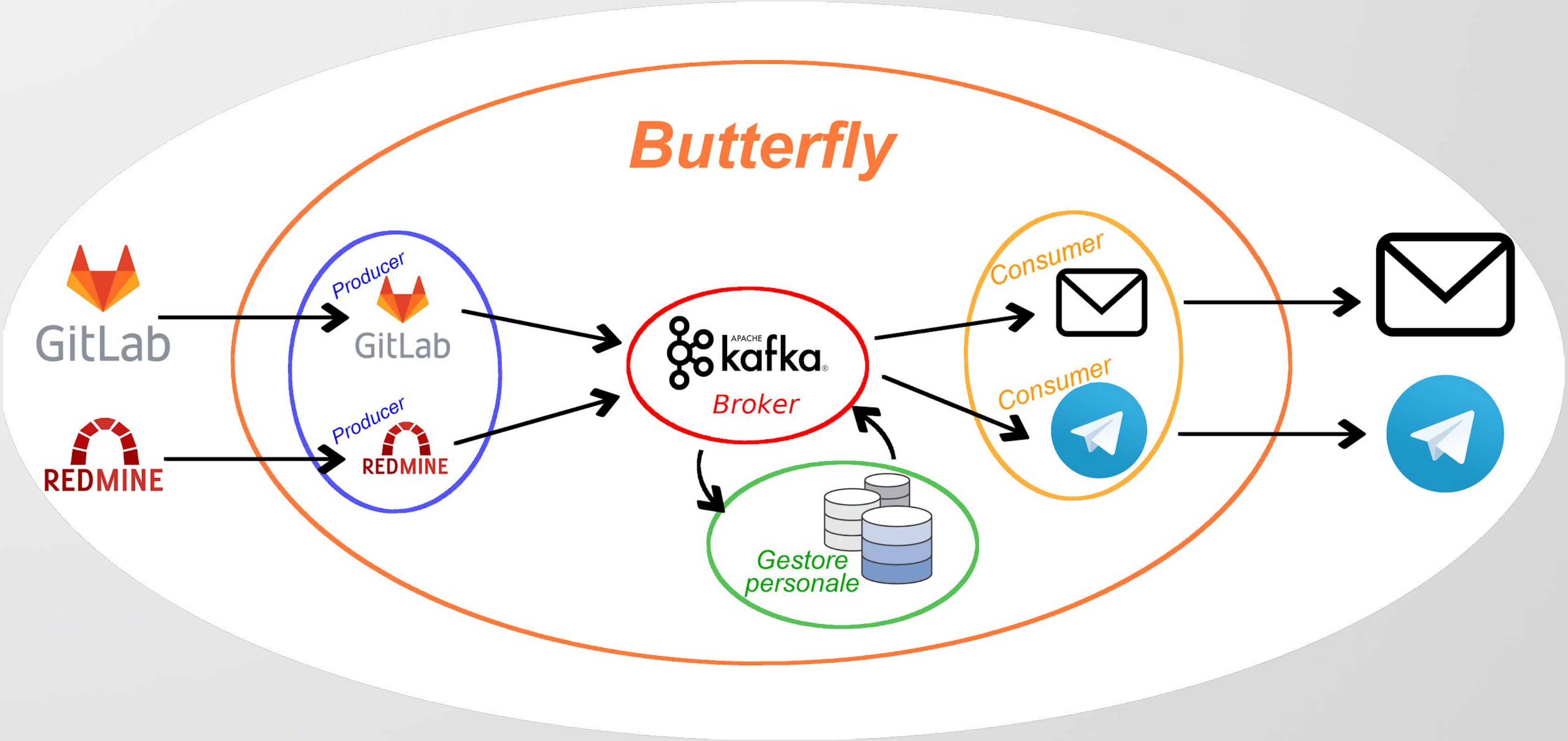
Butterfly

Product Baseline

Università degli Studi di Padova
02 / 04 / 2019

Butterfly in generale



Linguaggi e librerie

Python

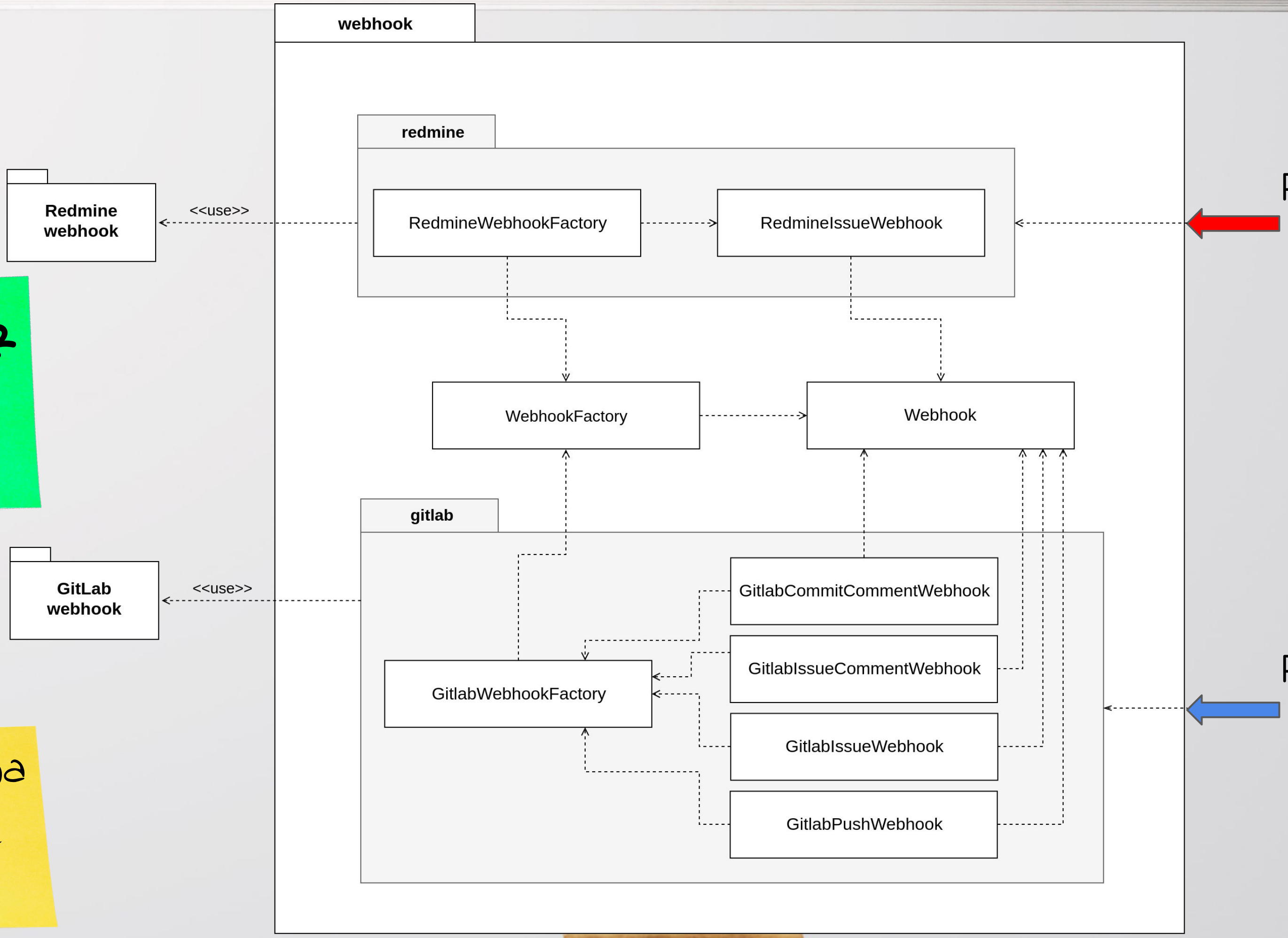
- Flask
- KafkaPython
- Telepot
- SMTPlib
- PyMongo

