Zowe Desktop Application State Persistence Mechanism & REST APIs for Files and Datasets

Mitesh Goplani

miteshgoplani@gmail.com

Student's Name:

Mitesh Goplani

Mentor:

Sean Grady

Project:

Zowe Desktop Application State Persistence Mechanism & REST APIs for Files and Datasets

Project Description:

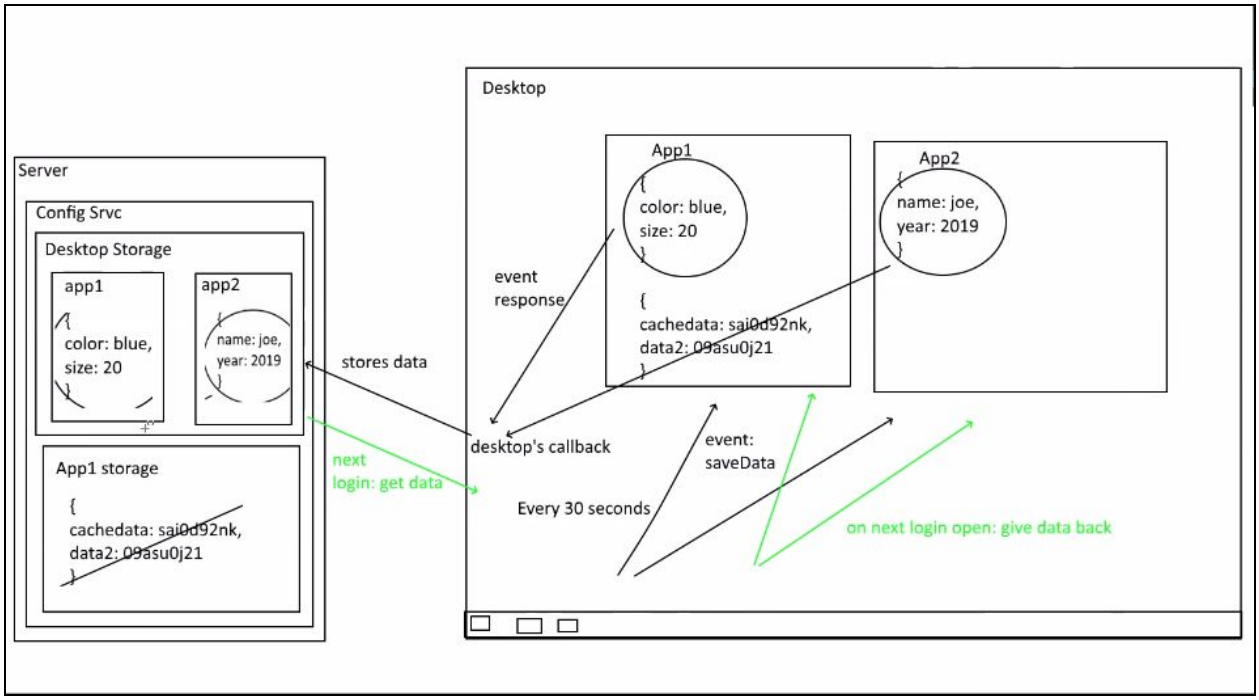
Web plugins running within the application framework do not have access to a secure state persistence mechanism that would allow a user to restore one or more plugins to an early state when restarting a session. The App Framework needs to provide this capability.

The mainframe has some additional features for files than a standard unix/linux system such as encoding, tagging, additional permissions and a whole other type of "file" called dataset and there currently is no perfect server that has all the command APIs required. The APIs for Zosmf and ZSS can be similar but the name of the variables and the shape of the json returned is different for both these servers. New APIs need to be created to accommodate the remaining commands and existing APIs need to be improved to transform the data in the right shape for the program that requested it.

