**Ice Cream Server**

Created: 02/22/21 by Tom Lever

Updated: 02/22/21 by Tom Lever

**1: Ice Cream Server**

I am presenting Ice Cream Server, to share a significant project that speaks to my passion and skill for understanding, designing, and creating software systems. Presenting Ice Cream Server will allow you to gain insight into my software development skills.

What is Ice Cream Server? Personally, I think it’s a great pun for software engineers. But for our purposes, Ice Cream Server is both the name of a Java project that I created, and an abstract object representing functionality within a larger Ice Cream System. So actually, I want to start by considering Ice Cream System.

**2: Ice Cream System**

What is Ice Cream System? Ice Cream System is a system that allows a Requester to submit ingredient search parameters and receive all products with those ingredients. We’ll go into what Requester, submit, search parameter, ingredient, receive, and product mean later.

**3: System Description: Context: DataBridge**

Let’s start off with a formal Ice Cream System Description. First, I’d like to share my understanding of DataBridge, and see if it fits. Per our discussions yesterday, it seems that DataBridge is a server-like system that interfaces either with a client controlled by a Requester or with an autonomous client. DataBridge also interfaces with the Digital Identity Team’s data lake of attributes of individuals, retrieves audiences of attributes from the data lake, and delivers those audiences to the client. Does this seem right?

**4: System Description: Opportunity Ice Cream System will Address**

The next part of our Ice Cream System Description is a description of the business opportunity that Ice Cream System will address.

Developing Ice Cream System provides me with foundational skills that will be useful in developing DataBridge functionality to listen for client requests, receive requests, interpret requests, complete actions based on requests, and provide responses to clients based on requests.

My intended outcomes of developing Ice Cream System are:

* To learn as much as possible about design, creation, and use of client-server systems,
* To understand what it means to serve beautiful React web apps on a port, and
* To support Merkle in developing the DataBridge system that interacts with clients and a data lake.

**5: Team**

Before describing what Ice Cream System will do in greater detail, I would like to tell you about the team. There’s me, and there’s Chip Staples. Chip is the Owner of Blue Ridge Software Solutions (for whom I work). Chip is also my mentor in Software Engineering, and a friend from the theater world. Regarding Ice Cream System, I am the sole developer. Chip and I design the Ice Cream System together. We complete code reviews and pulls into a development branch together. Chip is very familiar with industry standards, system design and decomposition, and coding concepts. Chip is very precise and exacting. We are refocusing our energy on a contract we have to develop music-literacy and audio-transcription applications.

**6: System Description: What Ice Cream System will Do**

At this point, I want to return to our System Description, and describe what Ice Cream System will do. There’s a Requester outside the Ice Cream System. The Requester can be either a human, or an application. The Requester submits Ingredient search parameters through a Requester and Ice-Cream System Interface to the Ice Cream System. The Requester receives from the Ice Cream System either the Matching products or a Clarification. An example of a Clarification is, “No products match the specified search parameters”. The Ice Cream System encapsulates a Client, a Server, and a Client-Server API, which we’ll talk about later. Hence, Ice Cream System is a Client-Server System.

**7: System Description: Iterations of Development**

Lizzie and I talked yesterday about how you iterate in small increments toward a minimum viable product. Here I’d like to present the Iterations of Development for Ice Cream System. First, the minimum viable functionality of the Ice Cream System will be to provide products or clarifications to the Requester, based on ingredient search parameters. The second Iteration of Development will be to extend the Ice Cream System to providing products or clarifications based on non-ingredient search parameters, like the sourcing value “Fairtrade”. After the third Iteration of Development, the Ice Cream System will provide to the Requester, based on signals from the Requester, additional mechanisms to Create Products in an Ice Cream Database, Update Products in the Ice Cream Database, or Delete Products from the Ice Cream Database.

**8. Requirements: Use-Case Description: Request Products by Ingredients**

I put together a “Request Products by Ingredients” Use Case Description for the Requester submitting ingredient search parameters and receiving Matching products or a Clarification.

Per the U.S. General Services Administration, “A use case is a written description of how users will perform tasks… It outlines, from a user’s point of view, a system’s behavior as it responds to a request”.

Per Oracle, “A use case describes how the system should respond to a request from [a] primary actor”.

**9. Requirements: Use-Case Description: Request Products by Ingredients: Use-Case Name**

The Use-Case Name is “Request Products by Ingredients”.

**10. Requirements: Use-Case Description: Request Products by Ingredients: Subject Area**

And the Subject Area is Ice Cream Server.

**11. Requirements: Use-Case Description: Request Products by Ingredients: TBE**

The Triggering Business Event of this Use Case is the Requester signaling a desire to request ice cream products matching ingredient search parameters. After signaling, the Requester might receive a web app to select and submit ingredients.

