I have this code that lights up a i2c oled display using a xiao esp32s3. I want to create a code with a web server with buttons with numbers 0-9 and when each button is pressed, the corresponding number appears on the oled display.

ChatGPT said:

Here's a complete example of how to set up a **web server** with **buttons labeled 0–9** on a **XIAO ESP32S3**, and have each button press display the corresponding number on an **I2C OLED display** (e.g., SSD1306 128x64) using **ESPAsyncWebServer** and **Adafruit SSD1306** library.

🔧 Requirements

Install these libraries in the Arduino IDE:

- `ESPAsyncWebServer` (via library manager or GitHub)
- `AsyncTCP`
- `Adafruit SSD1306`
- `Adafruit GFX`

Ensure you enable I2C on the right pins for the XIAO ESP32S3 (usually `SDA: GPIO 6`, `SCL: GPIO 7`).

✅ Arduino Code

cpp
CopyEdit

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>

// Replace with your network credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
```

```
// OLED display size
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// I2C OLED setup (default address is 0x3C)
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// Web server running on port 80
AsyncWebServer server(80);

// HTML page with number buttons
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>OLED Number Display</title>
  <style>
    button {
      font-size: 2em;
      margin: 10px;
      padding: 15px;
    }
  </style>
</head>
<body>
  <h1>Click a Number</h1>
  %BUTTONS%
</body>
</html>
)rawliteral";

String createButtons() {
  String html = "";
  for (int i = 0; i <= 9; i++) {
```

```
    html += "<button onclick=\"fetch('/display?num=" + String(i) +
"')\">" + String(i) + "</button>";
  }
  return html;
}

String processor(const String& var) {
  if (var == "BUTTONS") {
    return createButtons();
  }
  return String();
}

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected.");
  Serial.println(WiFi.localIP());

  // Initialize OLED
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 allocation failed");
    while (true);
  }
  display.clearDisplay();
  display.setTextSize(4);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Ready");
  display.display();

  // Web server routes
```

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });

  server.on("/display", HTTP_GET, [](AsyncWebServerRequest *request){
    if (request->hasParam("num")) {
      String num = request->getParam("num")->value();

      // Display number on OLED
      display.clearDisplay();
      display.setCursor(0, 0);
      display.print(num);
      display.display();
      Serial.println("Displayed: " + num);
    }
    request->send(200, "text/plain", "OK");
  });

  server.begin();
}

void loop() {
  // Nothing to do here
}
```

🧪 Testing

1. Upload the code.
2. Open Serial Monitor to get the IP address.
3. Open the IP address in a browser.
4. Click a number – it will appear on the OLED!

Would you like to add:

- A **"Clear"** button?
- **Multiple-digit input** capability?
- A **custom font or style**?

Let me know how you'd like to expand it!

You said:

/private/var/folders/sz/tvwsnn653mj2j1mq8wb59_xm0000gn/T/.arduinoIDE-unsaved2025428-64602-ergx5u.buq4k/sketch_may28b/sketch_may28b.ino:2:10: fatal error: ESPAsyncWebServer.h: No such file or directory 2 | #include <ESPAsyncWebServer.h> | ^~~~~~~~~~~~~~~~~~~~~ compilation terminated. exit status 1 Compilation error: ESPAsyncWebServer.h: No such file or directory

ChatGPT said:

That error means the **`ESPAsyncWebServer` library** isn't installed in your Arduino IDE yet. Follow these steps to fix it:

✅ Step-by-Step Installation (for ESPAsyncWebServer)

1. Install `ESPAsyncWebServer`

This library isn't in the Library Manager by default, so you need to install it manually.

- Go to: ESPAsyncWebServer GitHub Repo
- Click **"Code" > "Download ZIP"**
- In Arduino IDE:
    - Go to **Sketch > Include Library > Add .ZIP Library**
    - Select the downloaded ZIP file.

2. Install `AsyncTCP`

You must also install this dependency:

- Go to: AsyncTCP GitHub Repo

- Repeat the same process: **Download ZIP** > **Add .ZIP Library** via Arduino IDE.

> **Important**: Use the correct versions from **me-no-dev**, not forks, to ensure compatibility.

✅ After Installation

Try compiling your code again. If successful, it should go past the missing library error.

Let me know if you'd prefer using **WebServer.h** (built-in, simpler) instead of ESPAsyncWebServer to avoid these manual installs. I can rewrite the code accordingly.