Producer, Consumer, Gestore personale

HTML + CSS + JS

Gestore Personale

PRODUCER



webhook

redmine

RedmineWebhookFactory

RedmineIssueWebhook

WebhookFactory

Webhook

gitlab

GitlabCommitCommentWebhook

GitlabIssueCommentWebhook

GitlabIssueWebhook

GitlabPushWebhook

GitlabWebhookFactory

Redmine webhook

GitLab webhook

<<use>>

<<use>>

Producer Redmine

Producer GitLab

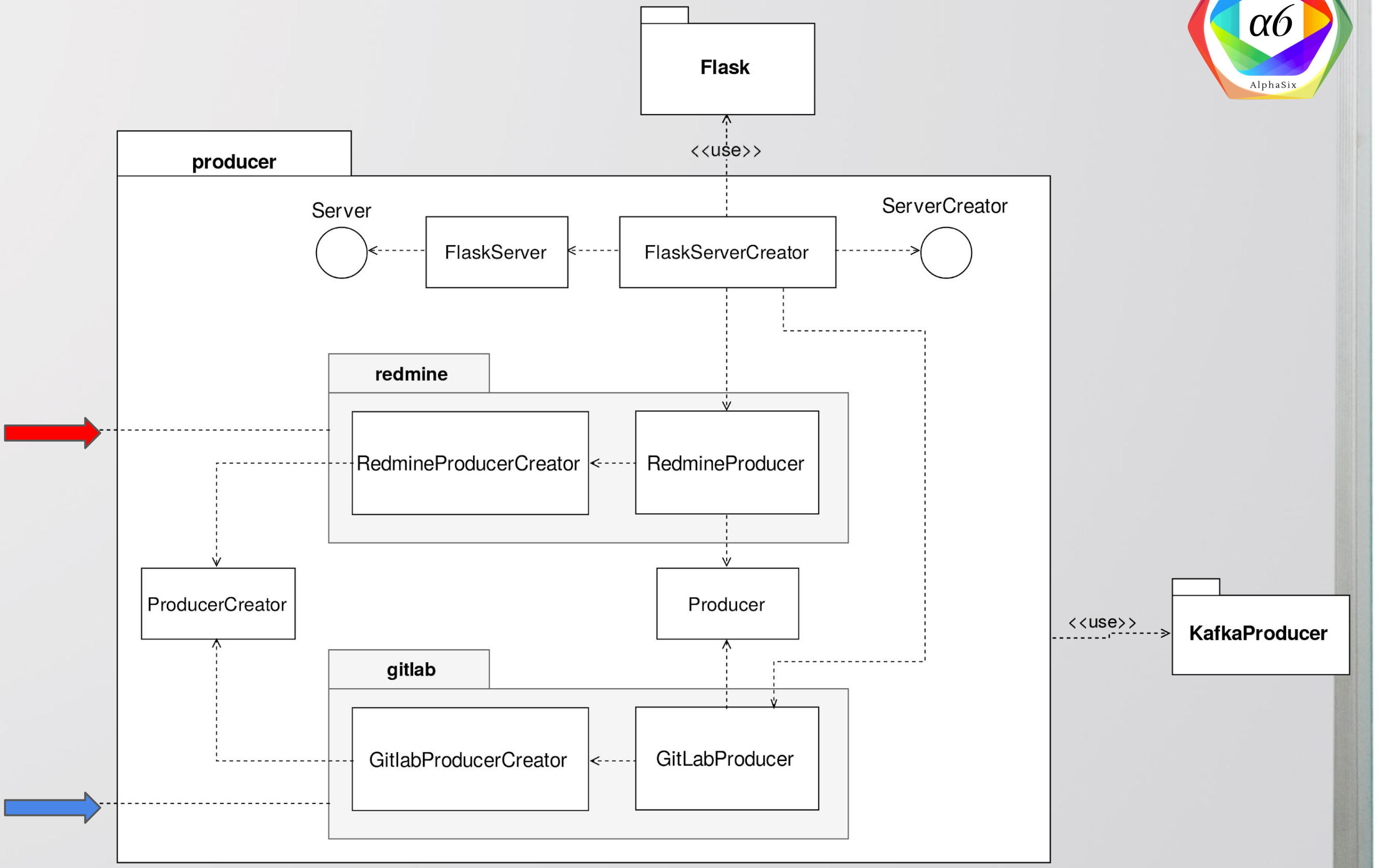
Diagramma Package

PRODUCER

Diagramma Package

Webhook Redmine

Webhook GitLab



