# A Comprehensive Development and Deployment Guide for a Custom ArcGIS Experience Builder Chatbot Widget

## Part 1: Application and Template Foundation in Experience Builder

### 1.1 Introduction to the Development Philosophy

The development environment is correctly established: ArcGIS Experience Builder (ExB) Developer Edition is installed, operational, and authenticated with an ArcGIS Online (AGOL) portal.1 This local developer environment is the fundamental prerequisite for creating and testing any custom widget.2 The initial objective is to construct the host application, or "Experience," which will contain and interact with the custom chatbot widget.

### 1.2 Strategic Template Selection

The initial step in creating a new application is "template selection".3 Experience Builder provides a gallery of templates, which are broadly categorized as map-centric 4 or blank canvas templates.7

Standard, map-centric templates such as "Foldable" 6 or "Launchpad" are pre-configured with a specific layout and a suite of interconnected widgets.9 While these are efficient for building standard, map-focused applications, they impose a rigid design structure.

The user's requirement is for a chatbot, which represents a non-traditional, "command-first" interface. The chatbot widget is not a secondary tool to be housed in a panel; it is a primary method of application interaction. This paradigm shift requires maximum layout flexibility to position the map and the chat interface as co-equal components.

For this reason, the **"Blank fullscreen"** or **"Blank scrolling"** templates are the optimal choice.7 These templates provide a blank canvas, freeing the developer from the constraints of a pre-defined layout. This approach allows the application to be designed *around* the custom widget, rather than forcing the widget into an existing container. This ensures that no layout conflicts arise from existing template logic or fixed-size containers.10

### 1.3 Creating the New Application

The following steps detail the creation of the host application within the local developer edition:

1. **Step 1:** Access the local Experience Builder environment (typically https://localhost:3001/).
2. **Step 2:** Click the "Create new" button, located in the top-right corner of the interface.3
3. **Step 3:** In the "Create new experience" template gallery, select the "Blank fullscreen" template. Click the "Create" button.7

### 1.4 Adding and Configuring the Foundational Map Widget

With the blank application created, the foundational map must be added. This map will be the target of all commands issued to the chatbot.

1. **Step 1:** On the left side of the builder, open the "Insert widget" panel.8
2. **Step 2:** Locate the **"Map"** widget from the list of core widgets.5
3. **Step 3:** Drag and drop the Map widget onto the blank canvas.6 Resize the widget to fill the desired screen area (e.g., 75% of the canvas, leaving room for the chatbot widget).
4. **Step 4:** Select the Map widget on the canvas. The widget's configuration panel will appear on the right.
5. **Step 5:** In the configuration panel, click the "Select map" button.6
6. **Step 6:** In the "Select data" panel, click "Add new data".6 This will open a window to browse for content.
7. **Step 7:** Navigate to the "ArcGIS Online" tab (or your connected portal) and select the web map that will be controlled by the application.11 This web map becomes the application's primary data source.8
8. **Step 8:** Save the application. The application is now saved as a new "Web Experience" item within your ArcGIS Online content folder.7

## Part 2: Scaffolding the Custom Chatbot Widget

### 2.1 The Widget Development Environment

Custom widget development does not occur within the Experience Builder's visual interface. Instead, it involves creating and editing files directly in the file system, using a code editor such as Visual Studio Code.2

The Experience Builder framework is architected using React and TypeScript.2 Consequently, all custom widgets are fundamentally React components designed to operate within the ExB ecosystem.

### 2.2 Creating the Widget Files

1. **Step 1:** In your local file system, navigate to the root directory of your ExB Developer Edition installation.
2. **Step 2:** Browse to the client/your-extensions/widgets directory.13 This is the designated folder where ExB scans for all custom widgets.
3. **Step 3:** Create a new folder for the widget. The folder name must not contain spaces.15 For this project, the folder will be named chatbot-widget.
4. **Step 4:** Inside the chatbot-widget folder, create the minimal required file structure.2 This includes the manifest.json file at the root, and a src folder containing runtime/widget.tsx and setting/setting.tsx.

### 2.3 The manifest.json File

The manifest.json file serves as the widget's "passport," describing its properties, requirements, and capabilities to the Experience Builder framework.17 This file must be configured correctly for the widget to load and access map data.

Key configurations for this manifest.json include:

* **"dependency": ["jimu-arcgis"]**: This is a critical entry. It instructs ExB to load the jimu-arcgis library, which provides the necessary modules to interact with the ArcGIS Maps SDK for JavaScript (JSAPI).18 Without this, the widget cannot perform geocoding, layer manipulation, or printing.
* **"useMapWidgetIds":**: This array property declares that the widget is designed to connect to one or more Map widgets.
* **"setting": "src/setting/setting.tsx"**: This property tells ExB that the widget has a configuration panel and specifies the path to the React component that defines its UI.16

### 2.4 The Runtime and Settings Files

* **src/runtime/widget.tsx**: This is the primary file for the widget. It defines the React component that users will see and interact with at runtime.15 This file will contain the chat interface, the message history display, and all the command-parsing logic.
* **src/setting/setting.tsx**: This file defines the React component for the widget's configuration panel, which is visible *only* inside the ExB builder interface.16 Its purpose is to allow the app author to connect the chatbot-widget to a specific Map widget (the one added in Part 1).

