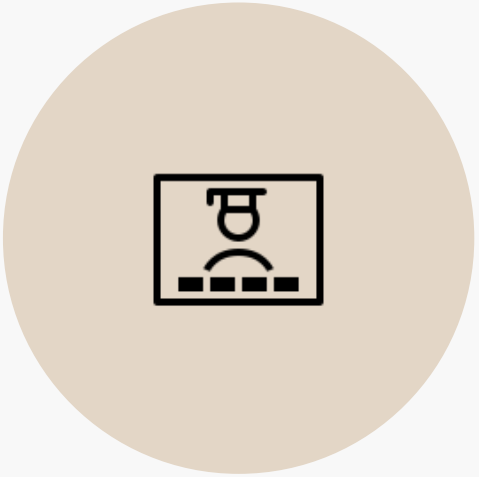
Web Services and Its Communication

09/07 - Web Service Dev & SOA

Adenda

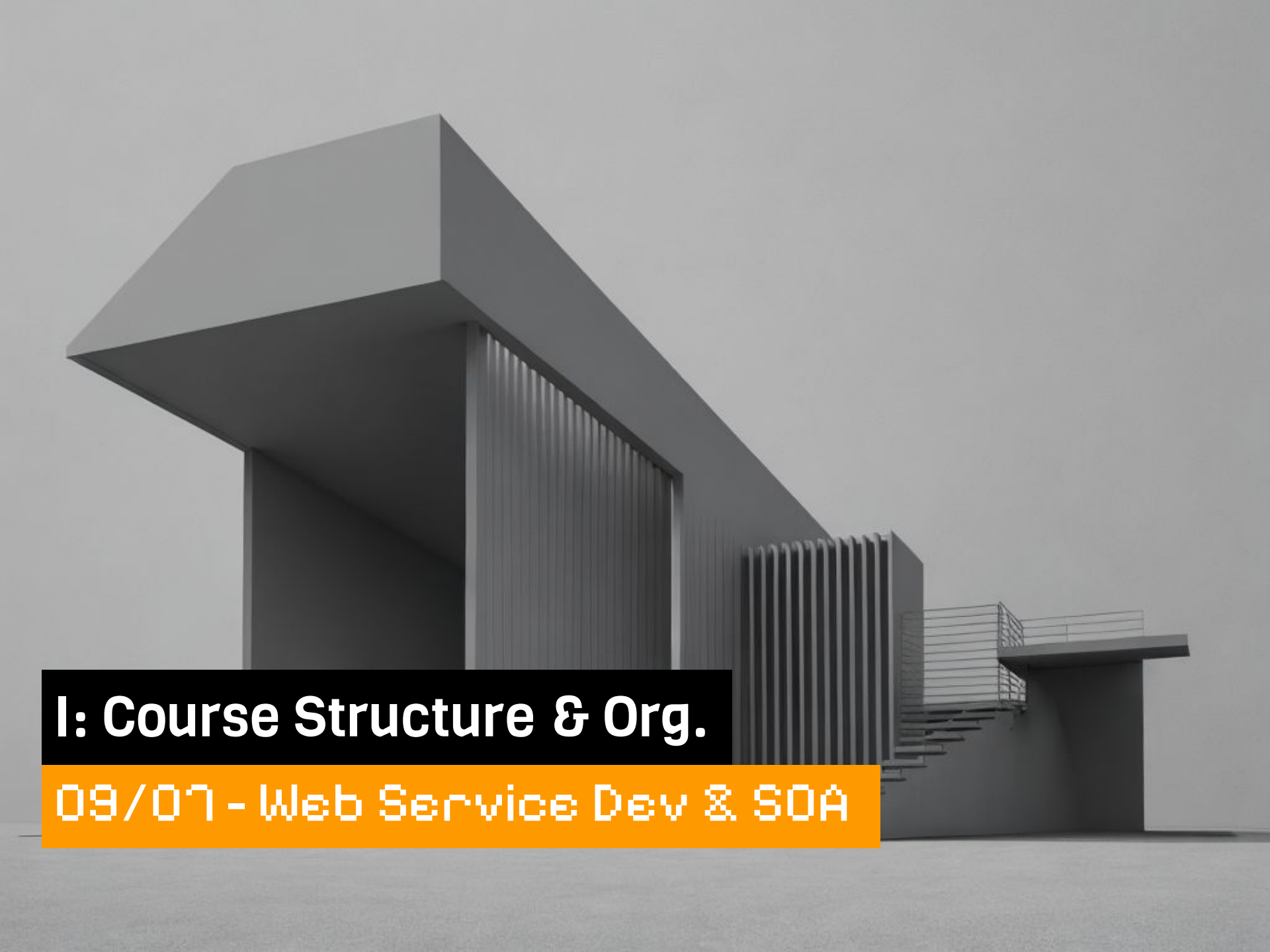
What's On the menu? - Week 2



I: Course Structure & Organisation

II: Basic of Web Services

III: Web Services Communication



I: Course Structure & Org.

09/07 - Web Service Dev & SOA

About Me

"Student" (for ~2 days)

Software Eng. (Int) BSc

College of Art, Media and Technology

Software Systems Eng. MSc

University College London

Widely-known for:
Telephone, Gandhi,
Christopher Nolan,
Chris Martin (Coldplay)