Flask

<<use>>

producer

Server

FlaskServer

FlaskServerCreator

ServerCreator

redmine

RedmineProducerCreator

RedmineProducer

ProducerCreator

Producer

gitlab

GitlabProducerCreator

GitLabProducer

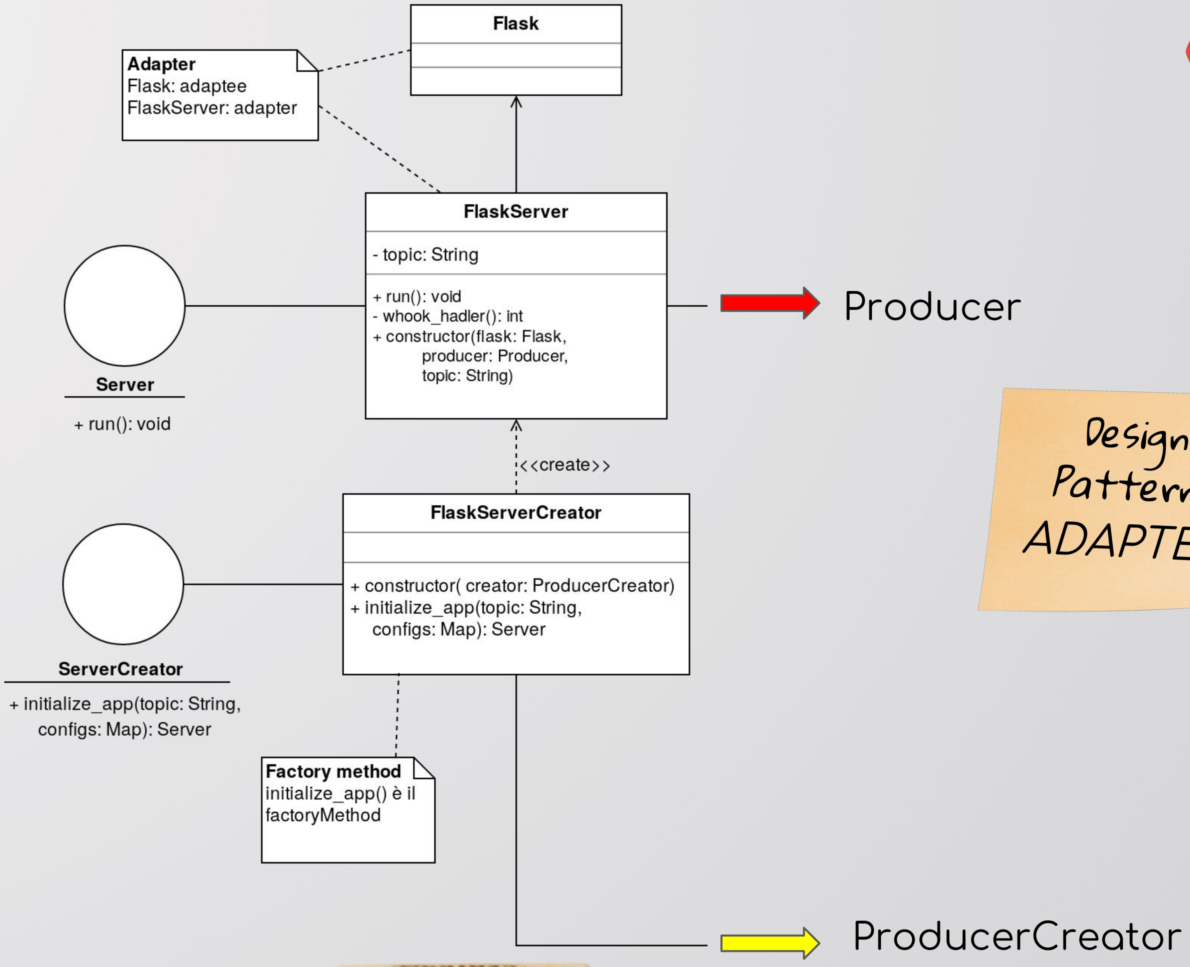
<<use>>

KafkaProducer

PRODUCER

Diagramma Classi

Design Pattern: ADAPTER



Flask

Adapter
Flask: adaptee
FlaskServer: adapter

FlaskServer
- topic: String
+ run(): void
- whook_hadler(): int
+ constructor(flask: Flask,
  producer: Producer,
  topic: String)

