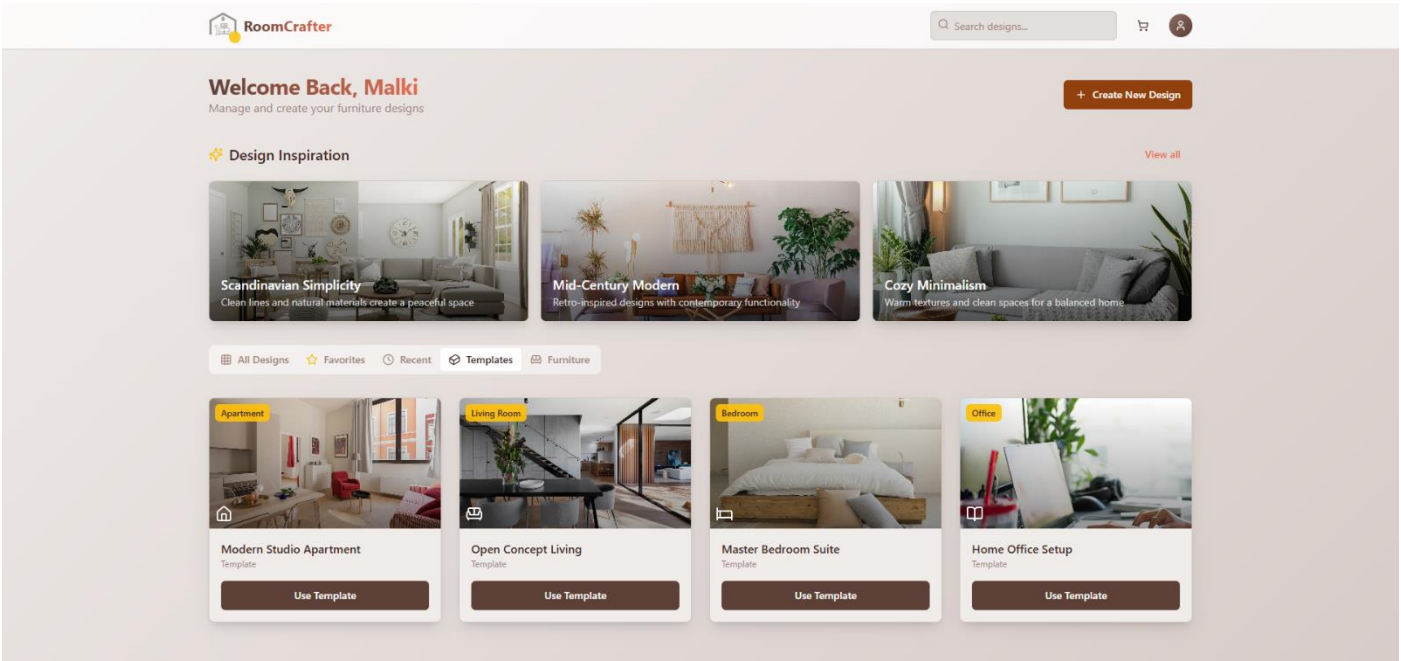

Figure 43 Templates Section

5) Furniture Catalogue

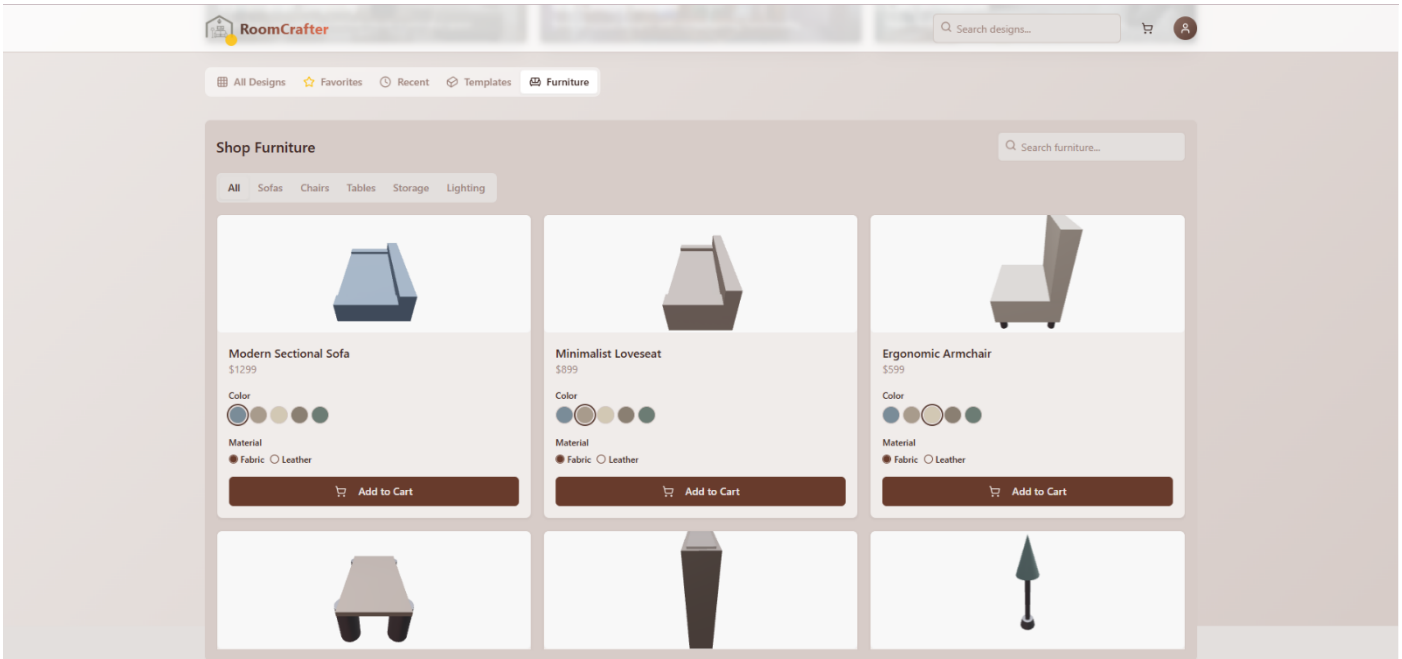

Figure 44 Catalogue Section

6) Create New Design – Room Type

