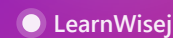


Reusable Navigation & Permissions

The four views you'll build, one shell that owns navigation, and a first taste of permissions.



Lesson goal

See the target you'll build in the lab: a reusable shell whose content area swaps between four views — Dashboard, Tickets, Customers and Settings — while the header, navigation and status bar stay the same.

1. The four views

The outer shell never changes. Each nav button loads a different **UserControl** into the content panel.

View	What it shows
DashboardView	Overview cards (Open Tickets, Customers, System Status) and a quick action such as <i>Run Health Check</i> .
TicketsView	A support-ticket list with <i>New Ticket</i> and <i>Close Selected</i> actions.
CustomersView	A customer list with <i>Add Customer</i> and <i>View Selected</i> actions.
SettingsView	The current role plus a <i>Save Settings</i> button that introduces permission behavior.



Why UserControls

Putting each page in its own UserControl keeps repeated UI organized and reusable, and lets the shell stay small — it only owns the navigation and the content panel.

2. One shell owns navigation

The exact names can change, but the idea stays the same: the shell creates the right view, docks it to fill the content panel, and updates the breadcrumb and status.

```
// Main shell owns the navigation buttons and the content panel.
public partial class MainPage : Page
{
    private string currentRole = "Support Agent";

    public MainPage()
    {
        InitializeComponent();
        NavigateTo("Dashboard");
        ApplyPermissions();
    }

    private void NavigateTo(string pageName)
    {
        pnlContent.Controls.Clear();

        UserControl view;
        switch (pageName)
        {
            case "Tickets":    view = new TicketsView();    break;
            case "Customers": view = new CustomersView(); break;
            case "Settings":   view = new SettingsView(currentRole); break;
            default:           view = new DashboardView(); break;
        }

        view.Dock = DockStyle.Fill;
        pnlContent.Controls.Add(view);
        lblBreadcrumb.Text = "Home / " + pageName;
        lblStatus.Text = DateTime.Now.ToString("hh:mm:ss tt") + " - Opened " + pageName + "
page.";
    }

    private void ApplyPermissions()
    {
        // Beginner version: a Support Agent can view Settings but not save.
        // Later this grows into real authentication and authorization.
    }
}
```

3. View-only Settings — the first permission

The Settings view is where permissions become visible. For a **Support Agent**, show the settings but disable saving — the *Save Settings* button is greyed out with a note such as *"Settings are view-only for this role."* A Manager would be allowed to save. This introduces permissions **before** any login system exists.

4. Designer checklist

- Use `Dock` or `Anchor` on the header, left nav, content panel and status bar.
- Remove the minimize button on a Window/Form1 unless you build a taskbar/restore area.
- Prefer a **Page** when the shell should fill the browser like a normal web app.
- Keep navigation logic centralized in the shell instead of duplicating it in every page.

- Use `UserControls` for repeated content — `DashboardView` , `TicketsView` , `CustomersView` , `SettingsView` .



You're ready for the lab

Next: watch the navigation-shell walkthrough, pass the knowledge check, then build the reusable navigation shell — header, four nav buttons, a swapping content area, a status log, and view-only Settings for a Support Agent.