**Nanobrowser Chrome Extension**

**High-Level & Low-Level Design Document**

**High-Level Design**

**Overview**

Nanobrowser is a Chrome extension that automates browser tasks using AI agents powered by LLMs (e.g., Azure OpenAI).  
It consists of a UI for configuration, background scripts for agent orchestration, a DOM analysis module, and a storage layer for provider and agent settings.

+---------------------+

| User (Browser UI) |

+----------+----------+

|

v

+---------------------+

| Chrome Extension UI |

+----------+----------+

|

v

+---------------------+

| Background Script |

| (Agents, LLM Calls)|

+----------+----------+

|

v

+---------------------+

| Storage Layer |

| (Provider Configs, |

| Agent Models) |

+----------+----------+

|

v

+---------------------+

| LLM Provider APIs |

| (Azure OpenAI, etc) |

+---------------------+

**Main Components**

| **Component** | **Responsibility** | **Key Files/Functions** |
| --- | --- | --- |
| UI Layer | User config, agent settings | ModelSettings.tsx |
| Background Script | Agent orchestration, LLM calls | helper.ts, agent scripts |
| Storage Layer | Provider/agent config persistence | llmProviders.ts |
| DOM Analysis | DOM tree building, element highlighting | [buildDomTree.js](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) |
| LLM Integration | API calls to Azure OpenAI | createChatModel, createOpenAIChatModel |

**Low-Level Design**

**1. DOM Analysis & Highlighting (**[buildDomTree.js](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)**)**

**Responsibilities**

* Traverse the DOM, identify interactive/visible elements.
* Optionally highlight elements for automation/debugging.
* Cache DOM properties for efficient repeated access.

**Key Functions**

* [buildDomTree](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Recursively builds a tree of DOM nodes, annotating interactive/visible nodes.
* [highlightElement](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Draws overlays/labels on interactive elements.
* Caching helpers: [getCachedBoundingRect](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [getCachedComputedStyle](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [getCachedClientRects](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).

[Start]

|

v

[buildDomTree(document.body)]

|

v

[For each node:]

|

+--> [Is element accepted/visible?]

| |

| v

| [Is interactive?]

| |

| v

| [Highlight if enabled]

| |

| v

| [Add to DOM\_HASH\_MAP]

|

v

[Recurse into children]

|

v

[Return rootId and DOM\_HASH\_MAP]

**2. LLM Provider & Agent Model Flow**

**Responsibilities**

* Store and retrieve provider configs (API keys, endpoints).
* Always use Azure OpenAI for LLM calls (as per your requirements).
* Create chat model instances for agent workflows.

**Key Functions**

* getDefaultProviderConfig: Returns default config for Azure OpenAI.
* createChatModel / createOpenAIChatModel: Instantiates LLM client for agent use.
* handleModelChange: Forces Azure OpenAI and a specific model for all agent actions.

[User/Agent triggers LLM action]

|

v

[Retrieve provider config (always Azure OpenAI)]

|

v

[Create chat model instance]

|

v

[Send prompt to Azure OpenAI endpoint]

|

v

[Receive and process response]

|

v

[Agent continues workflow]

**3. Data Flow**

[User Input/Action]

|

v

[UI Layer] <--> [Background Script]

|

v

[buildDomTree.js] <--> [Storage Layer]

|

v

[LLM Provider APIs]

**Notes**

* **Highlighting** can be disabled by removing [showHighlightElements](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") or setting it to false in all relevant places.
* **LLM calls** are always routed to Azure OpenAI as per your modifications.
* **Extensibility**: The architecture supports adding more providers and agent types if needed.

**How to Download**

1. **Copy all the above content.**
2. **Paste into a new Word document (.docx).**
3. **Save the file.**

**Low-Level Design Summary**

**1. DOM Tree Builder (**[buildDomTree.js](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)**)**

* **Purpose:**  
  Traverses the DOM to build a hierarchical map of elements and text nodes.
* **Responsibilities:**
  + Identifies visible and interactive elements.
  + Optionally highlights elements for automation/debugging.
  + Handles special cases (iframes, shadow DOM, contenteditable).
  + Caches computed styles and bounding rectangles for performance.
  + Returns a tree structure and a map of node metadata.