### 2.5 Starting the Server

Experience Builder's developer environment runs as two separate services.1

1. **Step 1:** Open two separate command prompt or terminal windows.
2. **Step 2:** In the first terminal, navigate to the client directory (e.g., .../arcgis-experience-builder-1.18/client) and run the command npm start. This starts the webpack server that watches for changes to your widget code.19
3. **Step 3:** In the second terminal, navigate to the server directory (e.g., .../arcgis-experience-builder-1.18/server) and run the command npm start.1

A critical and often-overlooked step: the npm start (client) service's file watcher detects *changes* to existing files, but it does **not** detect the *creation* of new widget folders.14 After creating the chatbot-widget folder and manifest.json file for the first time, the npm start process in the client terminal must be stopped (e.g., with Ctrl+C) and restarted. Failure to do so will result in the widget not appearing in the builder.

### 2.6 Adding the Widget to the App

Once the server has been restarted, the custom widget will be available in the builder:

1. **Step 1:** Refresh the browser window containing your Experience Builder app.
2. **Step 2:** Open the "Insert widget" panel from the sidebar.
3. **Step 3:** Scroll to the bottom of the widget list. A new group labeled "Custom" will be present.15
4. **Step 4:** The "chatbot-widget" will be listed. Drag this widget onto the canvas, for example, in a sidebar next to the Map widget.

### Table 1: Custom Widget File Structure

This table provides a clear architectural reference, mapping the key files to their specific roles within the Experience Builder framework.

| **File Path** | **Purpose** | **Key Configuration / Snippet** |
| --- | --- | --- |
| manifest.json | Declares the widget's properties, dependencies, and settings to ExB.2 | "dependency": ["jimu-arcgis"]  18  "useMapWidgetIds":,  "setting": "src/setting/setting.tsx" |
| src/runtime/widget.tsx | The main React component for the widget's UI and runtime logic.15 | import { React, AllWidgetProps } from 'jimu-core'  const Widget = (props: AllWidgetProps<any>) => {... } |
| src/setting/setting.tsx | The React component for the widget's configuration panel.16 | import { MapWidgetSelector } from 'jimu-ui/advanced/setting-components'  props.onSettingChange({... }) |

## Part 3: Developing the Chatbot User Interface with React and Jimu-UI

### 3.1 UI Implementation in widget.tsx

The user interface for the chatbot will be built within the src/runtime/widget.tsx file. This will be a React functional component utilizing React Hooks, a modern approach for managing state and side effects in React components.2

### 3.2 State Management with React Hooks

The useState hook will be employed to manage the component's internal state:

1. **currentMessage**: A state variable of type string, initialized as ''. This will store the text as the user types it into the input box.
2. **messageHistory**: A state variable of type Array, initialized as ``. This array will store an ordered log of all messages, with each message being an object (e.g., { sender: 'user' | 'bot', text: '...' }). This enables the rendering of the complete chat history.22

### 3.3 Integrating jimu-ui Components

To ensure the custom widget has a consistent look and feel with the rest of the Experience Builder application, standard HTML elements (like <input> or <button>) will be avoided. Instead, the implementation will use the pre-styled components provided by the jimu-ui library.23

* **TextInput**: This component will be used for the user's command input. It is imported directly from the jimu-ui package: import { TextInput } from 'jimu-ui'.25 The component's placeholder prop will be set (e.g., "Enter command...") 26, and its onChange event will be wired to update the currentMessage state variable.
* **Button**: This component will be used to submit the command. It is also imported from jimu-ui: import { Button } from 'jimu-ui'.25 Its onClick event will be wired to a handler function that triggers the command parsing logic (detailed in Part 5).

### 3.4 Rendering the Chat History

A scrollable div element will serve as the chat log container. Within this container, the component will iterate over the messageHistory state array using the JavaScript .map() function. For each message object in the array, a div element will be rendered, dynamically applying different CSS classes based on the sender property ('user' or 'bot') to visually distinguish between user commands and bot responses.

## Part 4: Core Widget-to-Map Communication

### 4.1 The Central Architectural Challenge

The application currently contains two independent React components: the Map widget and the custom chatbot-widget. A communication channel must be established to allow the chatbot to programmatically control the map.18

### 4.2 The Solution: JimuMapViewComponent

The Experience Builder framework provides a dedicated component for this exact purpose: the JimuMapViewComponent. This component, imported from the jimu-arcgis library, acts as the bridge.11 Its function is to monitor the application state, find the MapView or SceneView instance associated with a specified Map widget ID, and provide that view object to the custom widget.

### 4.3 Step-by-Step Implementation

The connection is established through a two-part process involving both the setting and runtime files.

1. **Step 1 (in src/setting/setting.tsx):**
   * The setting.tsx file will import MapWidgetSelector from jimu-ui/advanced/setting-components.
   * This component renders a dropdown list in the widget's configuration panel, showing all available Map widgets on the page.
   * When the app author selects a map, its unique ID is saved into the widget's configuration props, specifically props.useMapWidgetIds.
2. **Step 2 (in src/runtime/widget.tsx):**
   * Import the necessary types: import { JimuMapViewComponent, type JimuMapView } from 'jimu-arcgis'.11
   * Define a new state variable to store the map view object: const [jimuMapView, setJimuMapView] = useState<JimuMapView>(null).11
   * In the component's JSX (render) output, add the JimuMapViewComponent. This component is invisible to the user.
   * It must be configured with two key props 18:
     1. useMapWidgetId={props.useMapWidgetIds?.}: This tells the component *which* map to listen for, using the ID saved from the settings panel.11
     2. onActiveViewChange={(jmv: JimuMapView) => { setJimuMapView(jmv) }}: This is the callback handler. When the specified map view becomes active and loaded, this function is called, passing in the JimuMapView object, which is then saved to the widget's state.11

### 4.4 The jimuMapView Object

Once the onActiveViewChange handler fires, the jimuMapView state variable is populated. This object is the "golden object" and the gateway to all subsequent GIS operations.