You said:
i've never needed that library before when making a web server

ChatGPT said:

You're absolutely right — you **don't need** `ESPAsyncWebServer` unless you want **asynchronous/non-blocking handling** (useful for complex tasks or WebSockets). For **simple web servers**, like serving buttons and handling clicks, the built-in `WebServer` library (part of the ESP32 core) works perfectly and is much easier to set up.

Here's a revised version of your project using just **`WiFi.h`** and **`WebServer.h`** — no extra libraries needed for the server!

✅ Simpler Code Using `WebServer.h`

cpp
CopyEdit

```
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Replace with your credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
```

```
// OLED display setup
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// Web server on port 80
WebServer server(80);

String createHTML() {
  String html = "<!DOCTYPE html><html><head><title>OLED
Number</title><meta name='viewport' content='width=device-width,
initial-scale=1'><style>button{font-size:2em;margin:10px;}</style></he
ad><body><h2>Click a number:</h2>";
  for (int i = 0; i <= 9; i++) {
    html += "<a href='/display?num=" + String(i) + "'><button>" +
String(i) + "</button></a>";
  }
  html += "</body></html>";
  return html;
}

void handleRoot() {
  server.send(200, "text/html", createHTML());
}

void handleDisplay() {
  if (server.hasArg("num")) {
    String num = server.arg("num");
    Serial.println("Displaying: " + num);

    display.clearDisplay();
    display.setTextSize(4);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print(num);
```

```
    display.display();

    server.send(200, "text/html", "<h1>Displayed: " + num + "</h1><a
href='/'>Back</a>");
  } else {
    server.send(400, "text/plain", "Missing number!");
  }
}

void setup() {
  Serial.begin(115200);

  // OLED Init
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    while (true);
  }
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Ready");
  display.display();

  // Wi-Fi setup
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected! IP address: ");
  Serial.println(WiFi.localIP());

  // Server routes
  server.on("/", handleRoot);
  server.on("/display", handleDisplay);
  server.begin();
```

```
}

void loop() {
  server.handleClient();
}
```

🧪 How to Use

1. Replace `"YOUR_SSID"` and `"YOUR_PASSWORD"` with your Wi-Fi credentials.
2. Upload the code.
3. Open Serial Monitor to find the IP address.
4. Open that IP in a browser — click a number, and it appears on the OLED!

Would you like:

- a **"Clear display"** button?
- to **display multiple digits** as you type?
- a **numpad layout** (like a phone)?

Let me know!

You said:
combine with this code that controls the screen: #include <Wire.h> #include <Adafruit_GFX.h> #include <Adafruit_SSD1306.h> // Display dimensions #define SCREEN_WIDTH 128 #define SCREEN_HEIGHT 64 // OLED reset pin (not used with I2C) #define OLED_RESET -1 #define SCREEN_ADDRESS 0x3C // I2C pins for XIAO ESP32S3 #define SDA_PIN 8 // D10 #define SCL_PIN 9 // D9 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET); void setup() { Serial.begin(115200); delay(500); // Wait for Serial Monitor Wire.begin(SDA_PIN, SCL_PIN); // Initialize OLED display if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) { Serial.println(F("OLED init failed")); while (true); // Stop execution if OLED not found } display.clearDisplay(); display.setTextSize(2); display.setTextColor(SSD1306_WHITE); display.setCursor(0, 0); display.println("Hello!"); display.display(); } void loop() { // Do nothing }

ChatGPT said:

Here's your **combined code**:

- It **uses your custom I2C pins** (`SDA = 8`, `SCL = 9`) for the XIAO ESP32S3.
- It sets up a **Wi-Fi web server using `WebServer.h`**.
- The server displays **buttons 0–9**, and pressing one shows the number on the OLED display.