Server
+ run(): void

<<create>>

FlaskServerCreator
+ constructor(creator: ProducerCreator)
+ initialize_app(topic: String,
  configs: Map): Server

ServerCreator
+ initialize_app(topic: String,
configs: Map): Server

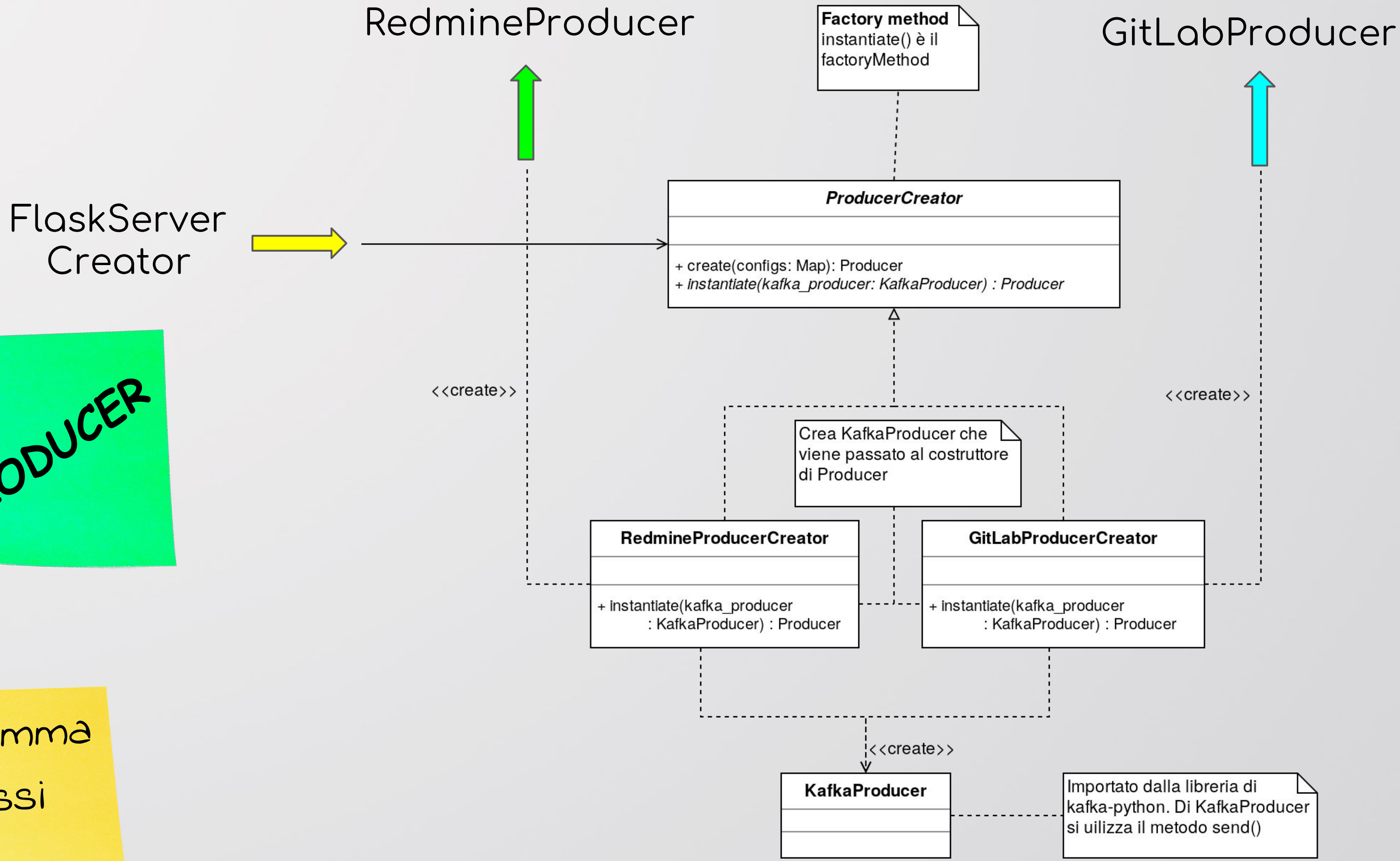
Factory method
initialize_app() è il
factoryMethod

Producer

ProducerCreator

PRODUCER

Diagramma Classi



RedmineProducer

Factory method
instantiate() è il factoryMethod

GitLabProducer

FlaskServer Creator

ProducerCreator

+ create(configs: Map): Producer
+ *instantiate(kafka_producer: KafkaProducer) : Producer*