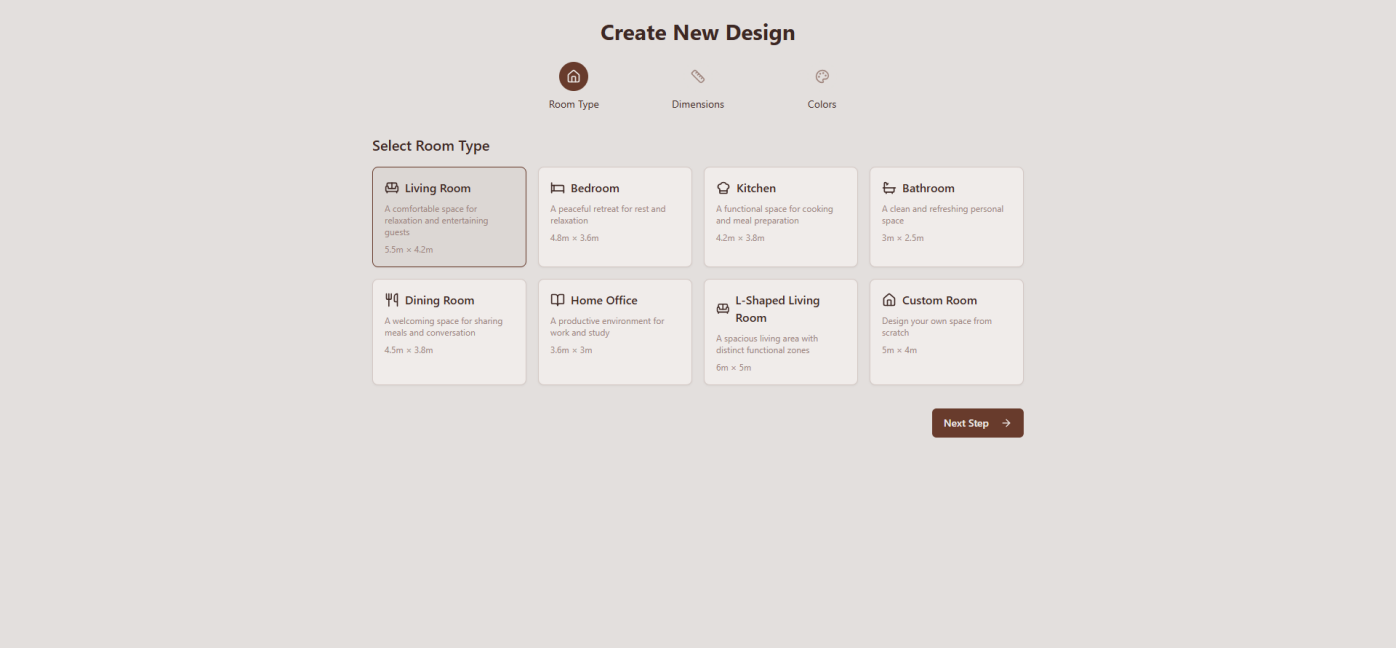

Figure 45 Customization Screen 1

7) Create New Design – Dimensions

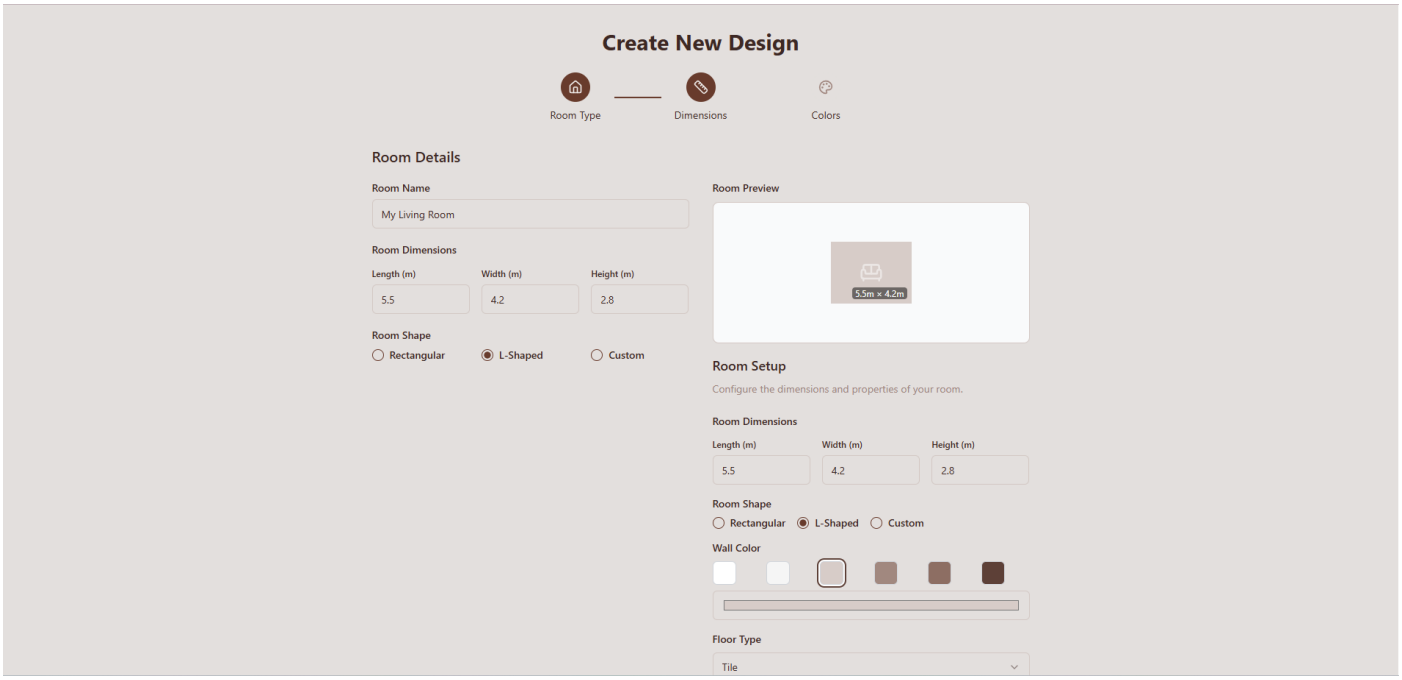

Figure 46 Customization Screen 2

8) Create New Design – Colors