The JimuMapView object 28 is an ExB wrapper class. The two most important properties it provides are:

* **jimuMapView.view**: This property provides direct access to the underlying ArcGIS Maps SDK for JavaScript MapView or SceneView instance.28
* **jimuMapView.view.map**: This property provides access to the JSAPI Map object itself, which contains the collection of all layers.11

All chatbot action functions (for geocoding, layer visibility, and printing) will be designed to first check if (jimuMapView) to ensure the map is ready before attempting any operation.

## Part 5: Implementing the Command Parsing Engine

### 5.1 Logic Flow

The onClick event handler for the "Send" Button will serve as the entry point.

1. It will retrieve the currentMessage string from the React state.
2. It will add the message to the messageHistory as a 'user' message.
3. It will pass the currentMessage string to a central processing function, parseCommand(command: string).
4. This function will be responsible for identifying the user's intent and dispatching the command to the appropriate action function (e.g., executeGeocode).
5. The action functions will then add their own 'bot' responses to the messageHistory.

### 5.2 Parsing Simple Keywords (Switch Statement)

For simple, single-word commands like "print," a standard TypeScript switch statement provides a clean and efficient parsing mechanism.29 The command string will be normalized (e.g., command.toLowerCase().trim()) before being evaluated.31

Example:

switch (command.toLowerCase().trim()) {

case 'print':

executePrint();

break;

...

}

### 5.3 Parsing Complex Commands (Regular Expressions)

For commands that include parameters, such as a postcode or a layer name, Regular Expressions (Regex) are required to validate the pattern and extract the values.

* **Postcode Parsing:** A Regex will be defined to validate a postcode. For a simple US ZIP code, this could be const postcodeRegex = /^\d{5}(-\d{4})?$/.32 The parseCommand function will test the input: if (postcodeRegex.test(command)) { executeGeocode(command); }. This can be expanded to support other regional formats (e.g., UK postcodes 35).
* **Layer Visibility Parsing:** A Regex with a capturing group is needed to extract the layer name. Example: const layerRegex = /^enable ('.+'|".+")$/i. This pattern looks for the keyword "enable" followed by a space, and then a layer name enclosed in either single or double quotes. The layer name can then be extracted from the match results.

### Table 2: Chatbot Command Parsing Logic

This table maps the user's intended command to the specific parsing method and the internal function that will be executed.

| **User Command (Example)** | **Parsing Method** | **Regex / Keyword** | **Captured Value(s)** | **Target Function** | **Core API/Module** |
| --- | --- | --- | --- | --- | --- |
| 90210 | Regex Test | ^\d{5}(-\d{4})?$ 32 | 90210 | executeGeocode(command) | esri/rest/locator 36 |
| enable "Park Boundaries" | Regex Match | `/^enable ('.+' | ".+")$/i` | "Park Boundaries" | executeLayerToggle(layerName, true) |
| disable "Park Boundaries" | Regex Match | `/^disable ('.+' | ".+")$/i` | "Park Boundaries" | executeLayerToggle(layerName, false) |
| print | Keyword Match | print | N/A | executePrint() | esri/rest/print 38 |
| find schools in view | Regex Match | /^find (.+) in view$/i | schools | executeQuery(layerName) | FeatureLayer.queryFeatures() 39 |

## Part 6: Implementing Chatbot Action 1: Geocoding and Navigation

### 6.1 Fulfilling the Requirement

This section implements the functionality to meet the requirement: "if i ask any postcode it should navigate to that postcode."

### 6.2 JSAPI Module

This action requires the locator module from the ArcGIS Maps SDK for JavaScript.36 This is imported at the top of widget.tsx:

import \* as locator from '@arcgis/core/rest/locator'.36

### 6.3 Geocoding Service URL

A geocoding service is required to convert the postcode (a string) into a geographic location (coordinates). The ArcGIS Online World Geocoding Service will be used, which is a standard REST endpoint available to all ArcGIS Online accounts.36

const geocodingServiceUrl = "https://geocode.arcgis.com/arcgis/rest/services/World/GeocodeServer".36

### 6.4 Code Implementation (executeGeocode function)

The executeGeocode function will be defined as an async function to handle the asynchronous nature of the API call.

1. **Step 1:** The function receives the postcode string (e.g., "90210") from the parseCommand function.
2. **Step 2:** A 'bot' message is added to the messageHistory state: "Searching for {postcode}...".
3. Step 3: The parameters for the geocoding request are defined. For a simple address, the SingleLine address field is sufficient.36  
   const params = { address: { SingleLine: postcode }, outFields: ["\*"] }
4. Step 4: The locator.addressToLocations() method is called and await-ed.36 This method sends the request to the geocoding service and returns a Promise.41  
   const candidates = await locator.addressToLocations(geocodingServiceUrl, params);
5. **Step 5:** The promise resolves to an array of AddressCandidate objects.44 The code must check if any results were returned: if (candidates && candidates.length > 0).
6. **Step 6:** The first result is assumed to be the best match: const bestMatch = candidates.
7. **Step 7:** The location property (a Point geometry) is extracted from bestMatch.location.
8. Step 8: The jimuMapView state object is used to navigate the map. The jimuMapView.view.goTo() method is called, which provides a smooth, animated zoom to the candidate's location.46  
   jimuMapView.view.goTo(bestMatch.location);
9. **Step 9:** A success message is added to the chat history: "Navigating to {bestMatch.address}." If no candidates were found, a "Postcode not found" message is added.