**12. Requirements: Use-Case Description: Request Products by Ingredients: Actor**

The Actor is the Requester.

**13. Requirements: Use-Case Description: Request Products by Ingredients: Overview**

The Overview of this Use Case is that a Requester requests ice cream products matching ingredient search parameters and receives those products.

**14. Requirements: Use-Case Description: Request Products by Ingredients: Preconditions**

Are there any preconditions for this Use Case to begin, such as other Use Cases completing successfully? There are none.

**15. Requirements: Use-Case Description: Request Products by Ingredients: TO**

The Termination Outcome of this Use Case is that a Requester receives the products matching the ingredient search parameters, or a clarification. Another Clarification might be that the Ice Cream Database is not available.

**16. Requirements: Use-Case Description: Request Products by Ingredients: CATO**

The Use Case always results in the Termination Outcome; there are no Conditions Affecting the Termination Outcome.

**17. Requirements: Use-Case Description: Request Products by Ingredients: Main Flow**

Step by step, what happens between the Requester signaling and the system providing Matching products? Well,

1. A Requester signals a desire to request ice cream products matching ingredient search parameters.
2. The system provides to the Requester a mechanism to request ice cream products matching those parameters.
3. The Requesters specifies no, some, or all ingredient search parameters.
4. The Requester signals that a request with the specified ingredient search parameters should be sent.
5. The system sends the request.
6. And the system provides to the Requester the products matching the ingredient search parameters.

**18. Requirements: Use-Case Description: Request Products by Ingredients: AF**

Alternatively, the system provides to the Requester a Clarification as to why no products are provided.

**19. Requirements: Use-Case Description: Request Products by Ingredients: Associations**

There are no associations between this use case and any other use case.

**20. Requirements: Use-Case Description: Request Products by Ingredients: TT**

This use case is traceable to the Requester and System Interface Design, which among other things defines Clarification.

**21. Requirements: Use-Case Description: Request Products by Ingredients: Inputs**

The Requester provides as input to the Ice Cream System ingredient search parameters.

**22. Requirements: Use-Case Description: Request Products by Ingredients: Outputs**

The system provides as outputs a mechanism to request ice cream products matching ingredient search parameters, and the actual Matching products, or a Clarification.

**23. Requirements: Use-Case Description: Request Products by Ingredients: UCV**

This is our first version of this Use-Case Description, …

**24. Requirements: Use-Case Description: Request Products by Ingredients: Assumptions**

I can’t think of anything we missed, .and …

**25. Requirements: Use-Case Description: Request Products by Ingredients: Notes**

I don’t think the software developers need to be aware of anything else right now.

That was a lot of information.

I invite us to pause.

Do you have any thoughts or questions?

**26. Requirements: Requester and Ice Cream System Interface Design**

Next up, I’d like to present my Requester and Ice Cream System Interface Design.

The Requester and System Interface is the interface through which the Requester and Ice Cream System interact. The interface is defined by the inputs and outputs that pass through it.

1. The inputs from the Requester are the ingredient search parameters.
   1. The ingredient search parameters are, simply, chosen ingredients.
      1. An ingredient is an item from any of the lists of ingredients for all products.
2. One output to the Requester is a set of Matching products.
   1. The Matching products are one or more products.
      1. A product consists of:
         1. A name
         2. A path to an image file of the closed product
         3. A path to an image file of the open product
         4. A description
         5. A story, which is an expression that encourages people to buy the product
         6. A list of sourcing values (like Fairtrade, or Non-GMO, or Cage-Free Eggs)
         7. A list of ingredients
         8. A list of allergens
         9. A list of dietary certifications (like Kosher)
         10. A product ID, which is a unique numeric label identifying the product
3. The other output to the Requester is a Clarification.
   1. A Clarification consists of a clarification value.
      1. A Clarification value can be “Ice Cream Database is not available”, or “No products match specific search parameters”.

At this point, I’d like to take another pause, and summarize what we’ve discussed so far. So far, we’ve been studying the whole Ice Cream System, which is a system that allows a Requester to submit ingredient search parameters and receive all products with those ingredients. We’ve talked about the Requester, the actions of the Requester and Ice Cream System, search parameters, ingredients, and products, at a high level. We’ve discussed our System Description, Use-Case Description for “Request Products by Ingredients”, and Requester and Ice Cream System Interface Design, which all relate to the Ice Cream System as a whole.

I’m interested in proceeding by decomposing the Ice Cream System into the Ice Cream Client, the Ice Cream Server, and the Client-Server Application Programming Interface (or API).