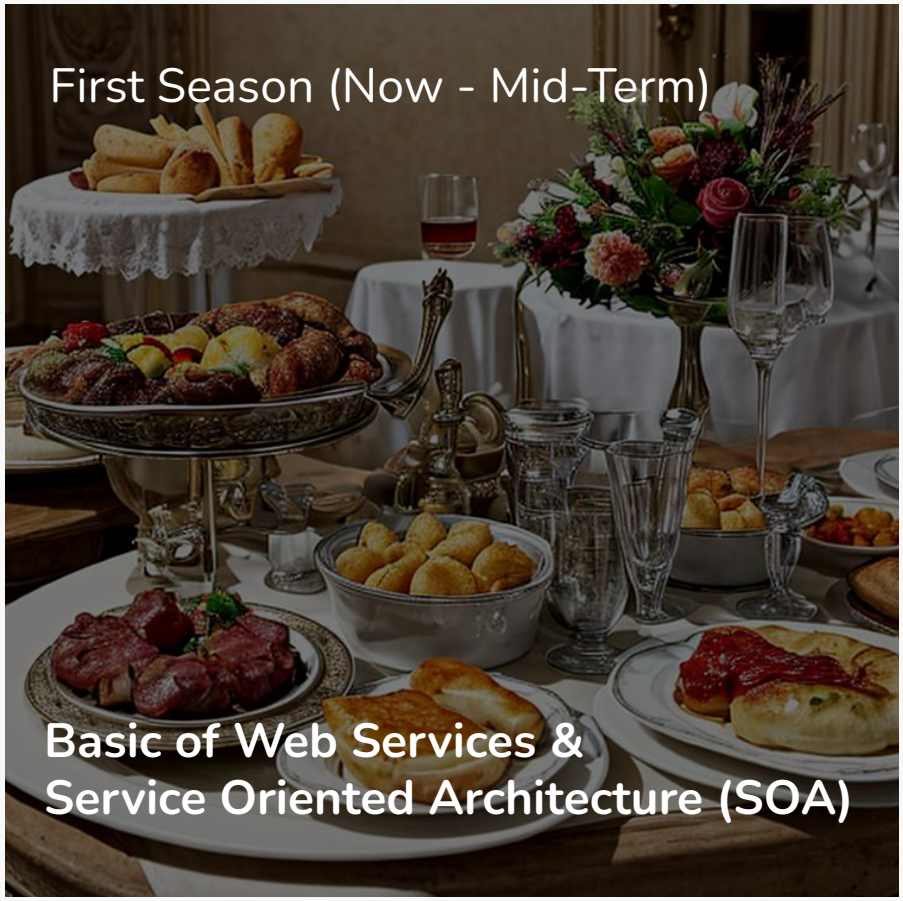
Computer Science PhD

University of Birmingham

Widely-known for:
DNA Structure,
The Lords of the Rings,
Peaky Blinders

Course Organisation

2 Seasons. Each with different flavours. ~5-6 Episodes/season



Class Organisation

For each episode (aka each week):

60

Mins. (max)

/w a group exercise (in-class) & discussions.

No homework.

Smart devices are allowed.

After the lecture.

A group exercise = Attendance record

Sent = Full Mark

A lecture slide = Attendance receipt

Class Assessments

Part I: Grinding

Goal: Prepare for exams.

Lecture for key components + group exercise each week.

Submit via email before the class ends.

Attendance record will be included in a **group project.**

Class Assessments

Part I: Grinding

Group project (30%) =
First Season (15%) +
Second Season (15%)

Group of **3** (based on the student's track).
1 system, **2** architectures
(SOA & MS).

Any programing languages/libraries are allowed.

Note: Projects details will be provided after submitting this week group exercise.

Class Assessments

Part I: Grinding

Group project will be assessed by **presentation** at the end of each season (Week 7 & Week 15).

The presentation includes:

- Slides
- Live demo

Presentation Duration: To be voted in **Week 4**.

Note: Presentation details will be provided after submitting this week group exercise.

Class Assessments

Part I: Grinding

Designed for Self-study outside class (i.e. 6 in 3-0-6)

Esp. for implementing a group project for **the live demo**. Some project requirements will **fail** without this.

To-do: Get familiar with `Docker & K8s`.

Hard to implement; earlier to do = better

(Relate to **Week 5** lecture,
may be too late to start impl. from there)

Class Assessments

Part I: Grinding

During self-study, you are encouraged to:

- Form **a study group** (outside your group project).
- Consult **additional materials** (including YouTube, other books, tutorials)
- Use **ChatGPT/Any LLMs** to test your understanding.
  - **Beware:** ChatGPT is bulls**t **(ref: 10.1007/s10676-024-09775-5)**

Class Assessments

Part II: Boss Fight

Individual Examinations (70%) =
First Season - Mini Boss (30%) +
Second Season - Final Boss (40%)

Goal: Use knowledge from exercises + project, design a system in a smaller scale app. Different apps per exams.

Hint: can be any apps in App/Play Store.

Class Assessments

Part II: Boss Fight

First Season covers:

- **Basic of Web Services**
- Service Oriented Architecture

Second Season covers:

- **Basic of Web Services**
- Microservices Architecture
- Cloud-Computing

TL;DR

Do this *consistently* to get A. For exams and a project.

Be Minimalist

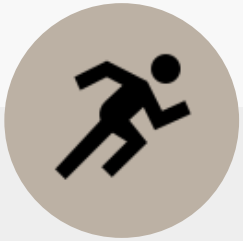
When design a system, do **just enough** to meet **scoring criteria** for A.

Be Specific

To the problem context. Both during exams and projects.

Reference when possible

No ref? Convince w/ an example.



Exercise, don't memorise

Can't remember? Practice more.

Birds of a feather

Stick together w/ your teammates when work & study

Note: Scoring criteria will be provided after submitting this week group exercise.

TL;DR

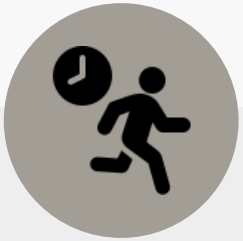
Do not do this during the course (please please please)

Skip a class

Will make **your teammate** suffer.

Get into coding late

The devil in the details. Coding can be **problematic.**



Be late

According to the course. **May entirely miss** the lecture.

Steal other group's ideas

Make it different somehow.

Contact

To:
suwichak.fu(at)kmitl.ac.th

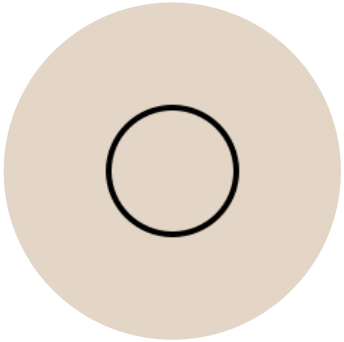
- Exercise/slide submissions.
- Course feedback and/or discussions.
- Appointment for in-person meeting (2 days in advance).

II: Basic of "Service"

09/07 - Web Service Dev & SOA

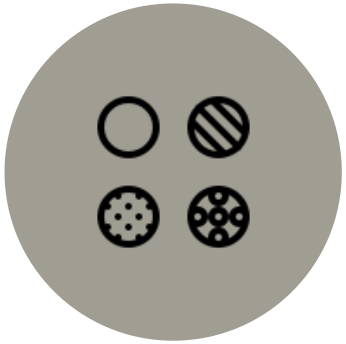
What is "Service"?

A software and hardware combination, aim to support **a business function**.



Atomic service
(Fine-grained)

- Independence to state of other service.



Composite service
(Coarse-grained)

- Consists of atomic or other composite service.

Note: A system can have both types of service (see running example).