## Part 7: Implementing Chatbot Action 2: Layer Visibility Control

### 7.1 Fulfilling the Requirement

This section implements the functionality to meet the requirement: "if i ask any layer to enable it should enable."

### 7.2 JSAPI Module

No new module imports are required for this action. The necessary objects are already available via the jimuMapView state object.28

### 7.3 Code Implementation (executeLayerToggle function)

1. **Step 1:** The function receives the extracted layer name (e.g., "My Layer") and a boolean enable flag from the parseCommand function.
2. **Step 2:** The layer name string is cleaned to remove the quotation marks captured by the Regex.
3. **Step 3:** The function accesses the map's complete, flattened list of layers via the jimuMapView object: const layers = jimuMapView.view.map.allLayers.37
4. Step 4: The allLayers property is a JSAPI Collection object 50, which provides utility methods like .find(). This method is used to iterate the collection and find the first layer whose title property matches the user's input.37  
   const targetLayer = layers.find(layer => layer.title === layerName);
5. **Step 5:** The code checks if targetLayer was found.
6. **Step 6:** If a match is found, the layer's visibility is set directly by modifying its visible property: targetLayer.visible = enable.11 The ExB framework and JSAPI are reactive; this direct property manipulation will immediately update the map's rendering. This is a more direct and efficient approach than programmatically interacting with the Map Layers widget.53
7. **Step 7:** A feedback message is added to the chat: Layer '{layerName}' is now {enable? 'visible' : 'hidden'}. If targetLayer was undefined, the message Layer '{layerName}' not found. is added.

## Part 8: Implementing Chatbot Action 3: Programmatic Printing

### 8.1 Fulfilling the Requirement

This section implements the functionality to meet the requirement: "if ask print it should print."

### 8.2 JSAPI Module

This action requires importing the print module and its associated helper classes from the JSAPI.38

import \* as print from '@arcgis/core/rest/print' 38

import PrintTemplate from '@arcgis/core/rest/support/PrintTemplate' 38

import PrintParameters from '@arcgis/core/rest/support/PrintParameters' 38

### 8.3 Print Service URL

This functionality relies on an ArcGIS Server PrintTask, an asynchronous geoprocessing service.55 Every ArcGIS Online and ArcGIS Enterprise portal provides a default utility service for this purpose.56

const printServiceUrl = "https://utility.arcgisonline.com/arcgis/rest/services/Utilities/PrintingTools/GPServer/Export%20Web%20Map%20Task".56

### 8.4 Code Implementation (executePrint function)

This function will also be defined as async.

1. **Step 1:** A 'bot' message is added to the chat: "Printing... This may take a moment."
2. Step 2: A PrintTemplate object is instantiated. This defines the output format, layout, and other cartographic elements.38 This mirrors the options available in the out-of-the-box Print widget.57  
   const template = new PrintTemplate({  
   format: "pdf",  
   layout: "a4-portrait",  
   layoutOptions: { titleText: "My Map Print", authorText: "Chatbot Widget" }  
   });
3. **Step 3:** A PrintParameters object is instantiated.38
4. Step 4: This is the most critical step in the process: the view property of the PrintParameters is set to jimuMapView.view.38  
   const params = new PrintParameters({  
   view: jimuMapView.view,  
   template: template  
   });  
   This action ensures that the print service receives the exact current map extent, scale, and layer visibilities (including those set in Part 7) from the user's Map widget.
5. Step 5: The print.execute() method is called and await-ed.38 This sends the asynchronous request to the printServiceUrl.  
   const result = await print.execute(printServiceUrl, params);
6. **Step 6:** The promise resolves with a result object. The result.url property contains a direct URL to the generated output file (e.g., the PDF).
7. **Step 7:** A final 'bot' message is added to the chat, embedding the link for the user: Print complete. <a href="{result.url}" target="\_blank">Download PDF</a>.

## Part 9: Expanding Chatbot Capabilities (Further Research)

The chatbot interface provides a powerful and extensible platform for GIS operations, addressing the user's request for "any other... features."

### 9.1 Feature 1: Query Features in View

A common GIS operation is to find features within the current map view, a function normally provided by the Query widget.39

* **Command:** find schools in view
* **Logic:** The parseCommand function would extract "schools" as the layerName.
* **Implementation:**
  1. Find the "schools" layer (a FeatureLayer) using the allLayers collection, as shown in Part 7.
  2. If found, get the current map extent from the jimuMapView: const viewExtent = jimuMapView.view.extent.62
  3. Import Query from @arcgis/core/rest/support/Query.
  4. Create a new Query object: const query = new Query();
  5. Set the query parameters: query.geometry = viewExtent, query.spatialRelationship = "intersects", query.outFields = ["\*"], query.returnGeometry = false.
  6. Call the layer's queryFeatures() method: const featureSet = await featureLayer.queryFeatures(query);
  7. Add a 'bot' response: Found {featureSet.features.length} schools in the current view.

### 9.2 Feature 2: Get Coordinates on Click

This mimics the Coordinates widget 63 or Coordinate Conversion widget 64, allowing a user to capture map coordinates.