**27. Client-Server Interface Requirements (Server Half)**

If we recall our diagram of the Ice Cream System, which I presented when discussing “What the Ice Cream System will Do”, we saw, through the encapsulating bubble of the Ice Cream System, an Ice Cream Client, an Ice Cream Server, and a Client-Server API.

Here is a diagram representing the Client-Server Interface Requirements. I say that this diagram represents the Server Half of the Client-Server Interface Requirements, because I’ve thought a lot about the Ice Cream Server, but haven’t thought really at all about the Ice Cream Client.

Search parameters pass through the Client-Server API on their way to the Ice Cream Server. Matching products or a Status pass through the Client-Server API on their way from the Ice Cream Server. A Status could be something like, “Zero products available: Request from Client contains invalid JSON”.

**28. Client-Server Interface Design (Server Perspective)**

Now that we know the server-half of the Client-Server Interface Requirements, we can create a Client-Server Interface Design (from the server’s perspective).

First, I’d like to make an analogy that a server is like a phone operator in an office, a port is like a desk, an endpoint is like a phone on that desk, and a message handler is like an employee that will answer the phone. By analogy, when making a phone call, I want to specify a type of phone call, the phone number of the operator, the extension of a desk, and the name of the phone on the desk. In computer-ese, I send search parameters by specifying an HTTP request method, a Universal Resource Identifier (or URI), a port number, and an endpoint.

The Client-Server Interface Design labels the phone on the desk or the endpoint, “/search-by-ingredients”.

The Client-Server Interface Design:

1. Names the incoming message to the server as “Search parameters”,
2. Defines what kind of object the incoming message must be, and
3. Defines the format of the incoming message.

The Client-Server Interface Design:

1. Names one outgoing message from the server as “Matching products”,
2. Defines what kind of object this outgoing message must be, and
3. Defines the format of thi a outgoing message.

The Client-Server Interface Design:

1. Names the other outgoing message from the server as “Status”,
2. Defines what kind of object this outgoing message must be, and
3. Defines the format of this outgoing message.

So, what are the definitions of the messages passing through the Client-Server API?

**29. Client-Server Interface Design: Incoming Message**

The Client-Server API Design defines the incoming message “Search parameters” as an object that:

1. is received by the POST HTTP request method,
2. has the key “search-parameters”, the delimiter “equals”, and a value that is an ingredients list, and
3. encodes the key and value in “urlencoded” format. If you’ve ever seen a white-space character represented as “%20”, that’s “urlencoded” format. “urlencoded” format is used to avoid special characters, like “&”, within the key and value. Ice Cream Server uses “&” after a key value pair as a delimiter before another key value pair.

So here is a schematic of the format of the incoming message “Search parameters”. The notation is Backus-Naur Form, which I use whenever I want to present a schematic of a format with definitions.

**30. Client-Server Interface Design: Outgoing Messages**

Let’s take a look at the outgoing messages from the server through the Client-Server API.

Let me start with the outgoing message “Status”.

The Client-Server API Design defines status as a byte array representing exactly what you see here, except that a value is substituted for the definition. A value could be the wording, “Zero products available: Ice Cream Database does not contain an ingredient specified in the client message”.

Proceeding to the outgoing message “Matching products”,

The Client-Server API Design defines the outgoing message “Matching products” as a byte array representing pretty much what you see here. Values are substituted for definitions, and newline characters are removed.

**31. Client-Server Interface Design: Product**

The Client-Server API Design defines a product as a byte array representing pretty much what you see here. Values are substituted for definitions, and most whitespace is removed.

At this point, I’d like to pause and summarize again.

So far, we’ve studied the whole Ice Cream System, which is a system that allows a Requester to submit ingredient search parameters and receive all products with those ingredients. We’ve decomposed the Ice Cream System into an Ice Cream Client, an Ice Cream Server, and a Client-Server API. We studied the incoming message to the server (named Search parameters), and the outgoing messages from the server (named Matching products and Status) The incoming message, Search parameters, and the outgoing messages, Matching products and Status, are required by the Client-Server Interface Requirements, and detailed in the Client-Server API Design.

At this point, I’d like to decompose once more. I’d like to decompose the Ice Cream Server.

**32. Ice Cream Server Decomposition**

This diagram presents the decomposition of Ice Cream Server into an Ice Cream Client Communication Subsystem and an Ice Cream Product Retrieval Subsystem.

The Ice Cream Client Communication Subsystem manages the resources of the server in listening for messages, works with received messages, and communicates with the ice cream product retrieval subsystem. The Ice Cream Client Communication Subsystem sends search parameters and receives ice cream products matching those search parameters.