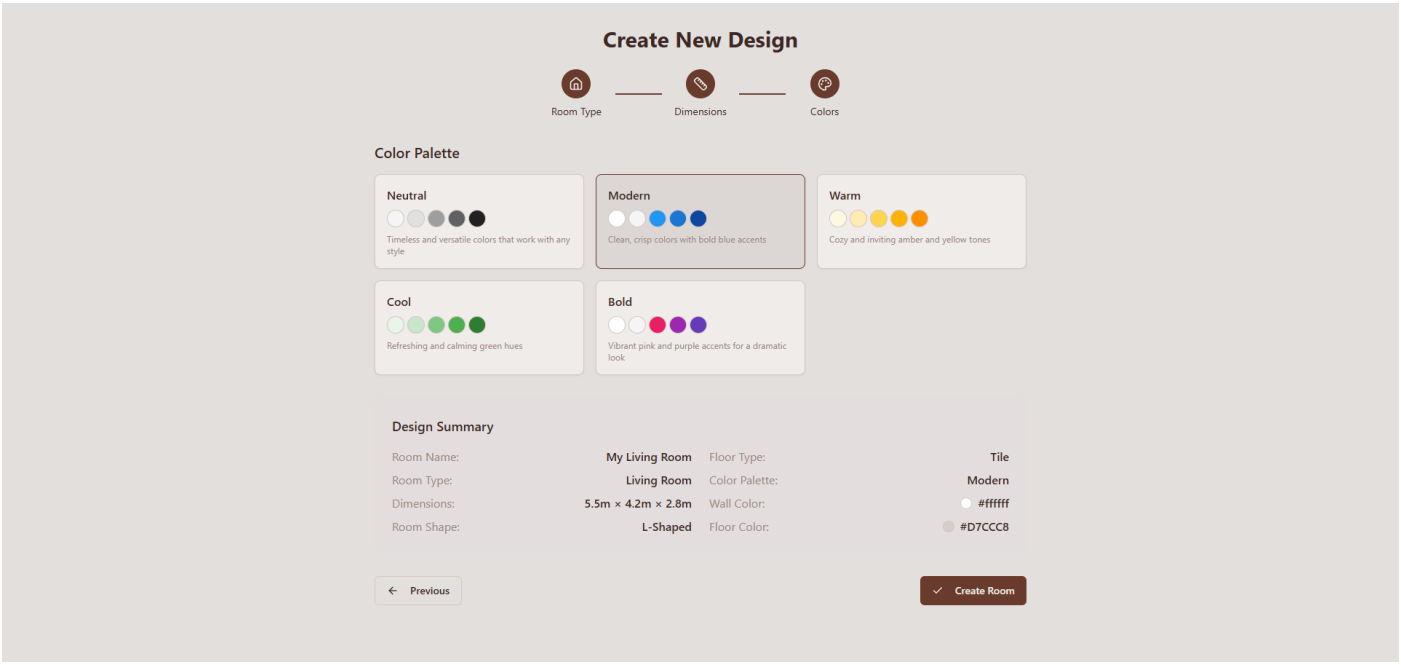

Figure 47 Customization Screen 3

9) 3D view of furniture

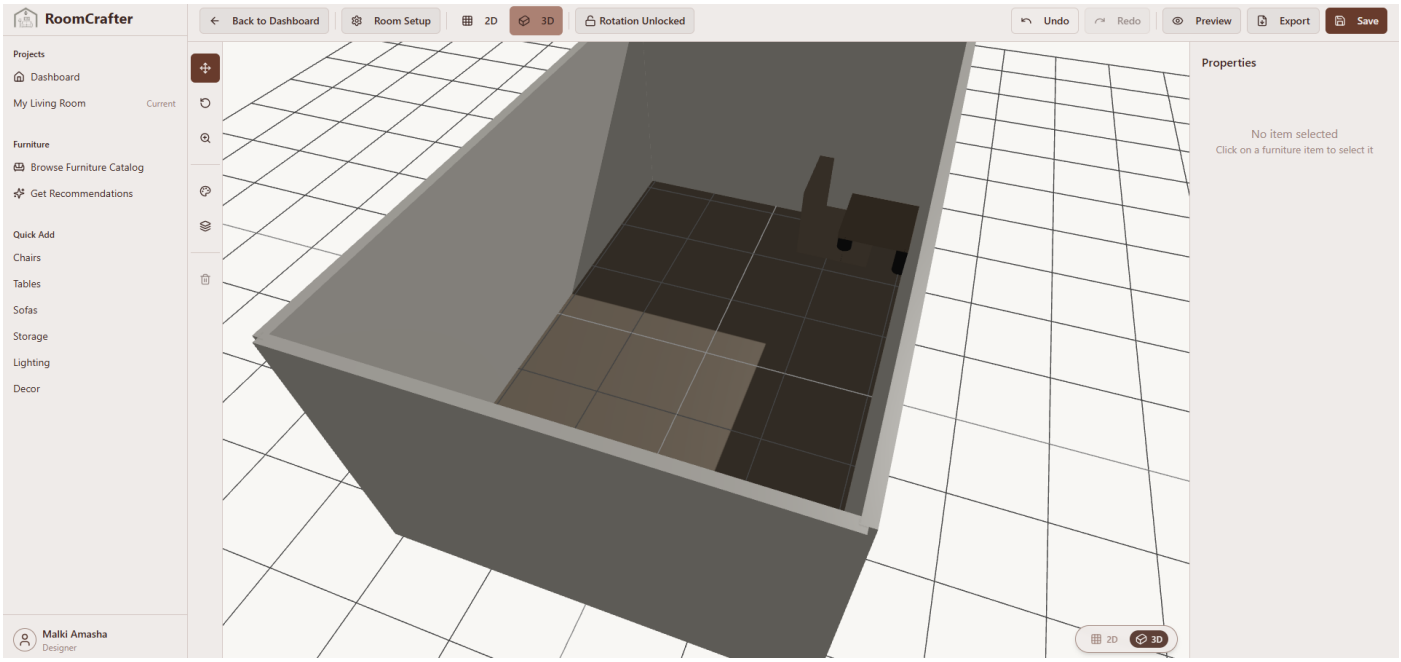

Figure 48 3D view furniture

10) 2D view of furniture