* **Command:** get coordinates
* **Logic:** This command would toggle a state variable in the widget, isPickingCoordinates = true.
* **Implementation:**
  1. The JimuMapViewComponent can be configured to provide an onClick handler for the map view.
  2. This handler would check if (isPickingCoordinates).
  3. If true, it would capture the mapPoint from the event.
  4. Add a 'bot' response: Coordinates: {mapPoint.longitude.toFixed(6)}, {mapPoint.latitude.toFixed(6)}.
  5. Set isPickingCoordinates = false to end the "picking" mode.

### 9.3 Feature 3: Generative AI Integration

The chatbot architecture being built is a *command parser*, not a true artificial intelligence. It uses rigid switch statements and Regular Expressions to match predefined patterns.

A more advanced, future-state architecture would integrate a true Generative AI model.65 In this model:

1. The ExB chatbot widget would *not* contain any Regex or switch logic.
2. When the user types a natural language query (e.g., "Show me all parks near Beverly Hills, and then print the map"), the *entire* string is sent to an external AI API (like GPT-4, or a specialized GeoAI model 65).
3. The AI model would be trained to parse the GIS-specific intent and entities. It would return a structured JSON object, such as:  
   { "action": "sequence", "steps":}
4. The ExB widget's only job would be to receive this JSON and *execute* the steps, using the same executeGeocode, executeLayerToggle, and executePrint functions. This separates the complex Natural Language Processing (NLP) from the GIS execution, making the widget infinitely more powerful and flexible.

## Part 10: Final Deployment and Registration in ArcGIS Online

The local developer server (npm start) is only for development and debugging.1 To use this widget in a production application or share it with others, it must be compiled, deployed to a web server, and registered with your ArcGIS Online organization.13

### 10.1 Step 1: Compile for Production

1. Stop the npm start client service in its terminal.
2. From the same client directory, run the production build command: npm run build:prod.13
3. This command will compile, minify, and bundle the widget's TypeScript and React code into optimized JavaScript.
4. The output will be placed in a new dist-prod folder (or dist-download in some versions).13 The compiled widget will be located at dist-prod/widgets/chatbot-widget.

### 10.2 Step 2: Host the Compiled Widget

1. Copy the *entire* chatbot-widget folder from the dist-prod/widgets directory.
2. Paste this folder onto any web-accessible server.13 This could be a traditional web server (IIS, Apache), a cloud storage provider (Amazon S3, Azure Blob Storage), or any location that can serve static files over HTTPS.
3. The objective is to obtain a permanent, public URL to the manifest.json file within that hosted folder.13
   * Example URL: https://my-company-server.com/exb-widgets/chatbot-widget/manifest.json

### 10.3 Step 3: Register the Widget in ArcGIS Online

This step registers the hosted widget with your ArcGIS Online organization, making it available to the standard, cloud-based Experience Builder.

1. Log in to your ArcGIS Online portal.13
2. Navigate to the "My Content" tab.13
3. Click "Add Item" and choose "An application".13
4. In the "Application type" dropdown, select **"Experience Builder widget"**.13
5. In the "URL" field, paste the public URL to your hosted manifest.json file.13
6. Click "Next." The title will auto-populate from the manifest.13 Add relevant tags (e.g., "chatbot," "custom widget") and click "Add Item."

### 10.4 Step 4: Use the Widget in ArcGIS Online

The custom widget is now an item in your portal, fully registered with your organization.67

1. Open the standard, non-developer Experience Builder from the ArcGIS Online app launcher.
2. Create a new Experience or open the application created in Part 1.
3. Open the "Insert widget" panel.
4. Scroll to the bottom. Your "Chatbot Widget" will appear in the "Custom" section, ready to be added to any application, configured, and used in a production environment.67

## Part 11: Generative AI Prompts for Widget Creation (for "Claude Sonut")

### 11.1 Instructions for Use

The following prompts are designed to be provided sequentially to a generative AI model, such as the one referred to as "Claude Sonut." These prompts will guide the AI to generate the complete code for all three required files, incorporating the necessary technical context from the ArcGIS "Jimu" and "JSAPI" libraries.

### 11.2 Sequential AI Prompts

**Prompt 1: Project Setup & manifest.json**

"You are an expert ArcGIS Experience Builder developer. Your task is to create a custom widget.

First, create the manifest.json file for a new widget named chatbot-widget. This widget needs to:

1. Have a label of "Chatbot Widget".
2. Be built with React.
3. Critically, it must depend on the "jimu-arcgis" library 18 so it can use the ArcGIS Maps SDK for JavaScript.
4. It must use a map widget, so add the useMapWidgetIds property as an empty array.
5. It must have a settings panel, so define the setting property to point to src/setting/setting.tsx.16

Provide the complete code for the manifest.json file."

**Prompt 2: setting.tsx (Map Widget Selection)**

"Next, create the src/setting/setting.tsx file. This file defines the widget's configuration panel in the builder.

The UI must:

1. Import React.
2. Import AllWidgetSettingProps from jimu-for-builder.
3. Import MapWidgetSelector from jimu-ui/advanced/setting-components.
4. Import SettingSection and SettingRow from jimu-ui/advanced/setting-components.
5. Define a functional component Setting(props: AllWidgetSettingProps<any>).
6. Inside the component, define a function onMapWidgetSelected that takes a useMapWidgetIds: string array and uses props.onSettingChange to save this array to the widget's configuration.
7. The JSX should render a SettingSection with a SettingRow that contains the <MapWidgetSelector> component.11
8. The <MapWidgetSelector> must be bound to props.useMapWidgetIds and use the onMapWidgetSelected handler.