Coupling

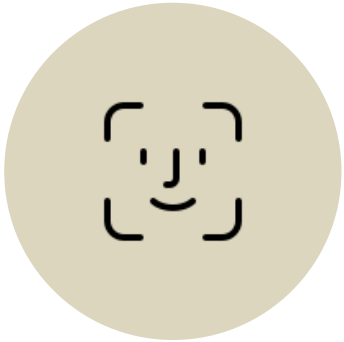
For scalability, each service should be **loosely coupled** from each other.

Coupling: A degree of **interdependence** between software modules/services.

Real-life examples (Coupling between User and his/her phone):

Loose: Do not exclusive to one user (e.g. Password sharing).

Tight: Cannot log in when wearing gloves, (but the chance of wearing gloves is slim.)



Tighter than the previous: Cannot log in when face is obstructed.

Running Example: Service

Milk's is a start-up restaurant, consists of the following service.

Front-end

For tableside **tablet** ordering. Self-serving here.

Log in

Membership for FB users. Can opt-out as guests.

Order Mgmt.

For **alter and/or cancel** orders.

Queuing Mgmt.

To consolidate cooking orders. **For the chefs.**

Running Example: Types of Service

Types of service.

Front-end

Composite; Login and ordering.

Log in

Atomic; one purpose only: To login.

Order Mgmt.

Composite; change or cancel the order.

Queuing Mgmt.

Composite; Manual & Auto cooking order merge.

Running Example: Coupling

Targeted degree of coupling.

Front-end

Loose; Broken tablet won't stop other services if **replacing with a spare**.

Log in

Somewhat Tight; No membership sales when **FB API is down**,

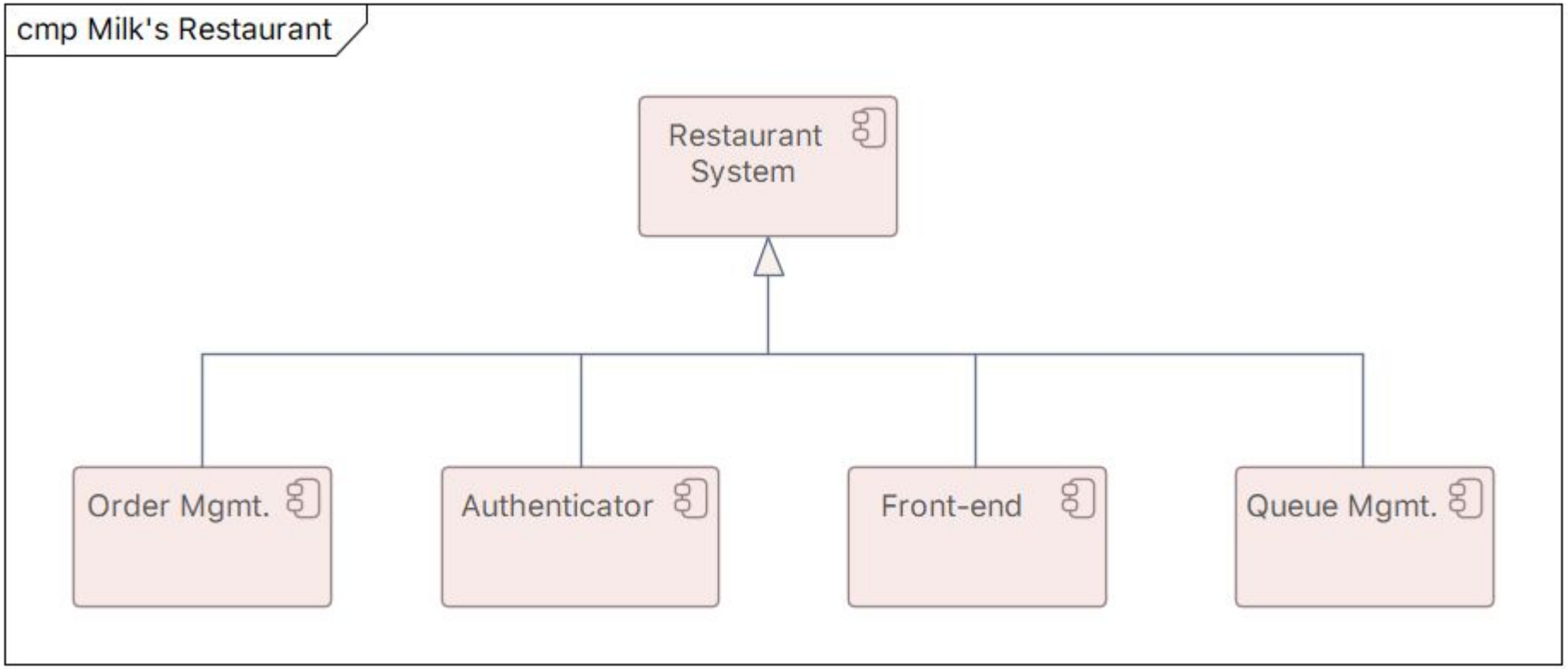
Order Mgmt.

Loose; Resetting a server **won't affected** other services.

Queue Mgmt.