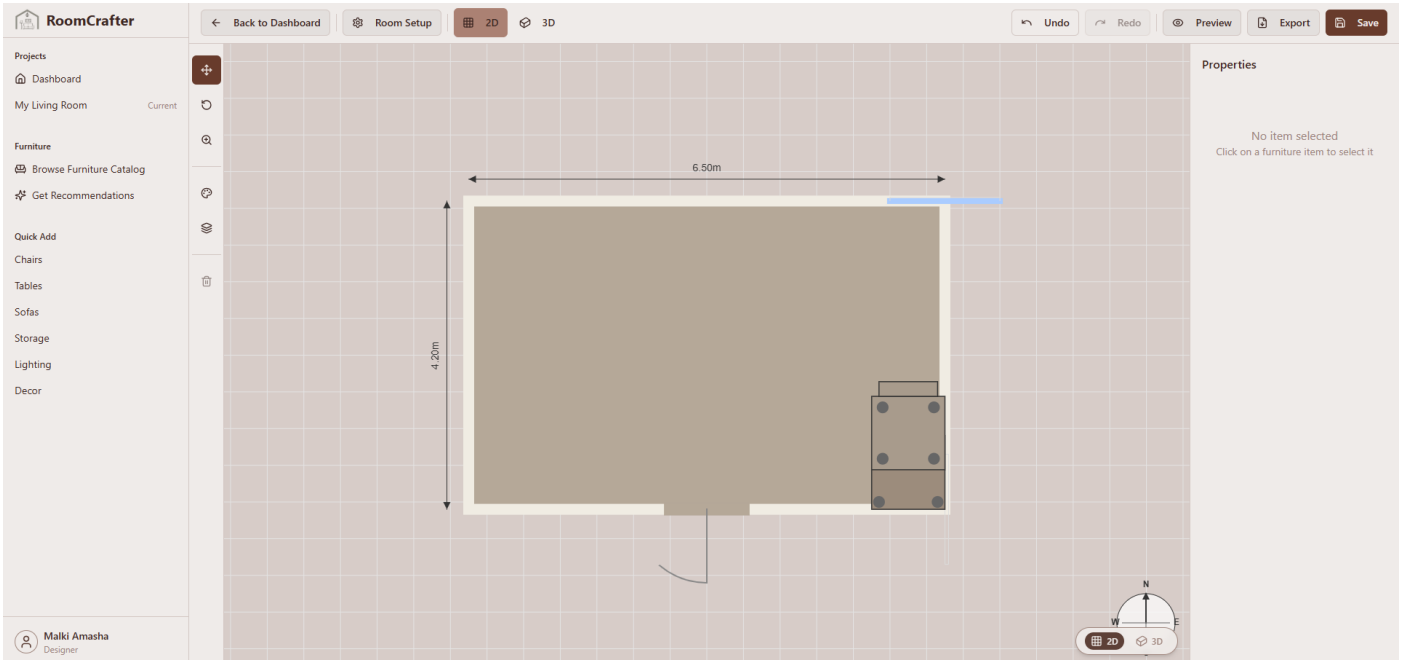

Figure 49 2D view furniture

11) Cart Page

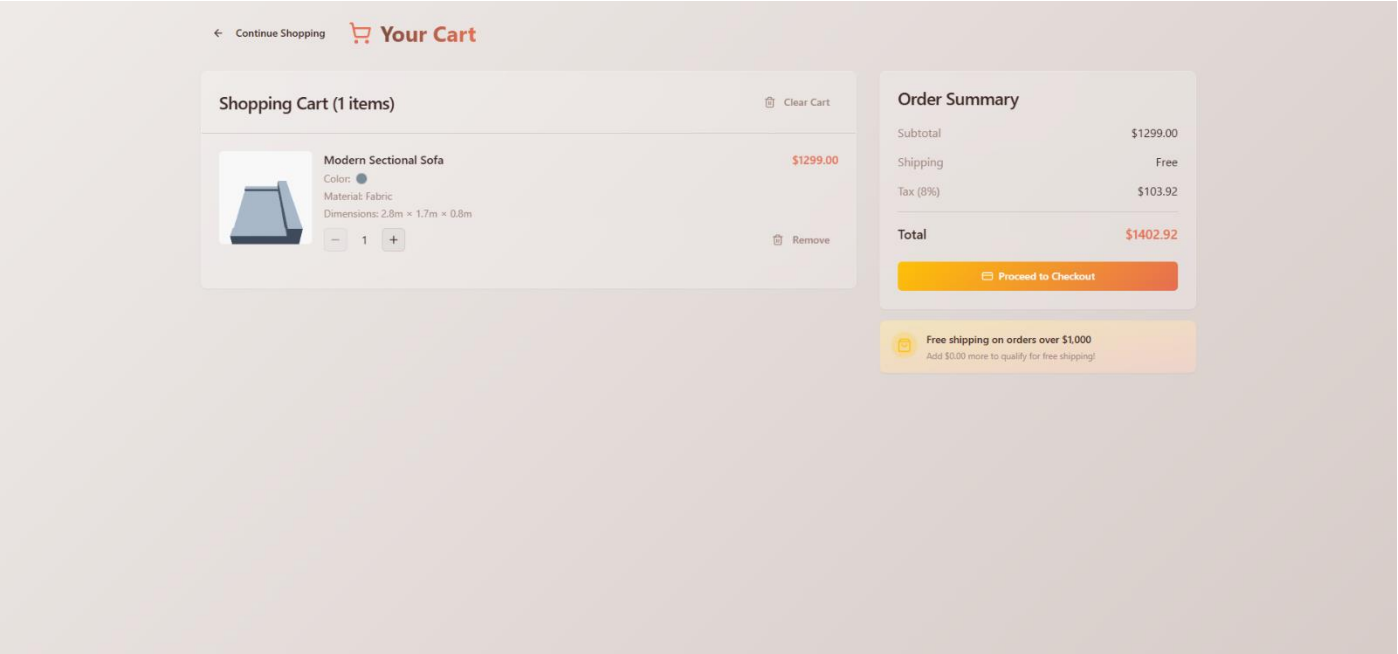

Figure 50 Cart details Screen

12) Checkout Details

