Software Engineer Coding Challenge

PREPARED FOR

Doug Sherman

Stratovan

PREPARED BY

Mike Kaufman

mikekaufman4@gmail.com

EXECUTIVE SUMMARY

MAR 31, 2021

Doug Sherman
Stratovan

202 Cousteau Pl #115,
Davis, CA 95618

Dear Doug Sherman,

Re: Enclosed Smart Kitchen SDK Proposal

Please find enclosed my detailed software proposal for your kind consideration. The challenge is somewhat open ended on various parts of the design so I've had to make various assumptions:

- By "integrates all the smart devices" I take this to mean combines whole, all the smart devices in the kitchen through communicative connectivity.
- All smart devices must be connected to the internet.
- The SDK provides support to initialize network connection by a phone app.

Yours Truly,

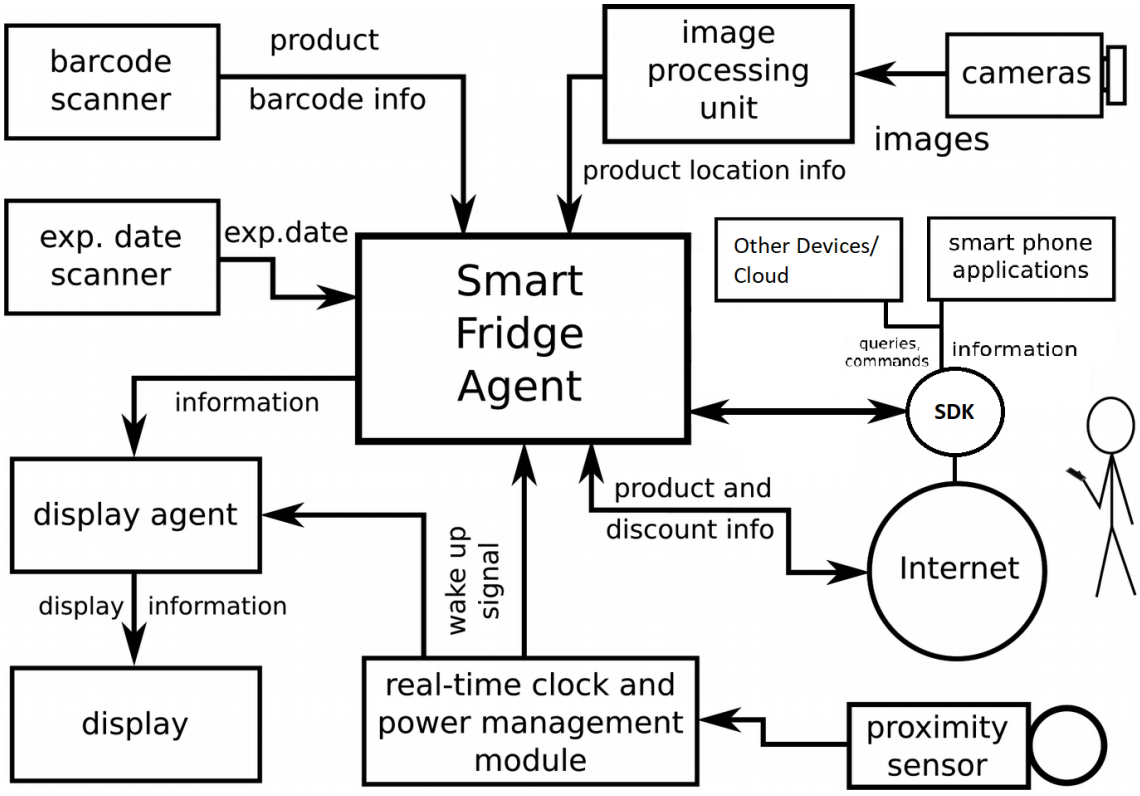
Mike Kaufman

A. Project Overview

The SDK uses inversion control design pattern

B. Diagrams

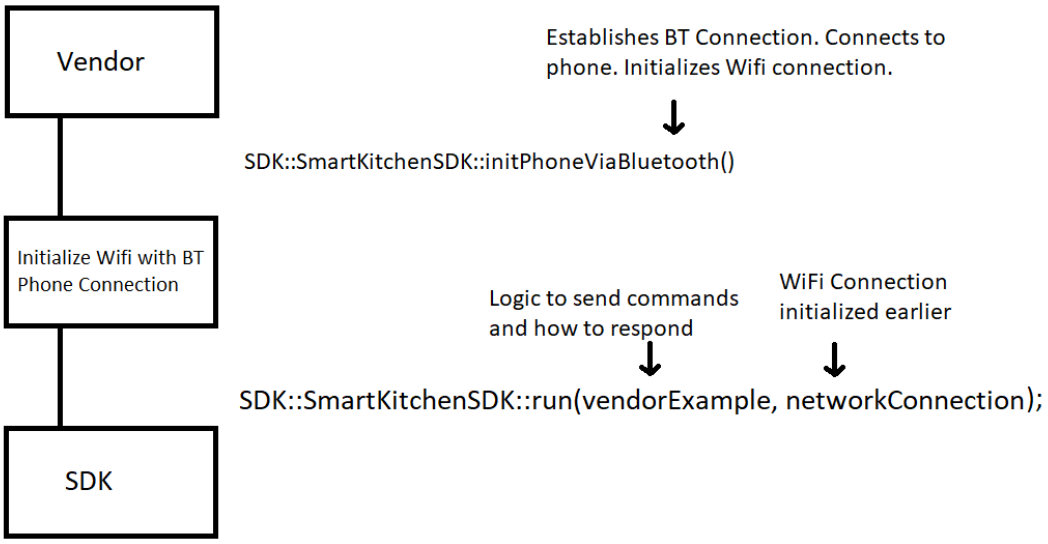
Vendor integration diagram



- By integrating the SDK into a vendor's product the following are capable: communication with other devices, smart phone control, and cloud services.

SDK diagram

```
void Vendor::readCommand(std::string) //required
// Add additional vendor logic ...
/*
  writeCommand(kargs), sendVideoToSDK(kargs), commandIntepreter(kargs)
 */
```




```
Reasons checkStream(std::stringstream &aStream) // Check aStream is valid, Safe, etc.
void processCommand(std::string)                // Interpret command into CoR, thread factory pool, etc
void broadcastCommand(std::string)              // send command to all devices by Secure Message Router
void log(std::string vendorID, Reasons reason, NetworkConnection networkconnection)
```

- Vendors first establish a connection to WiFi using a mobile phone, then run the SDK.
- Vendors can only read/ write to the command std::stringstream object due to inversion of control pattern for SDK. This makes the SDK safer because vendors do not have direct access to any components, the SDK can check if the command is: valid, malicious, etc, before continuing. If the command is found to be invalid, run will log the error then stop running and return to main. The vendor can do some logic before running ‘run’ again.