<<create>>

<<create>>

Crea KafkaProducer che viene passato al costruttore di Producer

RedmineProducerCreator

+ instantiate(kafka_producer : KafkaProducer) : Producer

GitLabProducerCreator

+ instantiate(kafka_producer : KafkaProducer) : Producer

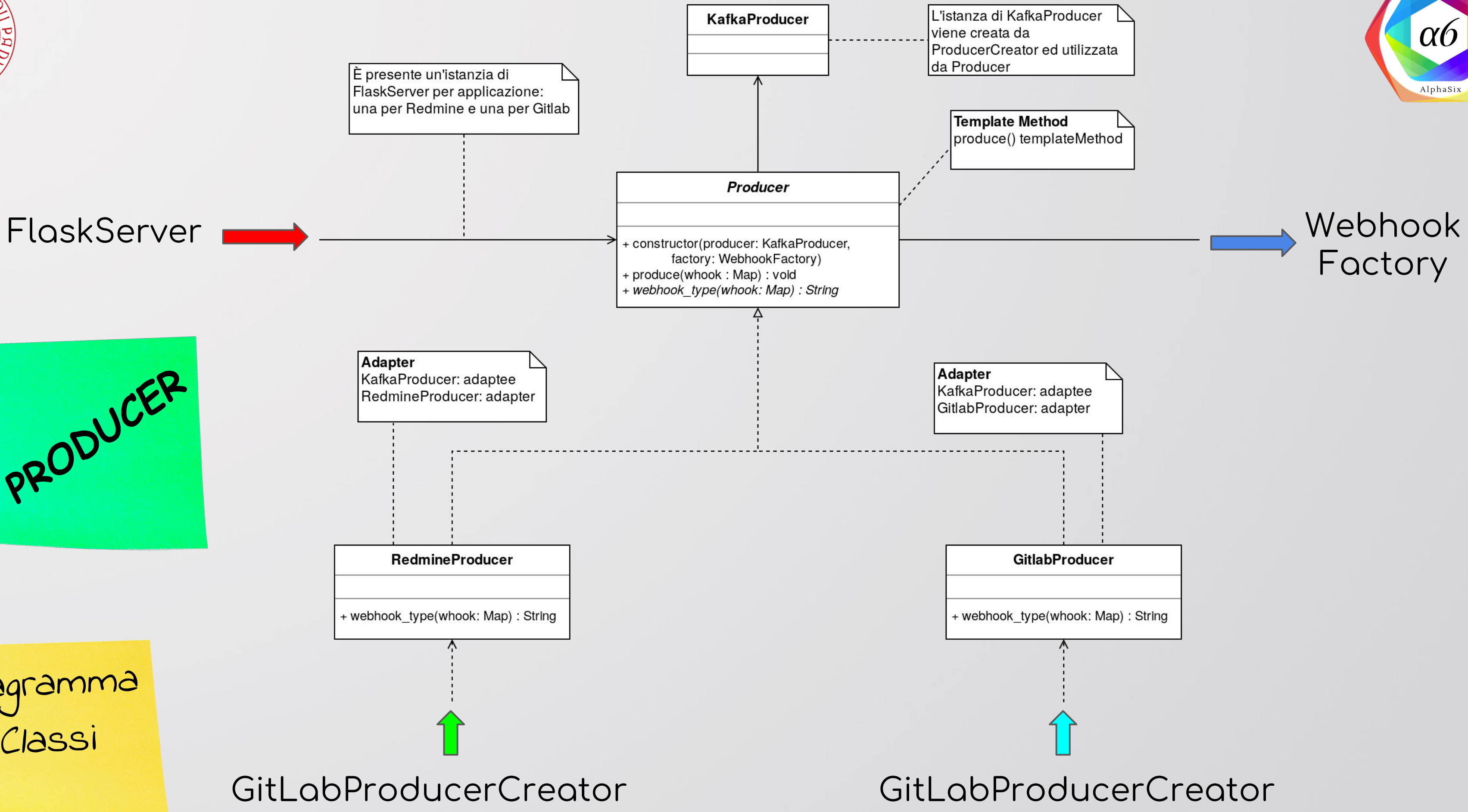
<<create>>

KafkaProducer

Importato dalla libreria di kafka-python. Di KafkaProducer si uilizza il metodo send()

Diagramma Classi

PRODUCER



FlaskServer

È presente un'istanza di FlaskServer per applicazione: una per Redmine e una per Gitlab

KafkaProducer

L'istanza di KafkaProducer viene creata da ProducerCreator ed utilizzata da Producer

Template Method
produce() templateMethod

Producer

+ constructor(producer: KafkaProducer, factory: WebhookFactory)
+ produce(whook : Map) : void
+ *webhook_type(whook: Map) : String*