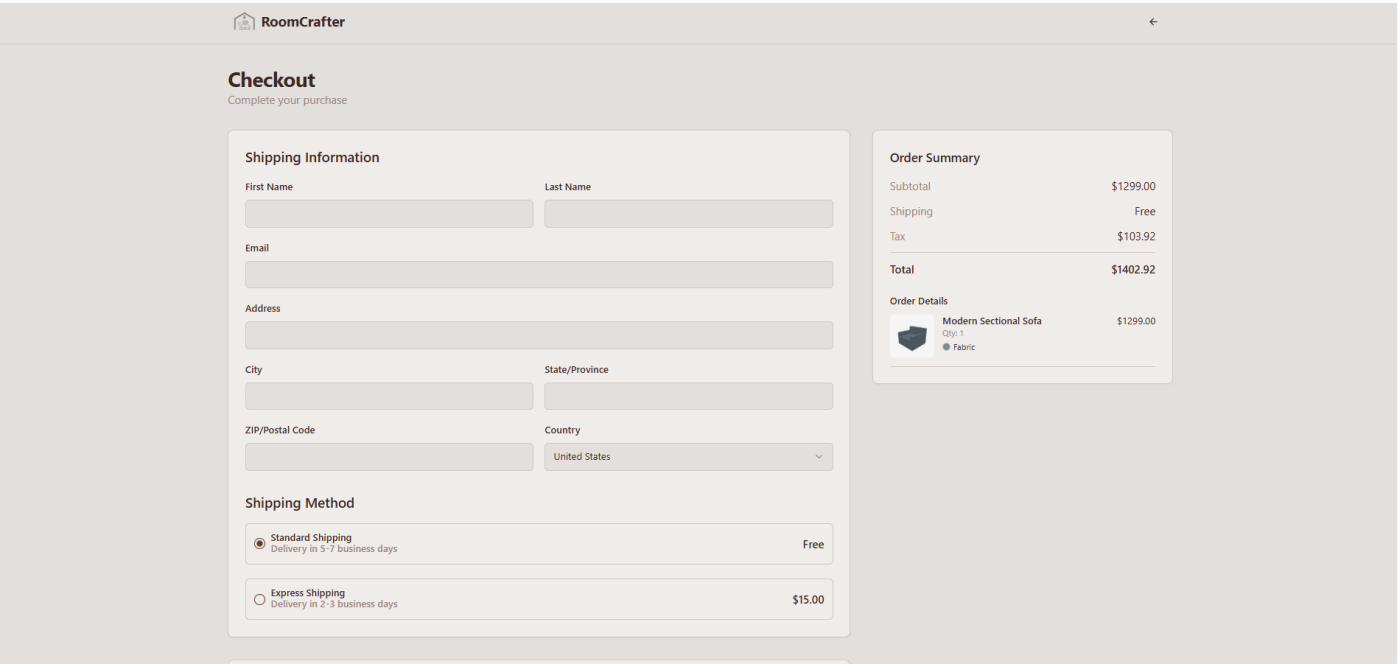

Figure 51 Checkout Screen

Usability Testing

Test Plan

Evaluation Type: Usability Testing

Number of Participants: 5

Environment: Remote (3 users), In-person lab-based (2 users)