- The SDK can respond to specific commands (Example: cloudFunctions, http GET/POST/PUT) into an SDK users account for a mobile/ web app.

C. Requirements

Questions 1-5

Tasks
1
1.1 - What kitchen devices are manageable by your SDK?

Sinks	Outlets	Dishwashers	Food Thermometers
Fridges	Speakers	Stoves	Computers
Microwaves	Lights	Small Appliances	Etc...

Kitchen devices that connect to the SDK are essentially 3rd party vendor products that want to communicate with other items in the kitchen: this may be a speaker to announce the turkey is at ServSafe® temperature, or perhaps a fridge has an interior camera feed available to a user via their phone.

- Essentially this the manufacturer of the device wants to have smart capability without having to invest in creating their own system.
- The device must have internet capability.

1.1 - Pick one such device and define how it's vendors would use your SDK to connect to your smart kitchen.

Suppose it is a fridge.

It may have internal cameras, food inventory + food expiration dates, barcode scanner, remote control for TV front panel screen, logs, humidity/temperature alerts, diagnostic reporting, etc.

- An interface is required for tcp data like JSON, but also udp data like internal cameras.
- Perhaps they may want persistent data to be stored in a datastore in the cloud.

1.2 - What devices are not?

Devices that would not need to use this particular SDK would be smart devices with their own systems, like Alexa, iRobot, etc.

For systems like Alexa and iRobot, this service can be expanded to communicate with these services through the cloud.

2 - What aspects of your system require more security, and how would you approach that?

Security issues that may arise are:

Stealing/ hacking access to a cloud account, which would allow a malicious user to access cameras, turn on and off devices, set alarms, damage speakers, turn on dangerous devices like stoves, blenders, microwaves, toasters, space heaters, etc.

- **Two Factor Authentication**
  - Use a user's phone to accept access from unrecognized devices.