The Ice Cream Product Retrieval Subsystem receives search parameters from the Ice Cream Client Communication Subsystem, and provides to the Ice Cream Client Communication Subsystem the ice cream products that match the search parameters.

After all of our design and decomposition, we have arrived at fairly well-defined input and output objects (so, Search parameters, Matching products, and Status), and relatively cohesive, loosely coupled subsystems (which are also objects) that manage communication and products retrieval. At this point, I’d like to think about software with you.

**33. Design Classes**

Design classes serve as a bridge between all the design and decomposition that we’ve been doing and software creation. They represent templates for objects. We can represent the Ice Cream Client Communication Subsystem, the Ice Cream Product Retrieval Subsystem, and the Search parameters, Matching products, and Status objects in software with design classes, which then will provide class templates to implement.

Before talking a little about implementation, I would like to introduce one last design: the Ice Cream Database Design. Products matching search parameters will be drawn from the Ice Cream Database.

**34. Ice Cream Database Design**

The Ice Cream Database consists of nine tables. The main table is a Products table. Each row in the table corresponds to one product. Each product has an id, a name, an image\_closed path, an image\_open path, a description, a story, and a productId. Ideally, all values are unique.

All ingredients are in a separate Ingredients table. All SourcingValues, AllergyInfos, and DietaryCertifications are also in their own tables. Considering the Ingredients table, each row corresponds to one ingredient. Each ingredient has an id and a value. Ideally, all values are unique.

To implement an SQL query to retrieve products matching ingredients, I started with the Ingredients table. I found all ID’s of ingredients matching the ingredient search parameters. So then the question becomes, how do we find all products with all of those ingredients?

So there’s an association table between products and ingredients. Each row represents a link between a product and an ingredient. Each link has an id, an index of product, and an index of ingredient. The values in the ID column are unique. In the index of product column, there are a lot of cells corresponding to a given product. For one product, each cell in the product column for that product corresponds to a unique ingredient. The indices in the index of ingredient column are not unique either, because multiple products can have the same ingredient.

So, when retrieving products matching ingredient search parameters, I find a relatively short list of ingredient ids and join in the associations between products and ingredients. At this point, I choose to look in the index of product column of the resulting table and find all products with a number of correspondences to ingredients equal to the number of ingredient search parameters. Once I have the list of product ids with a number of correspondences to ingredients equal to the number of ingredient search parameters, I join in the products table. The result is a table where each row represents a product matching all the ingredient search parameters.

To be honest, I haven’t yet joined arrays of sourcing values, allergy infos, or dietary certifications to the products, so the Ice Cream System really just provides half products to the Requester right now.

At this point, we have completed our review of the Ice Cream System, the Ingredient Search Parameters, Matching products, and Clarifications; the Client, Server, Client-Server API, Search parameters, Matching products, and Statuses; and the Ice Cream Client Communication Subsystem, Ice Cream Product Retrieval Subsystem, and Ice Cream Database. We have considered design classes that serve as a bridge from design to software development.

I think I’d like to end with some fun, implementation-oriented slides.

**35. Development Tools**

Here are all the development tools that I used during development of the Ice Cream System.

**36. Java Project Structure as of 02/21/21**

In case you’re curious, I’ve provided a screenshot of my Java project structure. I have a package of design classes at the top, and packages of server utilities, client communication utilities, and product utilities. I have a section for dependencies and specify dependencies and plugins in pom.xml.

**37. Execution**

Lastly, I’d like to present screenshots of submitting Search parameters from a local client, Postman, to Ice Cream Server running on an Amazon EC2 instance running Amazon Linux. The incoming message has key search-parameters and value {“ingredients”: [“cream”, “sugar”]}. The Ice Cream Server outputs all Matching products presently in Ice Cream Database in accordance with the Client-Server API Design (except that the products are half products right now).

**Acknowledgements**

I would like to conclude with some acknowledgements. I have immense gratitude for Chip. We work well together, given our love of understanding, precision, systems, software, and theater. He has invested countless hours over the past few months teaching me software-development concepts, introducing tools, guiding design of Ice Cream System, reviewing pull requests, and offering me work on React web apps. I am proud to have a working Ice Cream Server. I am confident I have the foundational networking and API-design skills that I need to be an effective member of Merkle’s DataBridge Team. I hope to transition into automatically designing and coding with only the scope of the document or object that I’m working on in mind, to be fastidious about coding in accordance with design documentation and decomposing classes and methods into subclasses and submethods, and to submit small commits and pull requests frequently.

I want to also thank all of you for your time, energy, interest, and consideration. I learned a lot about a growing, open-minded, cool company; a useful data lake and the DataBridge API; and about interviewing and development concepts and tools. I put together this presentation, which I’m proud of. I look forward to working with you. Thank you.