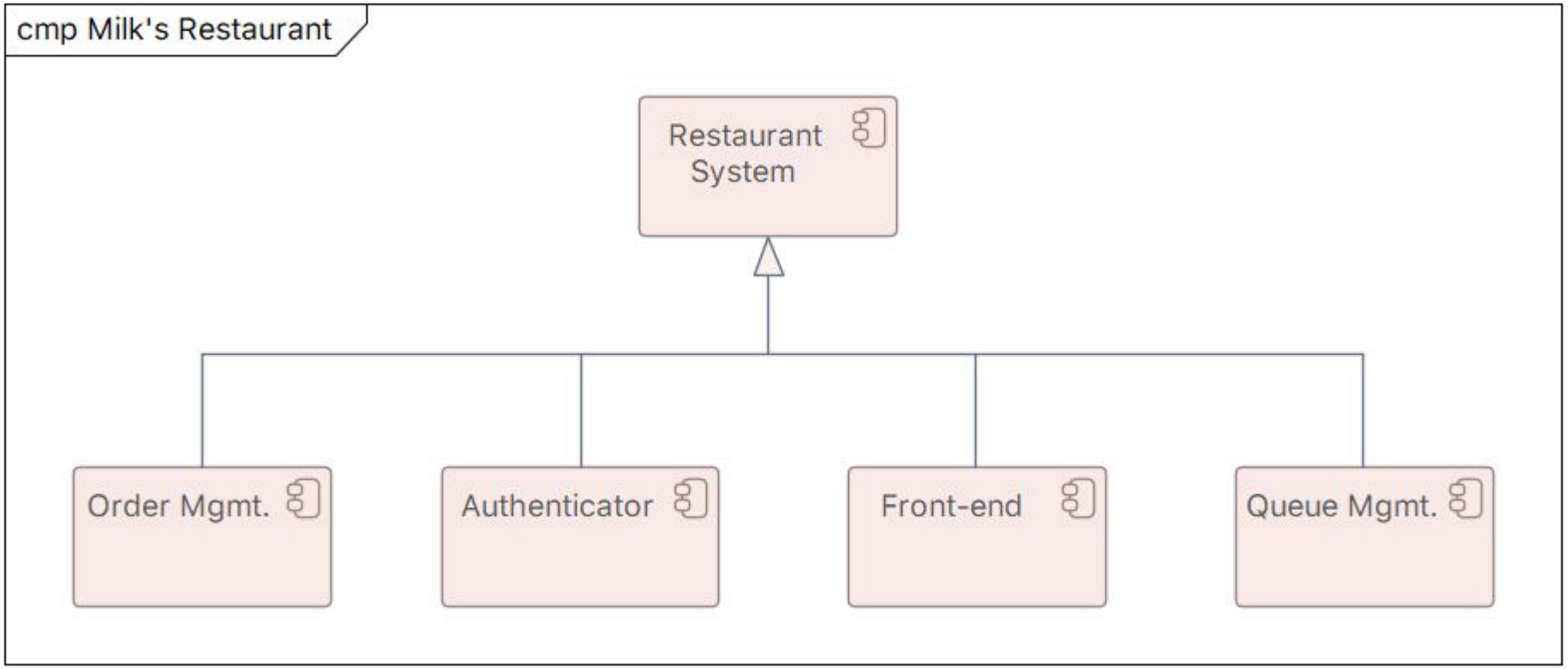
Loose; Resetting **won't affected** other services.

Running Example



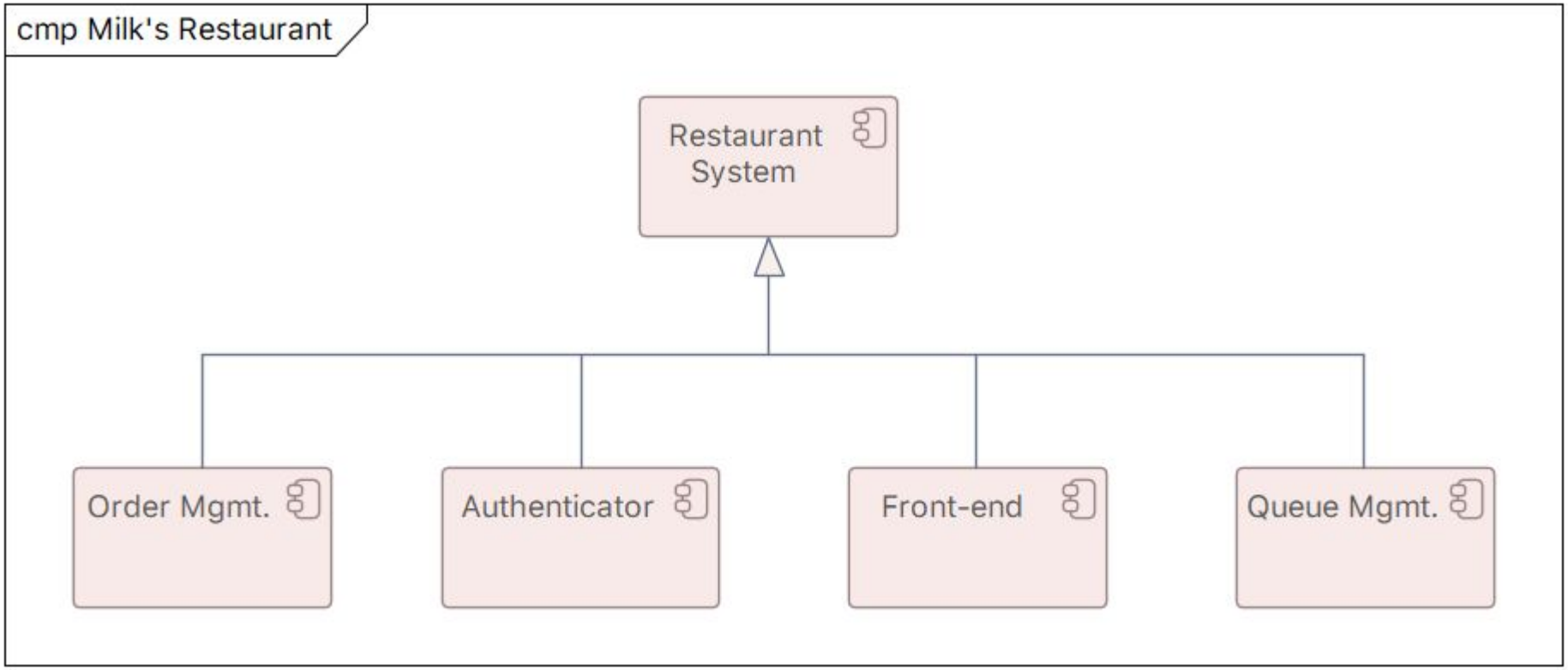
Running Example

Problem 1: How they communicate to each other?



Running Example

Problem 2: What if different services use different variables?



III: “Web” in Web Service

09/07 - Web Service Dev & SOA

REST API

Answer to Problem 1. **Commonly-used** communication between web services. Core principles are:

Statelessness.

(Next Slides)

Uniform Interface:
(for example:)

- GET for get info.
- POST to create.
- PUT to update.
- DELETE to delete.

Separation of Concerns.

(By design of SOA & MS)

Stateful vs Stateless

For scalability, one should aim for **stateless** architectures.

Stateful

- Request contains **limited** info.
- **Track & store** user's sessions.
- **Resource-intensive.**

Stateless

- Request contains **all** info to process.
- **Do not track** & store user's sessions.
- **Fewer** resource used.

Stateful vs Stateless

Cons of Stateless:

Stateful

- Ensure transaction integrity via **session data**.
- **Can have** offline support (from session data).
- **Minimal** requests.

Stateless

- Ensure transaction integrity via **requests.**
- **Limited** offline support (no session data).
- **Chatty** requests.