Webhook Factory

Adapter
KafkaProducer: adaptee
RedmineProducer: adapter

Adapter
KafkaProducer: adaptee
GitlabProducer: adapter

RedmineProducer

+ webhook_type(whook: Map) : String

GitlabProducer

+ webhook_type(whook: Map) : String

GitLabProducerCreator

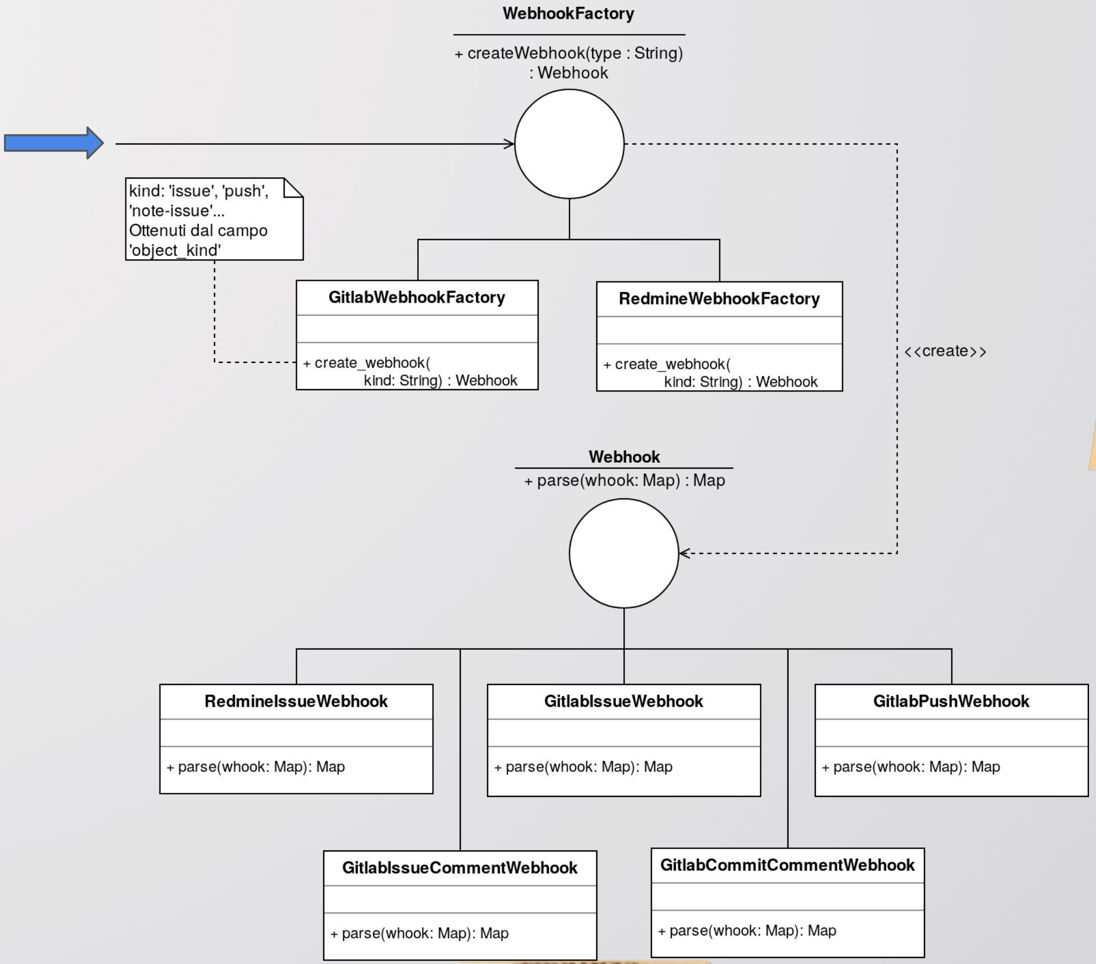
GitLabProducerCreator

PRODUCER

Diagramma Classi

Design Pattern: ABSTRACT FACTORY

Producer



WebhookFactory

+ createWebhook(type : String) : Webhook

kind: 'issue', 'push', 'note-issue'... Ottenuti dal campo 'object_kind'

GitlabWebhookFactory

+ create_webhook(kind: String) : Webhook

RedmineWebhookFactory

+ create_webhook(kind: String) : Webhook

<<create>>

Webhook

+ parse(whook: Map) : Map

RedmineIssueWebhook

+ parse(whook: Map): Map

GitlabIssueWebhook

+ parse(whook: Map): Map

GitlabPushWebhook

+ parse(whook: Map): Map

GitlabIssueCommentWebhook

+ parse(whook: Map): Map

GitlabCommitCommentWebhook

+ parse(whook: Map): Map

PRODUCER

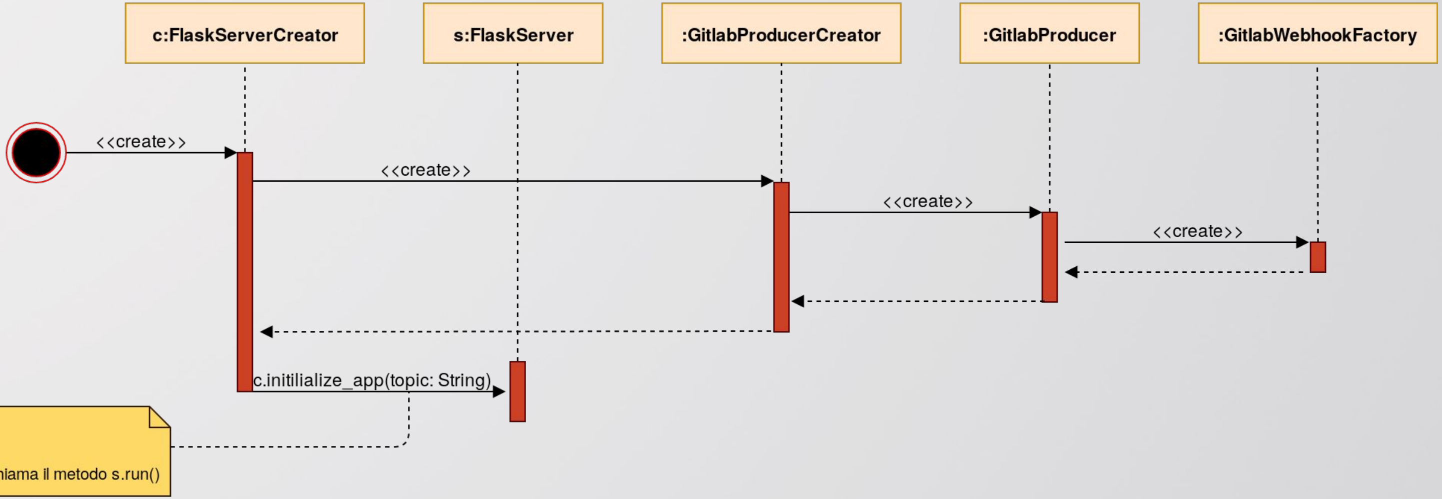
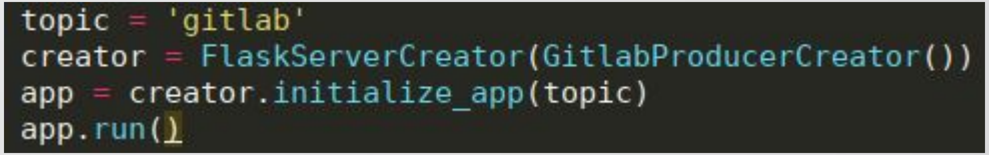