Provide the complete code for src/setting/setting.tsx."

**Prompt 3: widget.tsx (UI and State Management)**

"Now, create the main src/runtime/widget.tsx file. We will build this in steps.

**Step 1: UI and State**

1. Import React, type AllWidgetProps from jimu-core.15
2. Import React's useState hook.
3. Import TextInput and Button from jimu-ui.25
4. Import css from jimu-core to add styles.
5. Define a Message interface: interface Message { sender: 'user' | 'bot'; text: string; }.
6. Create the Widget(props: AllWidgetProps<any>) functional component.
7. Create two state variables:
   * const [currentMessage, setCurrentMessage] = useState<string>('');
   * const [messageHistory, setMessageHistory] = useState<Message>();
8. The JSX should render a chat interface. Create a widgetStyle variable using css for basic styling (e.g., a scrollable message container).
   * A scrollable div (the message list) that maps over messageHistory and displays the messages.
   * A div (the input area) containing a <TextInput> 25 bound to currentMessage (value and onChange) and a <Button> 25 with an onClick handler.
9. Create the handleSend function. For now, it should:
   * Check if currentMessage is not empty.
   * Add the currentMessage to messageHistory as a 'user' message.
   * Call a placeholder function parseCommand(currentMessage).
   * Clear currentMessage.
10. Create an empty parseCommand(command: string) function.

Provide the code for src/runtime/widget.tsx with this UI and state logic."

**Prompt 4: widget.tsx (Adding JimuMapViewComponent and Command Parser)**

"Now we will update src/runtime/widget.tsx to connect to the map and parse commands.

**Step 2: Map Connection and Command Parsing**

1. Import JimuMapViewComponent and type JimuMapView from jimu-arcgis.11
2. Add a new state: const [jimuMapView, setJimuMapView] = useState<JimuMapView>(null).11
3. In the JSX, add the <JimuMapViewComponent>.27 It must be configured with:
   * useMapWidgetId={props.useMapWidgetIds?.}.11
   * onActiveViewChange={(jmv: JimuMapView) => setJimuMapView(jmv)}.11
4. Create an addBotMessage(text: string) helper function that adds a 'bot' message to the messageHistory state.
5. Implement the parseCommand(command: string) function:
   * Check if (!jimuMapView) and call addBotMessage('Map is not loaded. Please wait.').
   * Define a postcode Regex: const postcodeRegex = /^\d{5}(-\d{4})?$/.32
   * Define a layer enable Regex: const enableRegex = /^enable ('.+'|".+")$/i.
   * Define a layer disable Regex: const disableRegex = /^disable ('.+'|".+")$/i.
   * Use if (postcodeRegex.test(command)) to call executeGeocode(command).
   * Use if (enableRegex.test(command)) to extract the layer name and call executeLayerToggle(layerName, true).
   * Use if (disableRegex.test(command)) to extract the layer name and call executeLayerToggle(layerName, false).
   * Use a switch (command.toLowerCase().trim()) 29 for the 'print' case, calling executePrint().
   * Add a default case to call addBotMessage('Unknown command.').
6. Create placeholder async functions: executeGeocode(postcode: string), executeLayerToggle(layerName: string, enable: boolean), and executePrint().

Provide the updated code for src/runtime/widget.tsx."

**Prompt 5: widget.tsx (Geocoding Function)**

"Update src/runtime/widget.tsx. We will now implement the executeGeocode function.

**Step 3: Geocoding and Navigation**

1. Import \* as locator from @arcgis/core/rest/locator.36
2. Define geocodingServiceUrl = 'https://geocode.arcgisonline.com/arcgis/rest/services/World/GeocodeServer'.36
3. Implement the async function executeGeocode(postcode: string):
4. Inside a try...catch block:
   * Call addBotMessage(\Searching for ${postcode}...`)`.
   * Define params = { address: { SingleLine: postcode }, outFields: ["\*"] }.36
   * const results = await locator.addressToLocations(geocodingServiceUrl, params).36
   * Check if results && results.length > 0.44
   * If true, get const bestMatch = results.
   * Call jimuMapView.view.goTo(bestMatch.location) 48 to zoom the map.
   * Call addBotMessage(\Navigated to ${bestMatch.address}`)`.
   * If false, call addBotMessage('Postcode not found.').
5. In the catch block, call addBotMessage('Geocoding error.').

Provide the updated code for src/runtime/widget.tsx including the new import and the fully implemented executeGeocode function."

**Prompt 6: widget.tsx (Layer Visibility Function)**

"Update src/runtime/widget.tsx. Implement the executeLayerToggle function.

**Step 4: Layer Visibility**

1. No new imports are needed.
2. Implement the async function executeLayerToggle(rawLayerName: string, enable: boolean):
3. Inside a try...catch block:
   * Clean the rawLayerName by removing the surrounding quotes: const layerName = rawLayerName.substring(1, rawLayerName.length - 1).
   * Find the layer: const targetLayer = jimuMapView.view.map.allLayers.find(layer => layer.title === layerName).37
   * If targetLayer is found:
     + Set targetLayer.visible = enable.11
     + Call addBotMessage(\Layer '${layerName}' is now ${enable? 'visible' : 'hidden'}`)`.
   * If not found, call addBotMessage(\Layer '${layerName}' not found.`)`.
4. In the catch block, call addBotMessage('Layer visibility error.').

Provide the updated code for src/runtime/widget.tsx with the implemented executeLayerToggle function."