Tasks Covered:

1. Register and log in
2. Create a new room design (set size and layout)
3. Add and customize furniture items
4. Switch between 2D and 3D views
5. Save the room design
6. Add an item to the cart

Success Guidelines:

1. Finish at least 5 out of 6 tasks without help from outside sources
2. Get an overall user rating of more than 4.0 out of 5 for usability
3. Avoid any major failures in completing tasks

Methods & Techniques Used

Task-Based Check:
Observers watched users' complete preset tasks in the Website. They tracked the time it took, errors made, and any challenges users faced.

Think-Aloud Strategy:
Testers shared their thoughts after using the Website. This approach revealed areas of confusion and user expectations.

5-point Likert scales measuring:
After testing, user reviews using 5-point Likert scales measured how satisfied users were. Then short interviews lasting about 5–10 minutes gathered more detailed feedback from them.

Participant Demographics

Participant	Age	Gender	Background
P1	22	Male	Interior design student
P2	44	Female	Furniture showroom assistant
P3	25	Female	E-commerce user
P4	36	Male	Freelance 3D visual designer
P5	23	Female	UX design intern

Data Collected

Participant	Avg. Task Time	Errors Noted	Tasks Completed (out of 6)	Usability Rating (1–5)
P1	5 min 12 sec	1 minor	6/6	5.0
P2	6 min 45 sec	2 (missed 3D toggle initially)	6/6	4.5
P3	4 min 58 sec	0	6/6	4.8
P4	5 min 30 sec	1 (saved twice unintentionally)	6/6	4.6
P5	6 min 10 sec	3 (confusion on color picker)	5/6	4.2

Data Conclusion

Average Usability Score: 4.62 / 5
Average Completion Time: ~5 min 43 sec
Completion Rate: 29/30 tasks completed (96.7%)

Key Findings

What Worked Well: People liked how the platform looked. They appreciated the different furniture option features and the 3D view option.

Needs Fixing: Some couldn’t find the 3D toggle. Others wanted clearer prompts after saving designs.

Steps Taken: Developers made important buttons easier to spot, and included save confirmation messages.