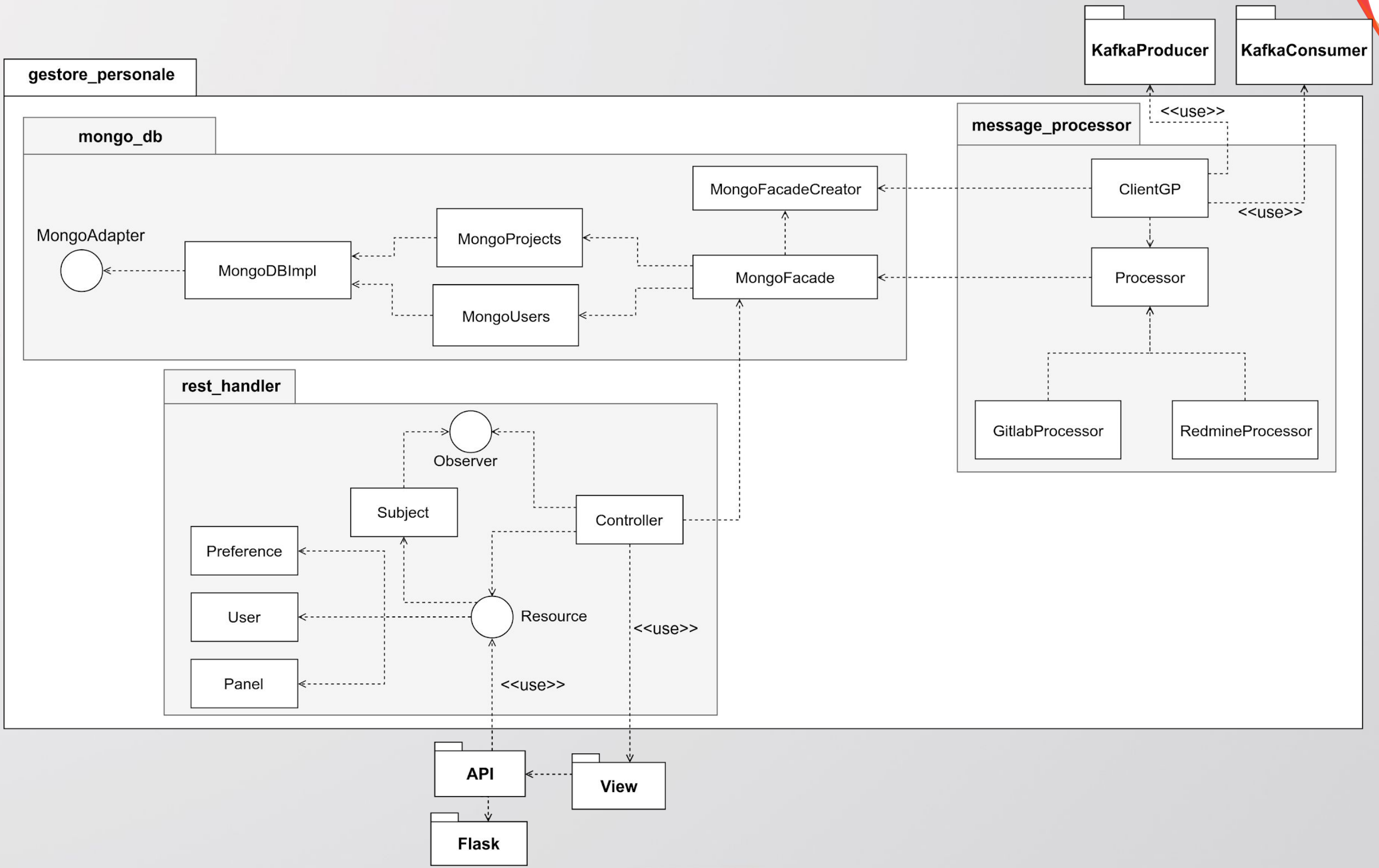
Diagramma Sequenza



```
topic = 'gitlab'
creator = FlaskServerCreator(GitlabProducerCreator())
app = creator.initialize_app(topic)
app.run()
```