- **Authentication/Authorization Enforcer pattern**
  - Used to manage and delegate authentication/ authorization processes
- **Secure Session Manager**
  - Securely centralized session information handling
- **Credential Synchroniser pattern**
  - Securely synchronizes credentials and principals across multiple applications using Identity provisioning

Perhaps, a malicious company could monitor other devices activity in the home by microphones, cameras, log activity and sell it or worse.

- **Inversion of control (IoC) design pattern**
  - Vendor code does not have direct access to the functionality of the SDK.
- **Intercepting Validator pattern**
  - Performs security validation for input data
- **Obfuscated Transfer Object pattern**
  - Protect data passed around in transfer objects and between application tiers
- **Secure Logger pattern**
  - Log sensitive data securely ensuring tamper-proof storage
- **Secure Message Router pattern**
  - Provides secure XML communication. It is a security intermediary component that applies message-level security delivering messages to multiple recipients where the intended recipient would be able to access only the required portion of the message and remaining message fragments are confidential.

3

3.1 - What decisions are you making for the user-friendliness of your SDK?

- For Integrating a vendor's product with the SDK, **vendors only need to work with reading and writing command data.**
- If there is any error with a command it will be logged and not broadcasted into the system, removing the risk of hard to find bugs. In other words, **bugs will be isolated to the device** and the reason for the error will be logged. This is better than if the SDK shared the bugged command with other devices, causing undefined behavior.

3.2 - How do these encourage adoption by the 3rd party vendors of kitchen devices?

Easy set up and integration, access to a smart ad hoc system.

- Vendors only need to worry about how to talk to the SDK.
- SDK takes care of wireless network credential initialization via bluetooth or WiFi mobile phone connection.
- SDK talks to databases, datastore, cloud-functions, for you.
- SDK handles multithreading of different services for different commands like: video, https, tcp/udp, etc.
- Secure message routing and logging built in.
- Secure user authentication/ authorization for control by mobile/ web devices.
- Access to the Smart Kitchen System: communicate and connect with different devices

4 - How do you ensure 3rd party vendors are “correctly” using your SDK?

A bug in a vendor’s code can cause undefined behavior.

- **Error detection/correction pattern**
  - Deduce errors and potentially correct them to guarantee correct exchange or storage.
- **Check-pointed system pattern**
  - Allows for the system to recover when a component fails
- **Standby Design Pattern**

    - Provide fall back so the device can resume after failing. Ex: Internet is down, but device sends a request
- **Replicated System Pattern**
    - Adds redundancy for load balancing to decrease chance of lost availability of service. Ex: N devices send requests at the same time.

Behavioral:

- Connection information with the user's phone is isolated inside SDK.
- Commands passed between devices are isolated inside SDK.
- Commands written are validated before broadcasted through the SDK.

5 - What aspects of the SDK must be standardized across all the devices?

The standardized aspects of the SDK are:

- **void Vendor::readCommand(std::string aCommand)**
    - Reads a command for the vendor to access.
- **std::stringstream Vendor::writeCommand()**
    - Writes a command from the vendor the SDK to validate before broadcasting to all devices.
- **SDK::SmartKitchenSDK::initPhoneViaBluetooth()**
    - Initialized WiFi network by Bluetooth or Wifi connection with a mobile phone. Saved credentials in SDK datastore.
- **SDK::SmartKitchenSDK::run(vendorExample, networkConnection);**
    - This allows for the SDK to be isolated from the vendor. The SDK would be included as binary files, no source code would be public aside from the interface. Run initializes the RealSmartKitchenSDK that will have all the core logic of the SDK.

The class I choose to explain what tests need to be written is SmartKitchenSDK.

- initPhoneViaBluetooth
    - Connects to a phone
    - Gets correct WiFi credential selection
- run
    - Exits for correct reasons: syntaxError, badCommand, authExpired, badAuth, offline, standby, notreachable, vendor forced exit
    - Writes/ reads/broadcasts correct command:
    - correct secure Message Router protocol (ie no messages sent do a device can be read by another), correct Intercepting Validator protocol (ie no bugs are broadcasted, correct command written, reads correct command, valid stringstream after stress test of all of these functions.
    - processCommand:
        - performs correct https function (POST/PUT/DELETE) to datastore
        - correct thread pooling for video streaming, downloading/ uploading udp/tcp,
    - Log: logs correct formatting for all reasons/ network issues.