✅ Complete Combined Code

cpp
CopyEdit

```
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Wi-Fi credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// OLED setup
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SCREEN_ADDRESS 0x3C

// I2C pins for XIAO ESP32S3
#define SDA_PIN 8  // D10
#define SCL_PIN 9  // D9

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// Web server on port 80
```

```
WebServer server(80);

String createHTML() {
  String html = "<!DOCTYPE html><html><head><title>OLED
Numbers</title><meta name='viewport' content='width=device-width,
initial-scale=1'><style>button{font-size:2em;margin:10px;}</style></he
ad><body><h2>Click a number:</h2>";
  for (int i = 0; i <= 9; i++) {
    html += "<a href='/display?num=" + String(i) + "'><button>" +
String(i) + "</button></a>";
  }
  html += "</body></html>";
  return html;
}

void handleRoot() {
  server.send(200, "text/html", createHTML());
}

void handleDisplay() {
  if (server.hasArg("num")) {
    String num = server.arg("num");
    Serial.println("Displaying: " + num);

    display.clearDisplay();
    display.setTextSize(4);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print(num);
    display.display();

    server.send(200, "text/html", "<h1>Displayed: " + num + "</h1><a
href='/'>Back</a>");
  } else {
    server.send(400, "text/plain", "Missing number!");
  }
}
```

```
void setup() {
  Serial.begin(115200);
  delay(500); // Wait for Serial Monitor

  // Start I2C on custom pins
  Wire.begin(SDA_PIN, SCL_PIN);

  // Initialize OLED display
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("OLED init failed"));
    while (true); // Stop if display not found
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("Hello!");
  display.display();

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Set up routes
  server.on("/", handleRoot);
  server.on("/display", handleDisplay);
  server.begin();
}

void loop() {
```

```
  server.handleClient();
}
```

✅ What to Do

1. Replace `"YOUR_SSID"` and `"YOUR_PASSWORD"` with your Wi-Fi credentials.
2. Upload the code to your XIAO ESP32S3.
3. Open the **Serial Monitor** to get the IP address.
4. Go to that IP address in a browser.
5. Click any number — it'll show up on the OLED display!

Let me know if you want:

- a **"Clear"** or **"Backspace"** button
- the ability to **type multiple digits**
- **numpad-style layout**
  Happy to help you extend it!

You said:

I'm confused about how this: design, build and connect wired or wireless node(s) with network or bus addresses and a local input and/or output devices is different from this: Write an application for the embedded board that you made. that interfaces a user with an input and/or output device(s)

ChatGPT said:

Great question — those two Fab Academy assignments **sound similar**, but they have **different goals and scopes**. Here's a breakdown to help clarify the difference:

🧩 1. "Design, build and connect wired or wireless node(s) with network or bus addresses and a local input and/or output device(s)"

→ **This is about networking communication between devices.**

🔍 **What it's really about:**

- **Multiple devices talking to each other**.
- Each device (node) has its own **address**.
- They are connected via **I2C, UART, SPI, CAN, or Wi-Fi/Bluetooth**, etc.
- One device (like a master or server) can **send or receive data** from another (slave/client).
- Each node has **at least one input or output** device (e.g., a sensor or LED).

✅ **Example:**

- You have **two microcontrollers**:
  - **Node A**: Reads temperature and sends it via I2C.
  - **Node B**: Displays that temperature on an OLED.
- Or:
  - Several ESP32s talk over Wi-Fi, each one controlling a motor or sensor.
- The key: there is **communication between nodes with addressing**.

👤 2. "Write an application for the embedded board that you made. that interfaces a user with an input and/or output device(s)"

→ **This is about user interaction with a single device.**

🔍 **What it's really about:**

- A **standalone application** running on **your own custom board**.
- Involves **user input/output**:

    - Input: button, touch sensor, potentiometer
    - Output: LED, OLED screen, speaker
- You're writing code that creates an **interface between the user and the hardware**.

✅ **Example:**

- Your board has a **touch button** and an **LED**.
    - When the user touches it, the LED blinks a pattern.
- Your board has a **rotary encoder**.
    - It changes a number shown on an **OLED screen**.
- Your final project may satisfy this if it includes interaction.

🧠 Summary:

Assignment	Focus	Devices	Interaction
Networking	**Communication between devices**	2+ boards (nodes)	Devices talk to each other
Interface Application	**User interaction with 1 board**	Your custom board	User talks to device

---

🛠️ TL;DR with Example

Let's say you made a temperature monitoring system:

- **Networking Week**: One ESP32 reads temperature, sends data to another ESP32, which displays it. (Communication!)
- **Interface Week**: A single board displays temperature on OLED and lets a user change units with a touch sensor. (User interface!)

Let me know what boards/devices you're using and I can give you a project idea that clearly separates the two weeks.

