**Storing data in the user's device**

There are times when we need to store some persisted pieces of data in the user's device. One such common scenario is storing the user's authentication token so that we can retrieve it even if the user closes and reopens our application. In web development, we have used the browser's localStorage object to achieve such functionality. React Native provides similar persistent storage, the [AsyncStorage](https://react-native-async-storage.github.io/async-storage/docs/usage/).

We can use the npx expo install command to install the version of the *@react-native-async-storage/async-storage* package that is suitable for our Expo SDK version:

npx expo install @react-native-async-storage/async-storage

The API of the AsyncStorage is in many ways same as the localStorage API. They are both key-value storages with similar methods. The biggest difference between the two is that, as the name implies, the operations of AsyncStorage are *asynchronous*.

Because AsyncStorage operates with string keys in a global namespace it is a good idea to create a simple abstraction for its operations. This abstraction can be implemented for example using a [class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes). As an example, we could implement a shopping cart storage for storing the products user wants to buy:

import AsyncStorage from '@react-native-async-storage/async-storage';

class ShoppingCartStorage {

constructor(namespace = 'shoppingCart') {

this.namespace = namespace;

}

async getProducts() {

const rawProducts = await AsyncStorage.getItem(

`${this.namespace}:products`,

);

return rawProducts ? JSON.parse(rawProducts) : [];

}

async addProduct(productId) {

const currentProducts = await this.getProducts();

const newProducts = [...currentProducts, productId];

await AsyncStorage.setItem(

`${this.namespace}:products`,

JSON.stringify(newProducts),

);

}

async clearProducts() {

await AsyncStorage.removeItem(`${this.namespace}:products`);

}

}

const doShopping = async () => {

const shoppingCartA = new ShoppingCartStorage('shoppingCartA');

const shoppingCartB = new ShoppingCartStorage('shoppingCartB');

await shoppingCartA.addProduct('chips');

await shoppingCartA.addProduct('soda');

await shoppingCartB.addProduct('milk');

const productsA = await shoppingCartA.getProducts();

const productsB = await shoppingCartB.getProducts();

console.log(productsA, productsB);

await shoppingCartA.clearProducts();

await shoppingCartB.clearProducts();

};

doShopping();

Because AsyncStorage keys are global, it is usually a good idea to add a *namespace* for the keys. In this context, the namespace is just a prefix we provide for the storage abstraction's keys. Using the namespace prevents the storage's keys from colliding with other AsyncStorage keys. In this example, the namespace is defined as the constructor's argument and we are using the namespace:key format for the keys.

We can add an item to the storage using the [AsyncStorage.setItem](https://react-native-async-storage.github.io/async-storage/docs/api#setitem) method. The first argument of the method is the item's key and the second argument its value. The value *must be a string*, so we need to serialize non-string values as we did with the JSON.stringify method. The [AsyncStorage.getItem](https://react-native-async-storage.github.io/async-storage/docs/api/#getitem) method can be used to get an item from the storage. The argument of the method is the item's key, of which value will be resolved. The [AsyncStorage.removeItem](https://react-native-async-storage.github.io/async-storage/docs/api/#removeitem) method can be used to remove the item with the provided key from the storage.

**NB:** [SecureStore](https://docs.expo.dev/versions/latest/sdk/securestore/) is similar persisted storage as the AsyncStorage but it encrypts the stored data. This makes it more suitable for storing more sensitive data such as the user's credit card number.