Документация: Динамический UI-Генератор

Назначение

Этот набор классов предназначен для динамической генерации пользовательского интерфейса на основе JSON-схемы, получаемой, например, с сервера. Он позволяет создавать интерфейсы с вкладками, группами, параметрами (readonly, number, flags) и командами (кнопками), обеспечивая гибкость, масштабируемость, безопасность (предотвращение утечек памяти) и модульность.

Архитектура

Система построена на принципах компоновки, наследования и публикации/подписки (Event Bus).

Основные компоненты:

1. UIGenerator: Главный класс. Принимает stateSchema и elementHandlers. Отвечает за инициализацию всех подсистем и запуск процесса генерации UI.
2. ElementFactory: Создает экземпляры конкретных классов элементов UI (TabElement, ParameterElement, CommandElement и т.д.) на основе данных из stateSchema. Обеспечивает слабую связанность UIGenerator с конкретными реализациями элементов.
3. UIElement (и потомки): Базовый класс для всех элементов UI. Определяет общие свойства (id, visibility, enabled) и методы (render, update, destroy). Потомки (TabElement, GroupElement, ParameterElement, CommandElement) реализуют специфичную логику отображения и взаимодействия.
4. UIStateManager: Управляет состоянием UI. Регистрирует все сгенерированные элементы и отвечает за их обновление (например, изменение значения параметра или видимости).
5. EventManager: Централизованный "шина событий". Элементы UI публикуют события (например, PARAMETER_VALUE_CHANGED, COMMAND_CLICKED). UIStateManager, UIGenerator, DataConnector и другие компоненты подписываются на эти события и реагируют на них.
6. DataConnector: Отвечает за взаимодействие с сервером (например, через WebSocket). Подписывается на события EventManager, отправляет их на сервер и публикует полученные от сервера обновления обратно в EventManager.
7. Logger: Управляет логированием с различными уровнями (error, warn, info, debug). Позволяет отслеживать работу системы и находить ошибки.
8. SchemaValidator: Проверяет корректность stateSchema перед её обработкой. Повышает надежность системы.

Использование

1. Подключение скриптов Убедитесь, что все файлы .js подключены в правильном порядке (согласно зависимостям) в вашем HTML или собраны в бандл.

2. Настройка elementHandlers Создайте объект с обработчиками, которые будут вызываться при взаимодействии с UI (например, изменение параметра, клик команды).

```
const myElementHandlers = {
    onParameterChange: (paramId, value) => {
        console.log(`Параметр ${paramId} изменен на ${value}`);
        // Отправить на сервер, обновить локальное состояние и т.д.
    },
    onCommand: (commandId) => {
        console.log(`Выполнена команда ${commandId}`);
        // Отправить на сервер, выполнить действие и т.д.
    }
};
```

3. Создание DataConnector (опционально, но рекомендуется)

```
const dataConnector = new DataConnector(eventManagerFromGenerator); // Или
общий eventManager
dataConnector.connect('ws://localhost:8080/ws'); // URL вашего WebSocket
сервера
```

4. Создание и запуск UIGenerator

5. Обновление UI извне

```
// Обновить значение параметра (например, при получении данных от сервера)
generator.updateParameter('speed', 1500);

// Обновить видимость элемента
generator.updateElementVisibility('group1', false);
```

6. Уничтожение UI Когда UI больше не нужен (например, при переключении между контроллерами), вызовите destroyUI, чтобы избежать утечек памяти.

```
generator.destroyUI();
```

События

Компоненты публикуют и подписываются на следующие события через EventManager:

- PARAMETER_VALUE_CHANGED: Публикуется при изменении значения параметра в UI.
    - Данные: { paramId, value, controllerName }
- COMMAND_CLICKED: Публикуется при клике кнопки команды.
    - Данные: { commandId, controllerName }
- ELEMENT_VISIBILITY_CHANGED: Публикуется при изменении видимости элемента.
    - Данные: { elementId, isVisible }

- SCHEMA_UPDATE_RECEIVED: Публикуется DataConnector при получении новой схемы от сервера.
    - Данные: { schema }

Валидация схемы

UIGenerator автоматически проверяет stateSchema с помощью SchemaValidator. При ошибках в схеме выводится сообщение в лог.

Логирование

Все компоненты используют Logger. Уровень логирования можно изменить в GlobalLogger или передавая индивидуальный Logger в конструкторы.

Безопасность и производительность

- Утечки памяти: Предотвращаются с помощью метода destroy() у всех UIElement и его потомков, который отписывает слушателей и очищает ссылки.
- Производительность: Для предотвращения подвисания можно разбивать большие операции на части с помощью requestAnimationFrame или setTimeout.
- Обработка ошибок: Критические участки кода обернуты в try...catch, особенно в EventManager.