Note: A system can have both types of architectures (see running example).

Stateful vs Stateless

Pros of Stateless:

Stateful

- Complex to **scale**.
- Fault from central can **affect other services.** (From centralised sess. data)
- Session management causes **overheads.**

Stateless

- Easier to **scale.**
- Fault **Isolation**.
- **No** session overhead.

Stateful vs Stateless

Real-world examples:

Stateful Coffee Shop

- **Memorise** orders.
- Get coffee **before** pay.
- **Memorise** customers' payment.

Stateless Coffee Shop

- **Write down** order & name on a coffee cup.
- Pay **before** write down order.
- **Print** customers' receipt.

Stateful vs Stateless

Simple examples: During rush-hour

Stateful Coffee Shop

- **Too much order** to memorise.
- **Too much customers** to memorise.
- Some may cons for **a free coffee**.

Stateless Coffee Shop

- Keep calm & **write the orders down.**
- Keep calm & **print the receipts.**
- **Profits.**

Stateful vs Stateless

Simple examples: Encounter a twin (act as an imposter)

Stateful Coffee Shop

- **Figure out** whose orders.
- **Figure out** who's paid.

Stateless Coffee Shop

- **Don't care** as long as they have been paid**.**
- Twin can poison **each others.**

Asynchronous vs Synchronous

For better resource mgmt, one should aim for **asynchronous** requests.

Synchronous Request

- When send a request, **wait** for a response.
- **Waste** time to wait.
- Get a response first, then **proceed** to the next one.

Asynchronous Request

- When send a request, **proceed** to the next one.
- **No** need to wait.
- Will be **notified** when receives a response.

Asynchronous vs Synchronous

For better resource mgmt, one should aim for **asynchronous** requests.

Synchronous Request

- Easy to **code**.
- Support for **real-time** response**.**

Asynchronous Request

- More **complex** to code.
- Do not support **real-time** response.

Note: A system can have both types of requests (see running example).

Asynchronous vs Synchronous

Simple examples

Synch. Coffee Shop

- Order by order.
- **Easy** for a barista.
- **Wait** even just a bottle of water**.**
- **Can** change the order (if haven't done).

Asynch. Coffee Shop

- Multitask (Consolidated orders).
- **Challenging** for a barista.
- **Less** waiting time.
- **Cannot** alter/change the order.

JSON & XML

Answer to Problem 2. **Commonly-used** representation in REST API:

JSON

- **Newer than** XML (2000s).
- Widely-used **now**.
- Finds in **FB.com**.
- More **compact**.
- **Less** overhead.
- Array **friendly.**

XML

- **Older than** JSON (late 90s).
- Widely-used **then**, still use now.
- Finds iOS system **components**, Word **doc**.
- More **readable**.
- **More** overhead.
- **Wordy** for array.

JSON & XML

Answer to Problem 2. **Commonly-used** representation in REST API:

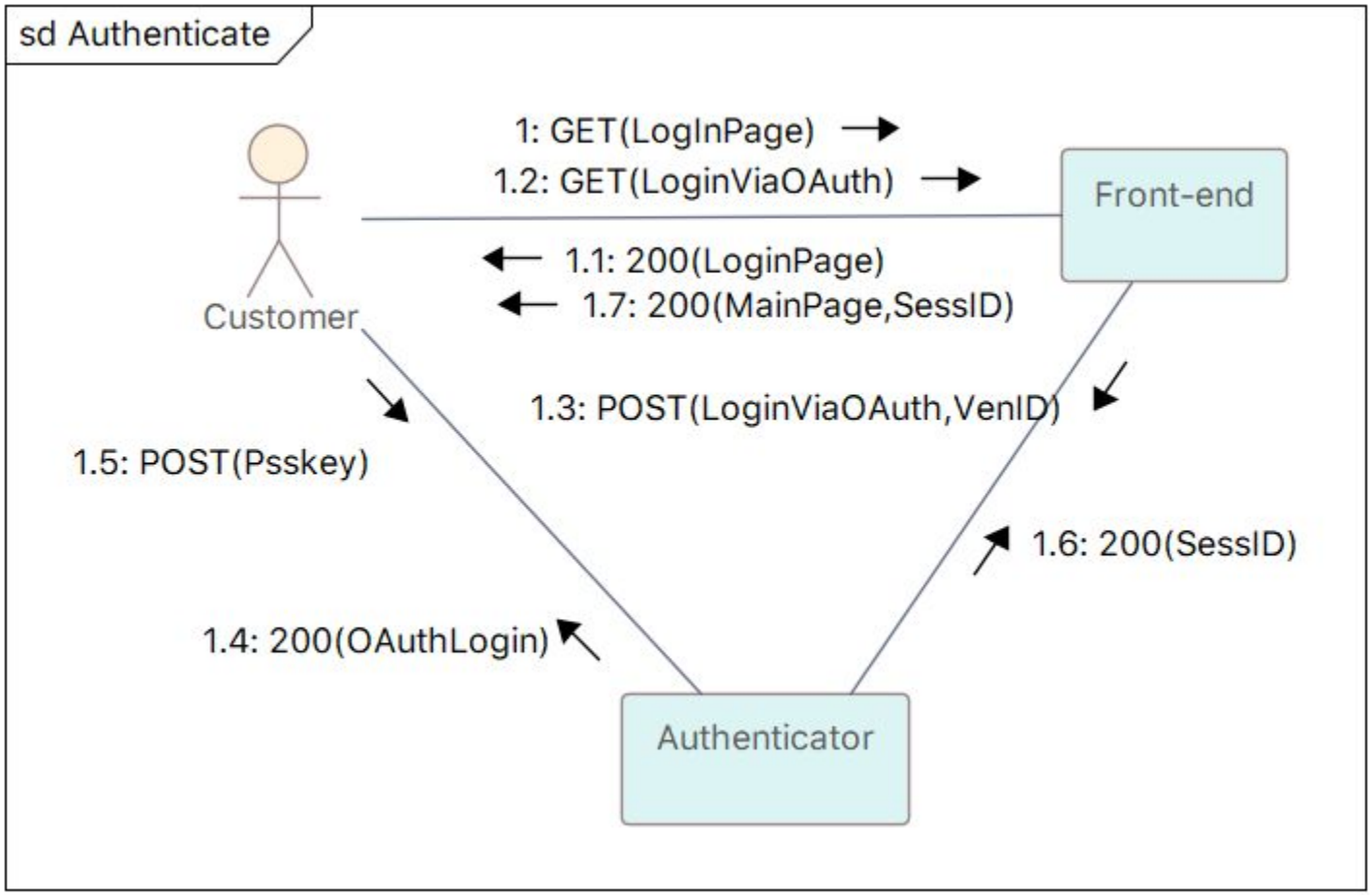
- **Better** than plain texts (e.g. tag parsing/mapping, check for the completeness of response)
- **Using both in many systems** (i.e. AJAX)
- XML is still in many legacy systems (i.e. banking, transportation systems.)