You said:

can i use a web interface for both weeks

ChatGPT said:

Yes, you **absolutely can use a web interface for both weeks**, but the way you **use it must align with the goal of each assignment**:

✅ Interface Week (Application/UI)

You can use a **web interface** to:

- Let a user **press buttons** (e.g., 0–9) from a browser
- Show sensor readings on the screen
- Change settings via sliders/switches

> ✅ **Allowed**: A **web interface on a single board** that shows output (OLED, LED, buzzer) or takes input (touch sensor, buttons)
> 🔗 It's about creating an app that **interacts with a user**

✅ Networking Week (Multiple Nodes)

You can still use a **web interface**, but only if:

- There are **multiple nodes** (microcontrollers) communicating
- The nodes are assigned **addresses** (like I2C or over IP/Wi-Fi)
- Data is sent **between the nodes**, not just between browser and board

> ✅ **Allowed**:
>
> - Web interface on **Node A** sends a command

- **Node A sends data to Node B**, which reacts (e.g., lights an LED or shows it on an OLED)
- Or one ESP32 hosts the web UI, and sends commands to another ESP32 over I2C/SPI/UART/Wi-Fi

❌ **Not Allowed** (for networking week):

- A single ESP32 with a web interface — that's an interface application, **not networking**

✅ Fab Academy-Proof Combo Example

Web UI + Multiple ESP32s over Wi-Fi (Networking)

- Web UI on Node A (ESP32 #1): lets the user click "Turn on LED"
- Node A sends a **message over Wi-Fi** (e.g., using HTTP, UDP, or MQTT) to Node B (ESP32 #2)
- Node B receives the message and turns on an LED
- Maybe Node B replies with status (LED on), and Node A updates the UI

🧠 Summary Table

Week	Use Web Interface?	How to use it correctly
Interface	✅ Yes	UI talks directly to **one board**
Networking	✅ Yes	UI on one node, **that node talks to another node**

If you tell me:

- How many boards you're using

- What devices you want to control
- How you'd like the web UI to work

…I can give you **clear Fab-approved** example setups for both weeks.

Chat 2:

Please provide more detail about your UI. Give more details about its purpose,functionality and etc.

ChatGPT said:

Sure! Here's a detailed explanation of a **User Interface (UI)** — its purpose, functionality, and key components — written generically, and then I can tailor it to your specific context (e.g. a web app, a physical device, or an embedded project) if you let me know more about what you're working on.

User Interface (UI): Purpose and Functionality

🧩 Purpose of the UI

The **User Interface** serves as the **bridge between the user and the system**. It enables users to interact with software, hardware, or embedded systems efficiently and intuitively.

Key goals:

- **Ease of use:** Allow users to perform tasks with minimal effort or confusion.
- **Clarity:** Present options and feedback clearly.
- **Accessibility:** Ensure usability for people with different abilities.
- **Feedback loop:** Provide real-time or context-aware feedback based on user actions.

⚙️ Functionality of the UI

The UI typically supports the following functional areas:

1. **Input Handling**

    - Accepts user inputs via buttons, touch, keyboard, mouse, or sensors.
    - Processes interactions like clicks, taps, gestures, or text input.
2. **Output Display**
    - Shows relevant information such as status, feedback, or data.
    - Can use screens, LEDs, audio cues, or haptic feedback.
3. **Navigation and Control**
    - Allows movement between screens, menus, or modes.
    - Enables control over functions (e.g., starting a task, changing settings, toggling outputs).
4. **Status Indicators**
    - Displays system status (e.g., on/off, error states, completion).
    - Uses icons, LED colors, animations, or text.
5. **User Feedback**
    - Confirms actions (e.g., "Task saved" notification).
    - Warns about errors or prompts for next steps.

📐 UI Components (Digital or Physical)

Depending on the system, the UI may include:

- **Graphical UI (GUI)**: buttons, sliders, icons, progress bars, menus.
- **Command-line UI**: text prompts, keyboard-based interaction.
- **Touchscreen Interface**: gesture-based controls, virtual keyboards.
- **Embedded/Physical UI**:

- **LEDs**: Indicate status or guide user actions.
- **Buttons/Switches**: Input commands.
- **LCD/OLED Displays**: Show messages, menus, or instructions.
- **Web-based UI**: Interfaces accessed through a browser, with interactive components such as dropdowns, toggles, and real-time data display.

🔍 Example (Embedded Project UI)

If you're working on something like an **ESP32-based game** or **interactive LED display**:

- The UI might include:
  - A **web interface** to select patterns, start/stop the game, and monitor scores.
  - **Capacitive touch sensors** as input (e.g., Whack-a-Mole hits).
  - An **LCD display** to show game status ("Ready", "Game Over", score).
  - **LEDs** or a matrix to guide the player visually.