**2. LLM Provider Management (llmProviders.ts)**

* **Purpose:**  
  Manages configuration for LLM providers (API keys, endpoints, deployment names).
* **Responsibilities:**
  + Stores provider configs in local storage.
  + Ensures backward compatibility for config formats.
  + Provides default configs (always Azure OpenAI in your setup).
  + Validates required fields before saving.

**3. Agent Model Selection (ModelSettings.tsx)**

* **Purpose:**  
  Lets users (or code) select which LLM model agents use for tasks.
* **Responsibilities:**
  + Presents available models and providers in the UI.
  + Handles model change events (but always forces Azure OpenAI and a specific model in your setup).
  + Persists agent model selections in storage.

**4. LLM Model Invocation (helper.ts)**

* **Purpose:**  
  Creates and configures chat model instances for LLM calls.
* **Responsibilities:**
  + Instantiates chat model objects using provider and model configs.
  + Handles Azure OpenAI specifics (endpoint, deployment, API version).
  + Passes parameters (temperature, topP, etc.) to the model.
  + Returns a model instance for use by agents.

**5. Agent Orchestration (Background Scripts)**

* **Purpose:**  
  Coordinates agent workflows and LLM interactions.
* **Responsibilities:**
  + Triggers DOM analysis and LLM calls based on user actions or automation scripts.
  + Uses the chat model instance to send prompts and receive responses.
  + Updates UI or browser state based on agent decisions.

**6. Storage Layer**

* **Purpose:**  
  Persists configuration and state for providers and agent models.
* **Responsibilities:**
  + Uses browser local storage for privacy.
  + Provides APIs for reading/writing configs and selections.

**Data Flow (Summary)**

1. **User/Agent triggers action**
2. **Background script requests DOM analysis or LLM response**
3. **DOM tree builder analyzes page, returns structure**
4. **Agent selects model (always Azure OpenAI)**
5. **LLM model instance is created and invoked**
6. **Response is processed and used for automation or UI update**

**1. DOM Tree Builder (**[buildDomTree.js](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)**)**

**Purpose**

* Recursively analyzes the DOM to build a hierarchical structure of elements and text nodes.
* Identifies interactive, visible, and actionable elements for browser automation.
* Optionally highlights elements for debugging or agent guidance.

**Key Features & Internal Logic**

* **Entry Point:**  
  [window.buildDomTree(args)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)  
  Accepts configuration for highlighting, viewport expansion, debug mode, etc.
* **Caching:**  
  Uses [WeakMap](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") caches for bounding rectangles, client rects, and computed styles to optimize repeated DOM queries.
* **Highlighting:**
  + [highlightElement(element, index, parentIframe)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) draws overlays and labels on interactive elements.
  + Uses a color palette and z-index to ensure overlays are visible.
  + Handles overlays in nested iframes and shadow DOM.
* **Visibility & Interactivity Checks:**
  + [isElementVisible(element)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Checks computed style and dimensions.
  + [isInteractiveElement(element)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Heuristically determines if an element is actionable (cursor style, tag, attributes, event listeners).
  + [isTopElement(element)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Ensures the element is not visually obscured.
  + [isInExpandedViewport(element, viewportExpansion)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Determines if the element is within the (optionally expanded) viewport.
* **XPath Generation:**
  + [getXPathTree(element)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Generates a unique XPath for each element, used for identification and automation.
* **Distinct Interaction Detection:**
  + [isElementDistinctInteraction(element)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Determines if an element represents a unique interaction (e.g., menu item inside a button).
* **Recursive Tree Building:**
  + [buildDomTree(node, parentIframe, isParentHighlighted)](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html):
    - Handles special cases for iframes, shadow DOM, contenteditable, and text nodes.
    - For each node, creates a data object with tag, attributes, XPath, children, and interactivity metadata.
    - Highlights interactive elements and assigns unique indices.
* **Output:**  
  Returns { rootId, map: DOM\_HASH\_MAP }, where [map](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) contains all node metadata indexed by unique IDs.