Testing Prototypes

- Test 1

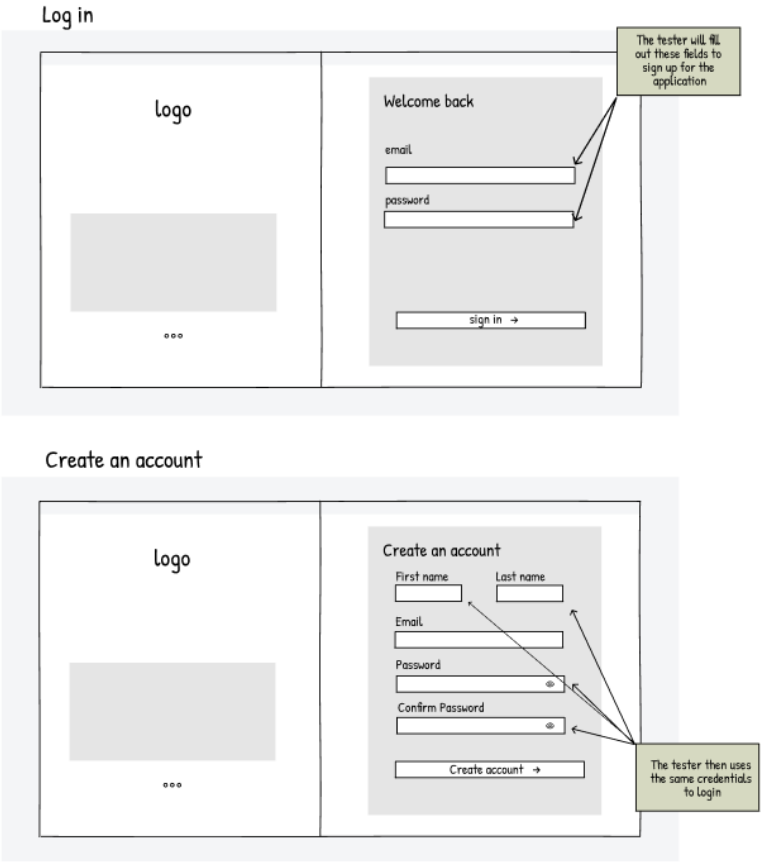

Figure 52 Test 1 - low fidelity

- Test 2

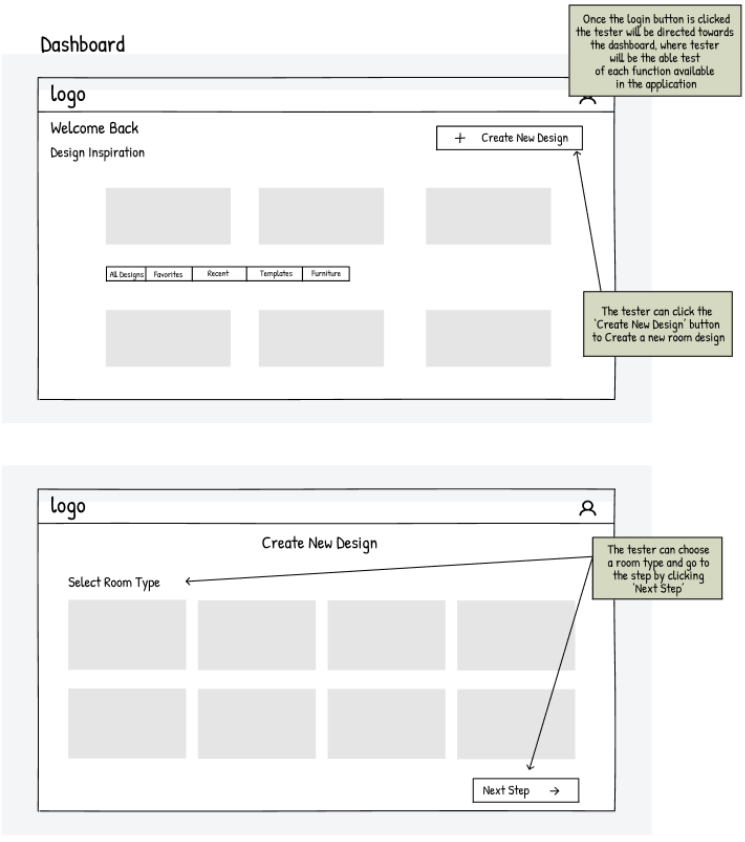

Figure 53 Test 2.1 - low fidelity

- Test 3

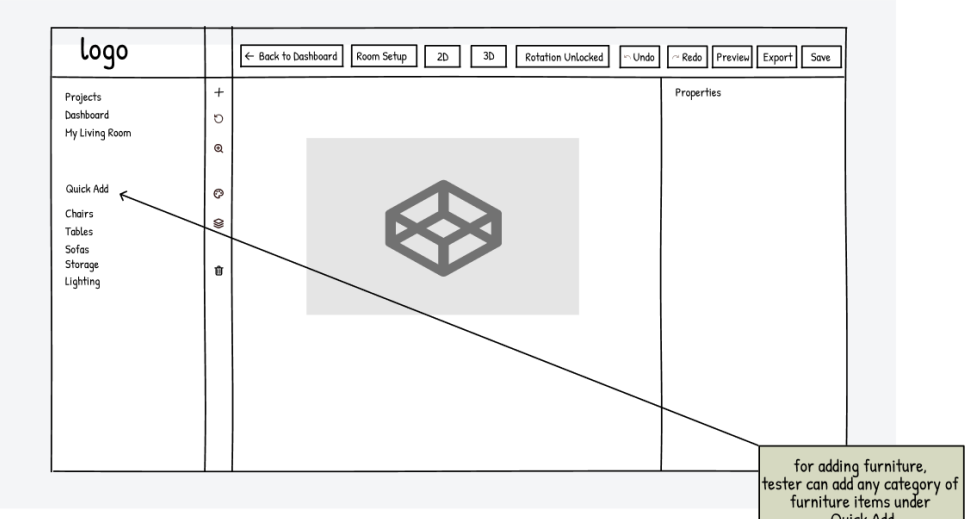

Figure 54 Test 3 - low fidelity

- Test 4

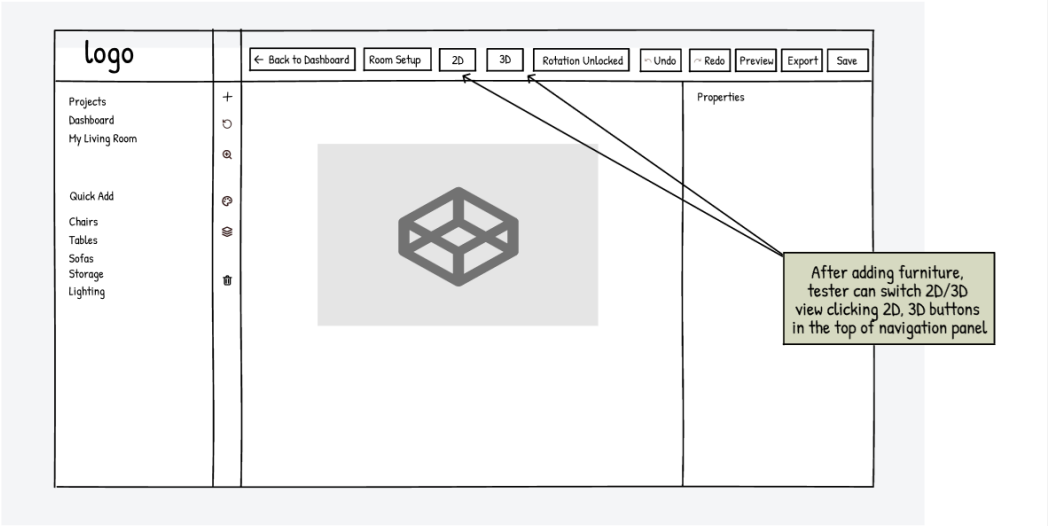

Figure 55 Test 4 - low fidelity

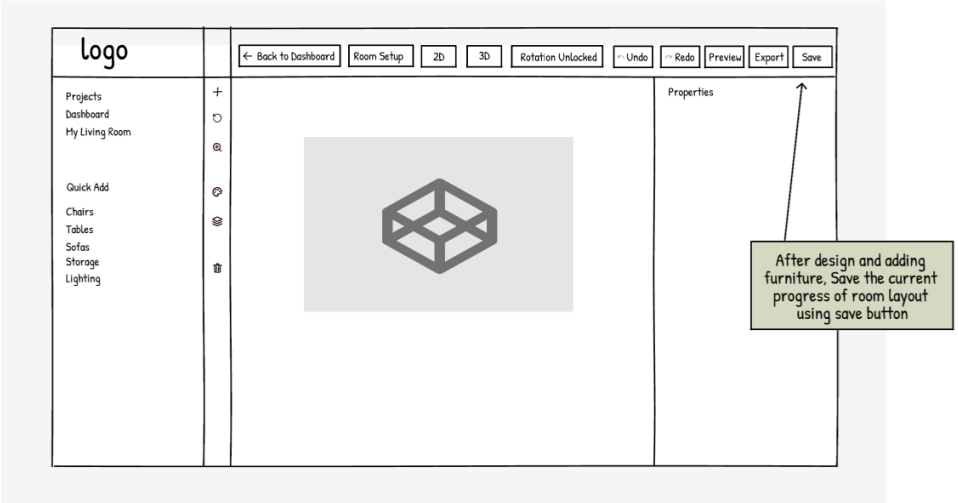

Figure 56 Test 5 - low fidelity

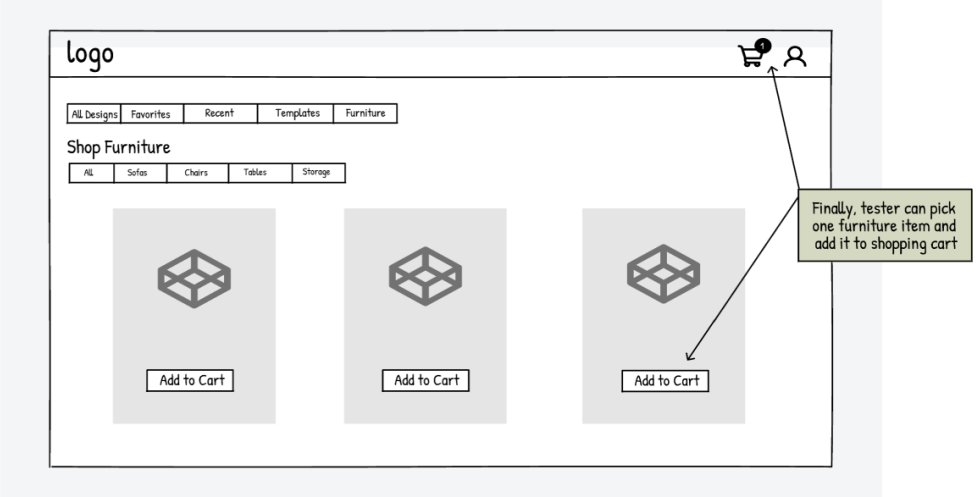

Figure 57 Test 6 - low fidelity

3. Limitations

Objective	% completion	Comments
Customer can provide the size, shape and colour scheme for the room.	100%	Implemented via room-store.ts, offering options for dimensions, shape (rectangular, L-shaped), wall colors, and floor materials.
Customer can create a new design based on the room size, shape and colour scheme.	100%	Implemented through the new-design workflow with room-setup.tsx component providing all necessary configuration options.
Customer can visualise the design in 2D.	100%	Fully implemented with floor-plan-canvas.tsx using HTML5 Canvas for interactive 2D visualization with pan, zoom, and selection capabilities.
Customer can visualise the design in 3D.	90%	Implemented with three-d-visualization.tsx using Three.js but has some performance limitations with complex scenes and lighting effects.
Customer can scale the design to best fit the room.	80%	Basic scaling functionality works, but automatic arrangement optimization for best fit is limited. Manual adjustment is still required for optimal placement.