Would you like me to describe the UI for your **ESP32-based project**, **Fab Academy final**, or something else? Just let me know!

You said:
ui is a web server and keep it brief in paragraph form

ChatGPT said:

The UI is a web-based interface hosted on an ESP32-S3 web server. Its purpose is to allow users to interact with the system remotely through a browser. It provides controls to trigger functions such as starting or stopping operations, selecting LED patterns, or viewing system status. The interface includes buttons, dropdown menus, and real-time feedback elements, enabling intuitive interaction without needing a physical display or controls. This setup makes it easy to monitor and control the device wirelessly from any device connected to the same network.

You said:

add on to this: For output week, one of my outputs was a LCD1602 screen, and for my final project, I thought it could be useful to connect this to a web server. I used a Xiao ESP32-S3 which has web server capabilities to make the web server and ChatGPT helped me make a code to run the web server. I modified the code from Output Week and changed it to incorporate a web server. It uses I2C to communicate. I did not come across any problems when making this interface.

ChatGPT said:

For output week, one of my outputs was a LCD1602 screen, and for my final project, I thought it could be useful to connect this to a web server. I used a Xiao ESP32-S3 which has web server capabilities to make the web server and ChatGPT helped me make a code to run the web server. I modified the code from Output Week and changed it to incorporate a web server. It uses I2C to communicate. I did not come across any problems when making this interface. The web interface allows me to send messages from a browser directly to the LCD screen in real time. This makes the display dynamic and easy to update without needing to reprogram the board, improving the usability of my final project by enabling wireless control and feedback through a simple, accessible UI.

You said:

Please provide more detail about your UI. Give more details about its purpose,functionality and etc.

ChatGPT said:

My user interface is a web-based UI hosted on the Xiao ESP32-S3's built-in web server. Its main purpose is to allow wireless interaction with my final project through a browser on any device connected to the same Wi-Fi network. The UI lets me input and send custom text messages that are displayed on the LCD1602 screen connected via I2C. It includes a simple text input field and a submit button, making it easy to update the display in real time without needing to reprogram the microcontroller. This adds flexibility and convenience, especially for testing or demonstrations. The web server handles incoming HTTP requests and updates the LCD accordingly, making the interface a lightweight yet powerful tool for remote control and feedback.

You said:

remove this: it includes a simple text input field and a submit button, making it easy to update the display in real time without needing to reprogram the microcontroller.

ChatGPT said:

My user interface is a web-based UI hosted on the Xiao ESP32-S3's built-in web server. Its main purpose is to allow wireless interaction with my final project through a browser on any device connected to the same Wi-Fi network. The UI lets me input and send custom text messages that are displayed on the LCD1602 screen connected via I2C. The web server

handles incoming HTTP requests and updates the LCD accordingly, making the interface a lightweight yet powerful tool for remote control and feedback.

You said:

make it fit the feedback more

ChatGPT said:

My user interface is a web-based UI hosted on the Xiao ESP32-S3's built-in web server. Its purpose is to enable wireless communication between the user and the LCD1602 screen used in my final project. By connecting through a browser, I can send custom messages to the screen without needing a physical connection. The UI communicates with the LCD via I2C and updates the display based on HTTP requests handled by the ESP32. This adds a layer of interactivity to the project, allowing for dynamic, real-time output and easier testing or demonstration.

You said:

add more. this is for fab academt documenting

ChatGPT said:

My user interface is a web-based UI hosted on the Xiao ESP32-S3's built-in web server. Its purpose is to enable wireless communication between the user and the LCD1602 screen used in my final project. By connecting through a browser, I can send custom messages to the screen without needing a physical connection, which makes the system more user-friendly and accessible. The UI communicates with the LCD via I2C, and the ESP32 handles incoming HTTP requests to update the screen in real time. This interface allows for quick iteration and testing, especially useful during development, presentations, or classroom use. It eliminates the need to re-upload code every time the message needs to change and introduces a flexible way to interact with the project remotely. The web UI is lightweight, loads quickly, and runs locally on the microcontroller, meaning it doesn't rely on external servers or internet access. This integration of hardware and software enhances the interactivity of my project and showcases how embedded systems can be combined with web technologies to create responsive, user-focused applications.