**2. LLM Provider Management (llmProviders.ts)**

**Purpose**

* Manages configuration for LLM providers (API keys, endpoints, deployment names).
* Ensures only Azure OpenAI is used (per your requirements).

**Key Features & Internal Logic**

* **Provider Config Structure:**
  + Stores apiKey, baseUrl, azureDeploymentNames, azureApiVersion, etc.
  + Uses local storage for persistence.
* **Default Provider:**
  + getDefaultProviderConfig(providerId): Returns a config object for Azure OpenAI, with default API version and empty endpoint.
* **Validation:**
  + Ensures required fields are present before saving (API key, endpoint, deployment name).
* **Backward Compatibility:**
  + ensureBackwardCompatibility(providerId, config): Adds missing fields and removes deprecated ones.
* **CRUD Operations:**
  + setProvider, getProvider, removeProvider, getAllProviders:
    - Save, retrieve, and delete provider configs.

**3. Agent Model Selection (ModelSettings.tsx)**

**Purpose**

* Allows users (or code) to select which LLM model agents use for tasks.
* In your setup, always forces Azure OpenAI and a specific model.

**Key Features & Internal Logic**

* **State Management:**
  + Uses React state for providers, selected models, and model parameters.
* **Model Change Handling:**
  + handleModelChange(agentName, modelValue):
    - Ignores user selection, always sets provider to Azure OpenAI and model to a hardcoded value (e.g., gpt-4o).
    - Updates state and persists selection.
* **Initialization:**
  + On mount, sets forced provider/model for all agents.
* **UI:**
  + Presents available models and providers, but backend always uses Azure OpenAI.

**4. LLM Model Invocation (helper.ts)**

**Purpose**

* Creates and configures chat model instances for LLM calls.

**Key Features & Internal Logic**

* **Model Creation:**
  + createOpenAIChatModel(providerConfig, modelConfig, extraFetchOptions):
    - Instantiates a chat model using provider and model configs.
    - Sets endpoint, API key, and parameters (temperature, topP, etc.).
    - Handles Azure OpenAI specifics (endpoint, deployment, API version).
* **Agent Model Flow:**
  + createChatModel(providerConfig, modelConfig):
    - Delegates to createOpenAIChatModel for OpenAI-compatible providers.
    - Returns a model instance for use by agents.

**5. Agent Orchestration (Background Scripts)**

**Purpose**

* Coordinates agent workflows and LLM interactions.

**Key Features & Internal Logic**

* **Workflow Management:**
  + Triggers DOM analysis and LLM calls based on user actions or automation scripts.
  + Uses chat model instances to send prompts and receive responses.
  + Updates UI or browser state based on agent decisions.
* **Integration Points:**
  + Calls [buildDomTree](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") for DOM analysis.
  + Calls createChatModel for LLM responses.

**6. Storage Layer**

**Purpose**

* Persists configuration and state for providers and agent models.

**Key Features & Internal Logic**

* **Local Storage:**
  + Uses browser local storage for privacy.
  + Provides APIs for reading/writing configs and selections.

**Data Flow (Detailed)**

1. **User/Agent triggers action (e.g., automation, analysis)**
2. **Background script requests DOM analysis or LLM response**
   * Calls [buildDomTree](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") to analyze the page and identify actionable elements.
   * Calls createChatModel to instantiate an LLM model (always Azure OpenAI).
3. **DOM tree builder returns structure and metadata**
   * Used for agent navigation, automation, or UI highlighting.
4. **Agent selects model (always Azure OpenAI)**
   * Model selection is forced in code.
5. **LLM model instance is created and invoked**
   * Sends prompt to Azure OpenAI endpoint.
   * Receives and processes response.
6. **Response is processed and used for automation or UI update**
   * Agent continues workflow or updates browser state.

**Interactions & Extensibility**

* **Highlighting:**  
  Can be disabled by removing [showHighlightElements](vscode-file://vscode-app/d:/Software/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") or setting it to false everywhere.
* **LLM Providers:**  
  Architecture supports adding more providers, but currently forced to Azure OpenAI.
* **Agents:**  
  Modular design allows adding more agent types or workflows.