- Course Mantra: “**Old** does not mean **dead**, **new** does not mean **best**”

: Slipknot - All Out Life

Running Example

Authenticate :
@1.6 & 1.7: SessID for Stateless



Running Example

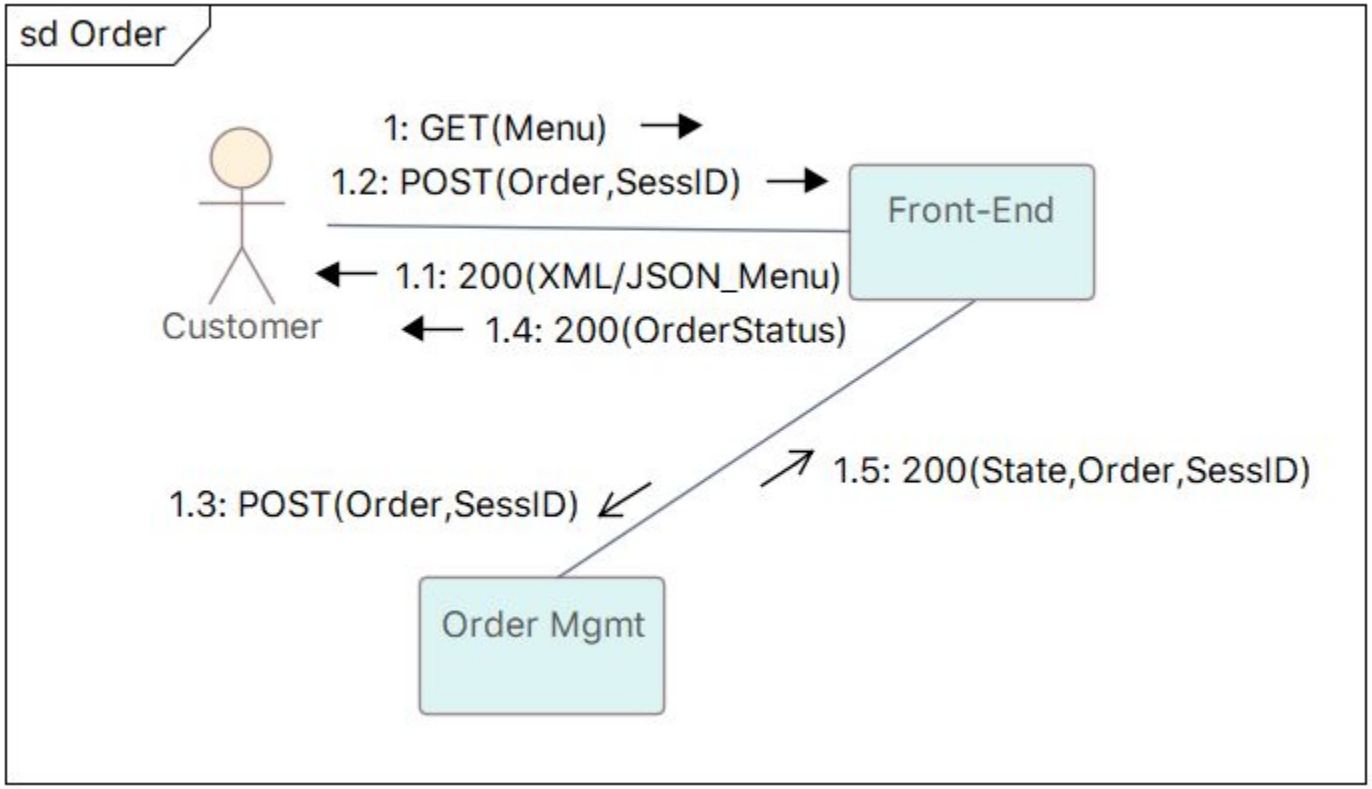
Authenticate :
@1.6 & 1.7: SessID for Stateless

- **Session ID (base64 - encode):**
  aWQ6IDAwMDEsICB1c2VyOiAiTWlsayIsICB
  GaXJzdE5hbWU6ICJTdXdpY2hhayIsICBMYX
  N0TmFtZTogIkZ1bmdwcmFzZXJ0a3VsIiwgIE
  V4cGlyYXRpb246IDE1MjUxMzI3OTk=

- **Session ID (base64 - decode):**
  id: 0001, user: "Milk", FirstName: "Suwichak",
  LastName: "Fungprasertkul", Expiration:
  1525132799