**Prompt 7: widget.tsx (Print Function)**

"Update src/runtime/widget.tsx. Implement the executePrint function.

**Step 5: Programmatic Printing**

1. Import \* as print from @arcgis/core/rest/print.38
2. Import PrintTemplate from @arcgis/core/rest/support/PrintTemplate.38
3. Import PrintParameters from @arcgis/core/rest/support/PrintParameters.38
4. Define printServiceUrl = 'https://utility.arcgisonline.com/arcgis/rest/services/Utilities/PrintingTools/GPServer/Export%20Web%20Map%20Task'.56
5. Implement the async function executePrint():
6. Inside a try...catch block:
   * Call addBotMessage('Generating print...').
   * Create const template = new PrintTemplate({ format: 'pdf', layout: 'a4-portrait',... }).38
   * Create const params = new PrintParameters({ view: jimuMapView.view, template: template }).38
   * const result = await print.execute(printServiceUrl, params).38
   * Call addBotMessage(\Print complete. Download PDF`)`.
7. In the catch block, call addBotMessage('Print service error.').

Provide the updated code for src/runtime/widget.tsx with the new imports and the fully implemented executePrint function."

**Prompt 8: Consolidate and Review**

"Please consolidate all the code from the previous prompts. Provide the final, complete, and production-ready code for all three files:

1. manifest.json
2. src/setting/setting.tsx
3. src/runtime/widget.tsx

Ensure all imports are correct, all async functions use try...catch for error handling, and the chat UI renders the 'bot' messages as raw HTML (using dangerouslySetInnerHTML) to make the print link clickable. This is the final step."