Customer can add shade to the design as a whole or selected parts.	70%	Basic lighting and shadow effects are implemented, but fine-grained control over specific shading aspects is limited. The glossiness parameter provides some control.
Customer can change the colour of the design as a whole or selected parts.	85%	Color customization for individual furniture items is well-implemented; however, applying uniform color schemes across multiple items still requires improvement.
Customer can edit/ delete the design.	95%	Editing and deletion functionality is robust with the history-store.ts providing undo/redo capabilities, but some edge cases in complex operations may cause inconsistencies.
Customer can save the design.	100%	Fully implemented through save-design-dialog.tsx with local storage persistence and URL state management.

References

AI based interior Designing app (2024) *International Journal of Innovative Research in Engineering*. journal-article. Fifth Dimension Research Publication, pp. 30–33. https://www.theijire.com/archiver/archives/ai_based_interior_designing_app.pdf.

Computational technology and artificial intelligence (AI) revolutionizing interior design graphics and modelling (2022). https://ieeexplore.ieee.org/abstract/document/9984232.

Furniture Sri Lanka | *Furniture Shop & Home Decor* | *Finez* (no date). https://finez.lk/.

Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. (no date). https://tailwindcss.com/.

Appendix

Figma Link of all prototypes:

https://www.figma.com/design/pPP0mpemku7xJsGNt1kGXm/HCI?node-id=0-1&t=qLfZWypavoug0Z9Z-1

Survey Link:

https://forms.gle/HupbcaCpYFjVjMW79