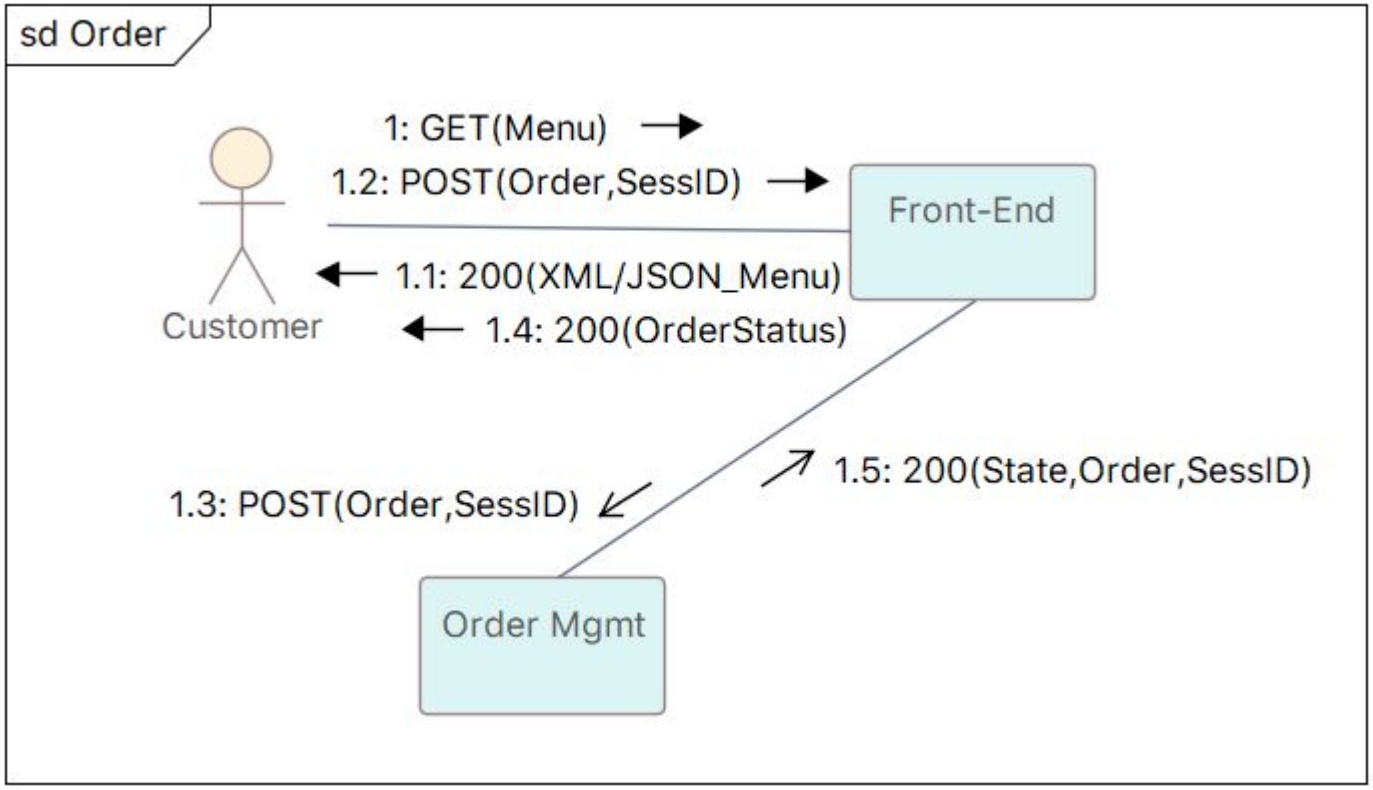
Running Example

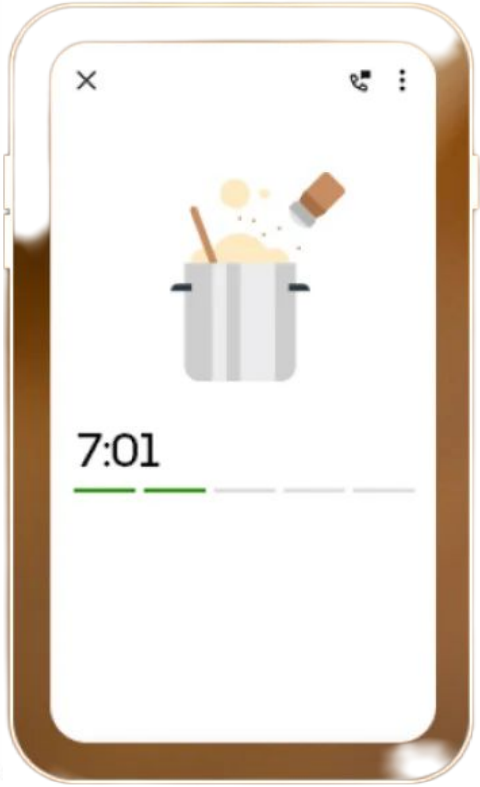
Order: @1.2: Use SessID for Stateless Front-End
@1.3: Asynchronous Request



Running Example

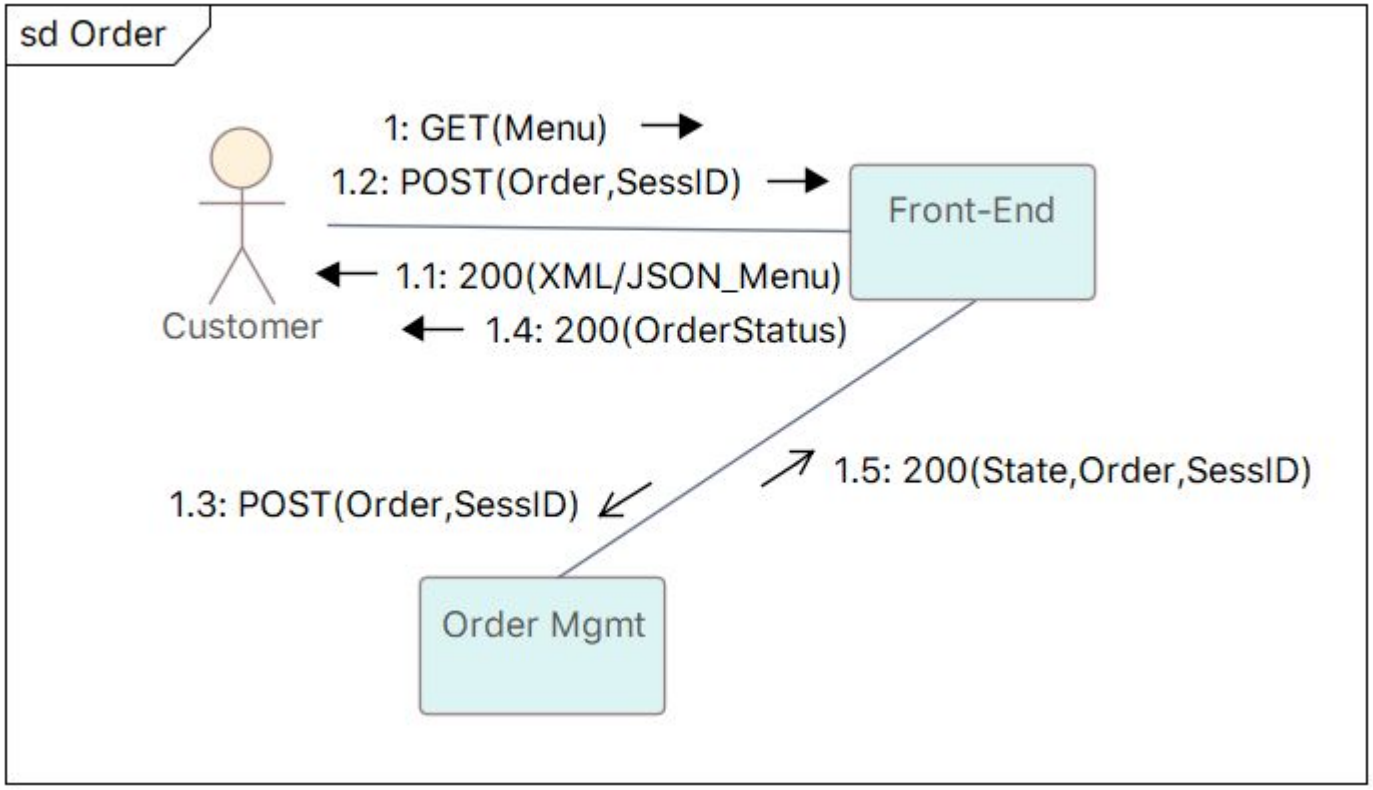
Order: @1.2: Use SessID for Stateless Front-End
@ 1.4 : “Preparing Order…”