#### Works cited

1. How to install Experience Builder developer edition - Esri Developer - ArcGIS Online, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/install-guide/>
2. Getting started with widget development | ArcGIS Experience Builder, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/getting-started-widget/>
3. Make an in-app tutorial in ArcGIS Experience Builder - Esri, accessed on November 8, 2025, <https://www.esri.com/arcgis-blog/products/experience-builder/mapping/exb-make-an-in-app-tutorial>
4. Widgets | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/widgets-overview/>
5. Widgets—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/widgets-overview.htm>
6. Get started with ArcGIS Experience Builder: Foldable template - Esri, accessed on November 8, 2025, <https://www.esri.com/arcgis-blog/products/experience-builder/mapping/get-started-experience-builder-foldable>
7. Templates to tempt you in ArcGIS Experience Builder - Resource Centre | Esri UK, accessed on November 8, 2025, <https://resource.esriuk.com/blog/templates-to-tempt-you-in-arcgis-experience-builder/>
8. ArcGIS Experience Builder overview and concepts - Esri, accessed on November 8, 2025, <https://www.esri.com/arcgis-blog/products/experience-builder/mapping/experience-builder-overview>
9. Tutorial on using the Experience Builder - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-experience-builder-questions/tutorial-on-using-the-experience-builder/td-p/1145417>
10. Add and connect widgets—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/build-apps/add-widgets.htm>
11. Add layers to a map | ArcGIS Experience Builder | Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/add-layers-to-a-map/>
12. Tutorials | Documentation - Esri Developer - ArcGIS Online, accessed on November 8, 2025, <https://developers.arcgis.com/documentation/app-builders/low-code/arcgis-experience-builder/arcgis-experience-builder-tutorials/>
13. Add custom widgets—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/11.0/configure-widgets/add-custom-widgets.htm>
14. How do I add a custom widget to Developer Edition? - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-experience-builder-questions/how-do-i-add-a-custom-widget-to-developer-edition/td-p/1659977>
15. Create a starter widget | ArcGIS Experience Builder | Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/create-a-starter-widget/>
16. Calcite components in a custom widget | Documentation - Esri Developer - ArcGIS Online, accessed on November 8, 2025, <https://developers.arcgis.com/documentation/app-builders/low-code/arcgis-experience-builder/calcite-components-in-a-custom-widget/>
17. Widget manifest | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/widget-manifest/>
18. Use Map widget in widget | ArcGIS Experience Builder | Esri ..., accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/use-map-widget-in-widget/>
19. Widget implementation | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/extend-base-widget/>
20. React Chat Tutorial: How to build a chat app - GetStream.io, accessed on November 8, 2025, <https://getstream.io/chat/react-chat/tutorial/>
21. Reusing Logic with Custom Hooks - React, accessed on November 8, 2025, <https://react.dev/learn/reusing-logic-with-custom-hooks>
22. Creating a Chat Window Using React and Hooks - PubNub, accessed on November 8, 2025, <https://www.pubnub.com/blog/creating-a-chat-window-using-react-and-hooks/>
23. Jimu | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/core-concepts/jimu/>
24. Jimu UI | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/api-reference/jimu-ui/>
25. Create UI for widget | ArcGIS Experience Builder | Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/widget-ui/>
26. Components / jimu-ui / index / Select - Docs ⋅ Storybook, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/storybook/?path=/docs/components-jimu-ui-index-select--docs>
27. JimuMapViewComponent | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/api-reference/jimu-arcgis/JimuMapViewComponent/>
28. JimuMapView | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/api-reference/jimu-arcgis/JimuMapView/>
29. switch - JavaScript - MDN Web Docs - Mozilla, accessed on November 8, 2025, <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch>
30. TypeScript switch Statement - TutorialsTeacher, accessed on November 8, 2025, <https://www.tutorialsteacher.com/typescript/typescript-switch>
31. Type Safe and Exhaustive 'switch' statements, aka Pattern Matching in TypeScript - Medium, accessed on November 8, 2025, <https://medium.com/technogise/type-safe-and-exhaustive-switch-statements-aka-pattern-matching-in-typescript-e3febd433a7a>
32. ZIP code regex - UI Bakery, accessed on November 8, 2025, <https://uibakery.io/regex-library/zip-code>
33. JavaScript validation with regular expression: Check whether a given value is US zip code or not - w3resource, accessed on November 8, 2025, <https://www.w3resource.com/javascript-exercises/javascript-regexp-exercise-12.php>
34. 4.14. Validate ZIP Codes - Regular Expressions Cookbook, 2nd Edition [Book] - O'Reilly, accessed on November 8, 2025, <https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s14.html>
35. Regex for UK Post Code Validation implemented via JavaScript - Jon Does Flow, accessed on November 8, 2025, <https://www.jondoesflow.com/post/regex-for-uk-post-code-validation-implemented-via-javascript>
36. locator | API Reference | ArcGIS Maps SDK for JavaScript - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/api-reference/esri-rest-locator.html>
37. Map | API Reference | ArcGIS Maps SDK for JavaScript - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/api-reference/esri-Map.html>
38. print | API Reference | ArcGIS Maps SDK for JavaScript, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/api-reference/esri-rest-print.html>
39. Query widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/11.2/configure-widgets/query-widget.htm>
40. Find places | Overview | ArcGIS Maps SDK for JavaScript - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/tutorials/find-places/>
41. Locator | ArcGIS API for JavaScript 4.18 - Cartagene, accessed on November 8, 2025, <https://cartagene.cerema.fr/arcgis_js_api/sdk/latest/api-reference/esri-tasks-Locator.html>
42. ArcGIS Geocoding service | ArcGIS REST APIs - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/rest/geocode/>
43. Using the what3words Locator within ArcGIS API for JavaScript, accessed on November 8, 2025, <https://developer.what3words.com/tutorial/using-the-what3words-arcgis-locator-within-arcgis-api-for-javascript>
44. AddressCandidate | API Reference | ArcGIS API for JavaScript, accessed on November 8, 2025, <https://programs.iowadnr.gov/maps//apis/esri/library/arcgis_js_v36_sdk/arcgis_js_v36_sdk/arcgis_js_api/sdk/jsapi/addresscandidate.html>
45. Locator (legacy) | API Reference | ArcGIS API for JavaScript 3.46 - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/3/jsapi/locator.html>
46. Get map coordinates (function) | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/sample-code/widgets/get-map-coordinates-function/>
47. Solved: Experience Builder: Go to XY - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-experience-builder-questions/experience-builder-go-to-xy/td-p/1369243>
48. How do I simply zoom to an address? - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-experience-builder-questions/how-do-i-simply-zoom-to-an-address/td-p/1408722>
49. Search widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/11.0/configure-widgets/search-widget.htm>
50. Collection | API Reference | ArcGIS Maps SDK for JavaScript - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/api-reference/esri-core-Collection.html>
51. Layer | API Reference | ArcGIS Maps SDK for JavaScript - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/javascript/latest/api-reference/esri-layers-Layer.html>
52. Map Layers widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/10.9.1/configure-widgets/map-layers-widget.htm>
53. Map Layers widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/map-layers-widget.htm>
54. Unable to toggle the visibility of the map layers in the Map Layers widget in ArcGIS Experience Builder - Esri Support, accessed on November 8, 2025, <https://support.esri.com/en-us/knowledge-base/problem-unable-to-toggle-the-visibility-of-the-map-laye-000028614>
55. PrintTask | ArcGIS API for JavaScript 4.18 - Cartagene, accessed on November 8, 2025, <https://cartagene.cerema.fr/arcgis_js_api/sdk/latest/api-reference/esri-tasks-PrintTask.html>
56. Print widget—ArcGIS Web AppBuilder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/web-appbuilder/latest/create-apps/widget-print.htm>
57. Print widget | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/print-widget/>
58. Print widget—ArcGIS Experience Builder | Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/11.2/configure-widgets/print-widget.htm>
59. Print widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/print-widget.htm>
60. What is the syntax for the printtask.execute errback? - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-server-with-javascript-api-questions/what-is-the-syntax-for-the-printtask-execute/td-p/770377>
61. Query widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/query-widget.htm>
62. Query widget—ArcGIS Experience Builder | Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/10.9.1/configure-widgets/query-widget.htm>
63. Coordinates widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/coordinates-widget.htm>
64. Coordinate Conversion widget—ArcGIS Experience Builder - Esri Documentation, accessed on November 8, 2025, <https://doc.arcgis.com/en/experience-builder/latest/configure-widgets/coordinate-conversion-widget.htm>
65. Integration of ArcGIS Experience Builder with GPT - Esri Community, accessed on November 8, 2025, <https://community.esri.com/t5/arcgis-experience-builder-videos/integration-of-arcgis-experience-builder-with-gpt/td-p/1385201>
66. Widget and theme deployment | ArcGIS Experience Builder - Esri Developer, accessed on November 8, 2025, <https://developers.arcgis.com/experience-builder/guide/widget-theme-deployment/>
67. Add Experience Builder Custom Widgets In ArcGIS Enterprise - Esri, accessed on November 8, 2025, <https://www.esri.com/arcgis-blog/products/arcgis-enterprise/developers/add-experience-builder-custom-widgets-in-arcgis-enterprise>