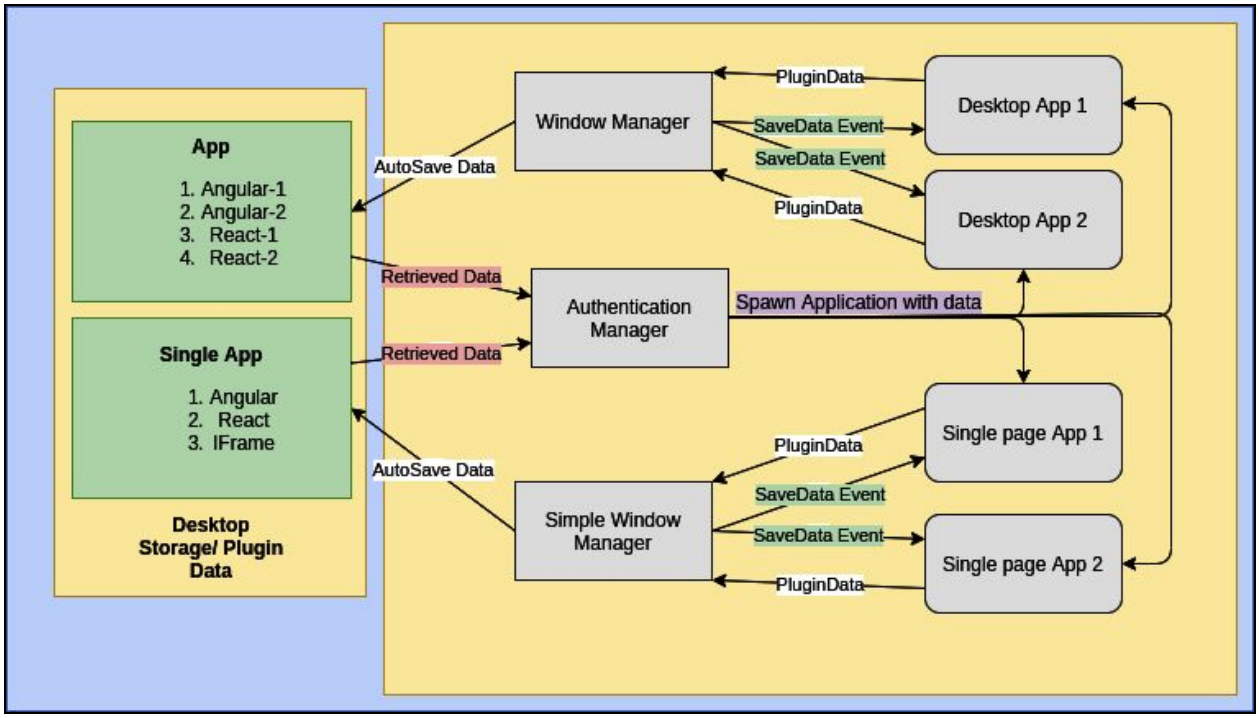
Deliverables:

1) State Persistence for Zowe Desktop Applications
2) REST APIs for Files and Datasets (Adding new ones and enhancements to https://github.com/zowe/zlux/wiki/URI-Broker to do the zss & zosmf data-shaping)

Architecture Design



Detailed Design



Explanation:

1. **Window Manager** and **Simple Window Manager** issues a global **saveData** event for Desktop Apps and Single Page Apps.
2. The Apps respond back with the data they need to Save.
3. The Manager services use **Configuration Data Service** and saves the data in Desktop's storage
4. On the next login/refresh action, the **Authentication Manager** issues some post-login actions and spawns Applications with the data.
5. As soon as the data is recovered the **old data** stored **is deleted** to avoid data redundancy.
6. In case of multiple desktop windows, the **window** which was **most recently opened** takes control of the AutoSave data.

Data Retrieval:

1. **Angular App Example**

   This example demonstrates how to save the `parameters` and `AppId` of Angular App:

   1. Inject the Session Events token

```
@Inject(Angular2InjectionTokens.SESSION_EVENTS) private
sessionEvents: Angular2PluginSessionEvents
```

   2. Subscribe to autosaveEmitter

```
this.autoSaveEvent =
this.sessionEvents.autosaveEmitter.subscribe((saveThis: any)=> {

 if (saveThis) {

saveThis({'appData':{'requestText':this.parameters,'targetAppId':thi
s.targetAppId}});

 }

});
```

3. Receive saved data

In order to receive the data, use `data.appData.requestText` **and** `data.appData.targetAppId` **as seen in** `handleLaunchOrMessageObject()`

Note: ***It is necessary to unsubscribe to the AutoSave Event when the component id destroyed***

2. React App Example

This example demonstrates how to save the parameters and AppId of React App:

1. Retrieve the sessionEvents from the resources props

```
this.sessionEvents = this.props.resources.sessionEvents;
```

2. Subscribe to autosaveEmitter

```
this.autoSaveEvent =
this.sessionEvents.autosaveEmitter.subscribe((saveThis: any)=> {

 if (saveThis) {

      saveThis({'appData':{'requestText':this.state.parameters,'targ
etAppId':this.state.appId}});

       }

});
```

3. Receive saved data

In order to receive the data, use **data.appData.requestText** and **data.appData.targetAppId** as seen in **handleLaunchOrMessageObject()**

Note**: It is necessary to unsubscribe to the AutoSave Event when the component unmounts**