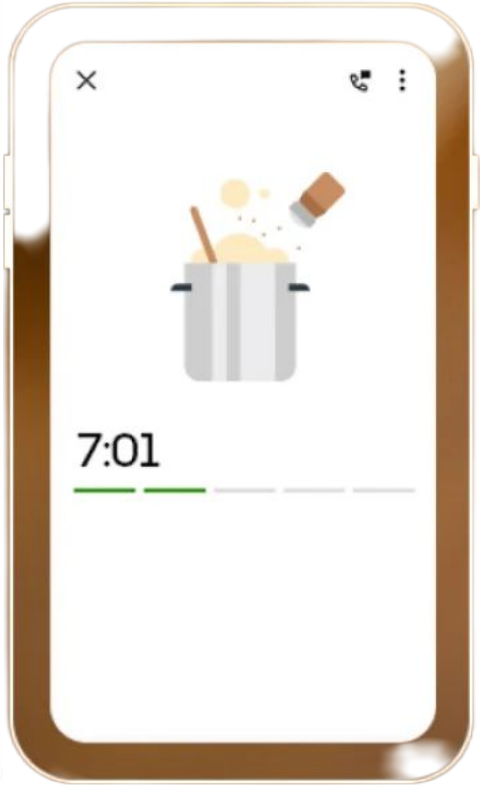




Running Example

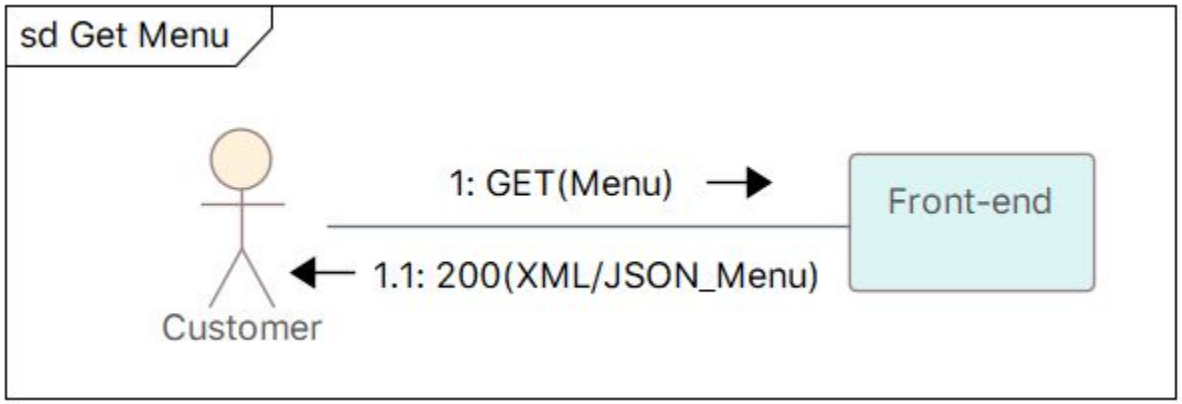
Order: @1.2: Use SessID for Stateless Front-End
@ 1.5 : Asynchronous Response





Running Example

See Menu



JSON & XML

Example:

```
{"restaurant":{"name":"Milk's",
"address":"Lat Krabang, Bangkok,
Thailand","postcode":"10520"},"menu":
{"food":{"name":"New York Cheeseburger",
"price":"250 THB"},"drink":
[{"name":"Cherry Coke","price":"35 THB"},
{"name":"Vanilla Milkshake","price":"65
THB"}]}}
```

```
<restaurant><name>Milk's</
name><address>Lat Krabang, Bangkok,
Thailand</address><postcode>10520</
postcode></
restaurant><menu><food><name>New York
Cheeseburger</name><price>250 THB</
price></food><drinks><drink><name>Cherry
Coke</name><price>35 THB</price></
drink><drink><name>Vanilla Milkshake</
name><price>65 THB</price></drink></
drinks></menu>
```

JSON

- **Word count** (excl. spaces): 241
- Need to look up what the second last "}" is for.

XML

- **Word count** (excl. spaces): 340
- From XML, we know the second last "}" is </drinks>


Group Exercise
09/07 - Web Service Dev & SOA

Group Exercise -Week 2

1. Form a group of 3 according to the students' track. Write down your group name, track & team members (feel free to 'cc' team members).
2. For Metaverse track:
   a. What services do you think are essential to a virtual banking system (via a component diagram)?

   For IoT track:
   b. What services do you think are essential to a smart meter system (via a component diagram)?
3. Provide an interaction of between services of your system (via a communication diagram). The interaction must include one REST protocol.
4. Provide an example of a data structure that can gathered from the system in XML or JSON format. The example must include at least one rationale of why XML or JSON has been chosen.

Send To:
suwichak.fu(at)kmitl.ac.th

Subject:
[6622][(Team Name)][IoT/Metaverse] Group Exercise Submission
Example:
[6622][Saltburn][Metaverse] Group Exercise Submission

Group Exercise -Week 2

1. Form a group of 3 according to the students' track. Write down your group name, track & team members (feel free to 'cc' team members).
2. For Metaverse track:
   a. What services do you think are essential to a virtual banking system (via a component diagram)?

   For IoT track:
   b. What services do you think are essential to a smart meter system (via a component diagram)?
3. Provide an interaction of between services of your system (via a communication diagram). The interaction must include one REST protocol.
4. Provide an example of a data structure that can gathered from the system in XML or JSON format. The example must include at least one rationale of why XML or JSON has been chosen.

Send To:
suwichak.fu(at)kmitl.ac.th

Subject:
[6622][(Team Name)][IoT/Metaverse] Group Exercise Submission
Example:
[6622][Saltburn][Metaverse] Group Exercise Submission