GESTORE PERSONALE

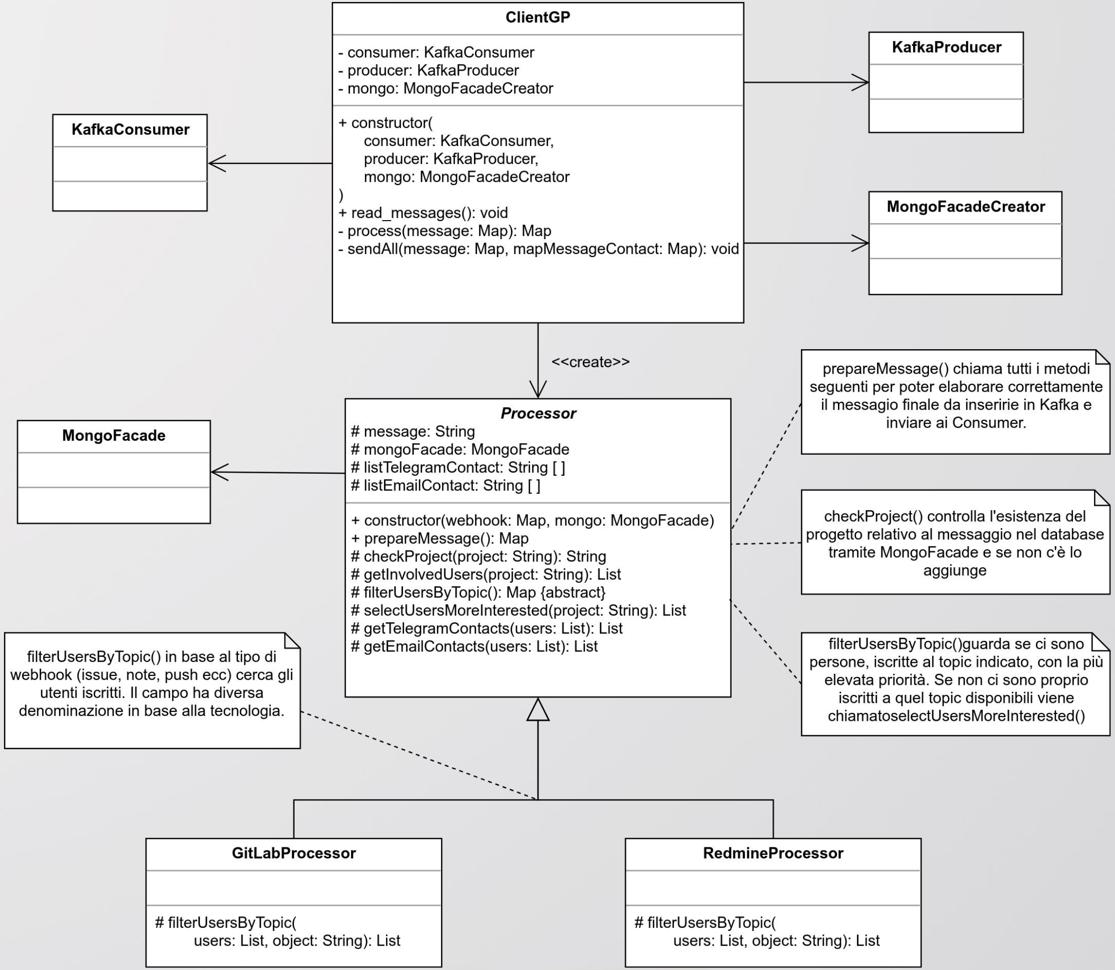
Diagramma Package



GESTORE PERSONALE

Diagramma Classi

Design Pattern: TEMPLATE METHOD



ClientGP

- consumer: KafkaConsumer
- producer: KafkaProducer
- mongo: MongoFacadeCreator

```
+ constructor(
      consumer: KafkaConsumer,
      producer: KafkaProducer,
      mongo: MongoFacadeCreator
)
+ read_messages(): void
- process(message: Map): Map
- sendAll(message: Map, mapMessageContact: Map): void
```

KafkaConsumer

KafkaProducer

MongoFacadeCreator

<<create>>

Processor

```
# message: String
# mongoFacade: MongoFacade
# listTelegramContact: String [ ]
# listEmailContact: String [ ]

+ constructor(webhook: Map, mongo: MongoFacade)
+ prepareMessage(): Map
# checkProject(project: String): String
# getInvolvedUsers(project: String): List
# filterUsersByTopic(): Map {abstract}
# selectUsersMoreInterested(project: String): List
# getTelegramContacts(users: List): List
# getEmailContacts(users: List): List
```

MongoFacade

prepareMessage() chiama tutti i metodi seguenti per poter elaborare correttamente il messaggio finale da inseririe in Kafka e inviare ai Consumer.

checkProject() controlla l'esistenza del progetto relativo al messaggio nel database tramite MongoFacade e se non c'è lo aggiunge

filterUsersByTopic()guarda se ci sono persone, iscritte al topic indicato, con la più elevata priorità. Se non ci sono proprio iscritti a quel topic disponibili viene chiamatoselectUsersMoreInterested()

filterUsersByTopic() in base al tipo di webhook (issue, note, push ecc) cerca gli utenti iscritti. Il campo ha diversa denominazione in base alla tecnologia.

GitLabProcessor

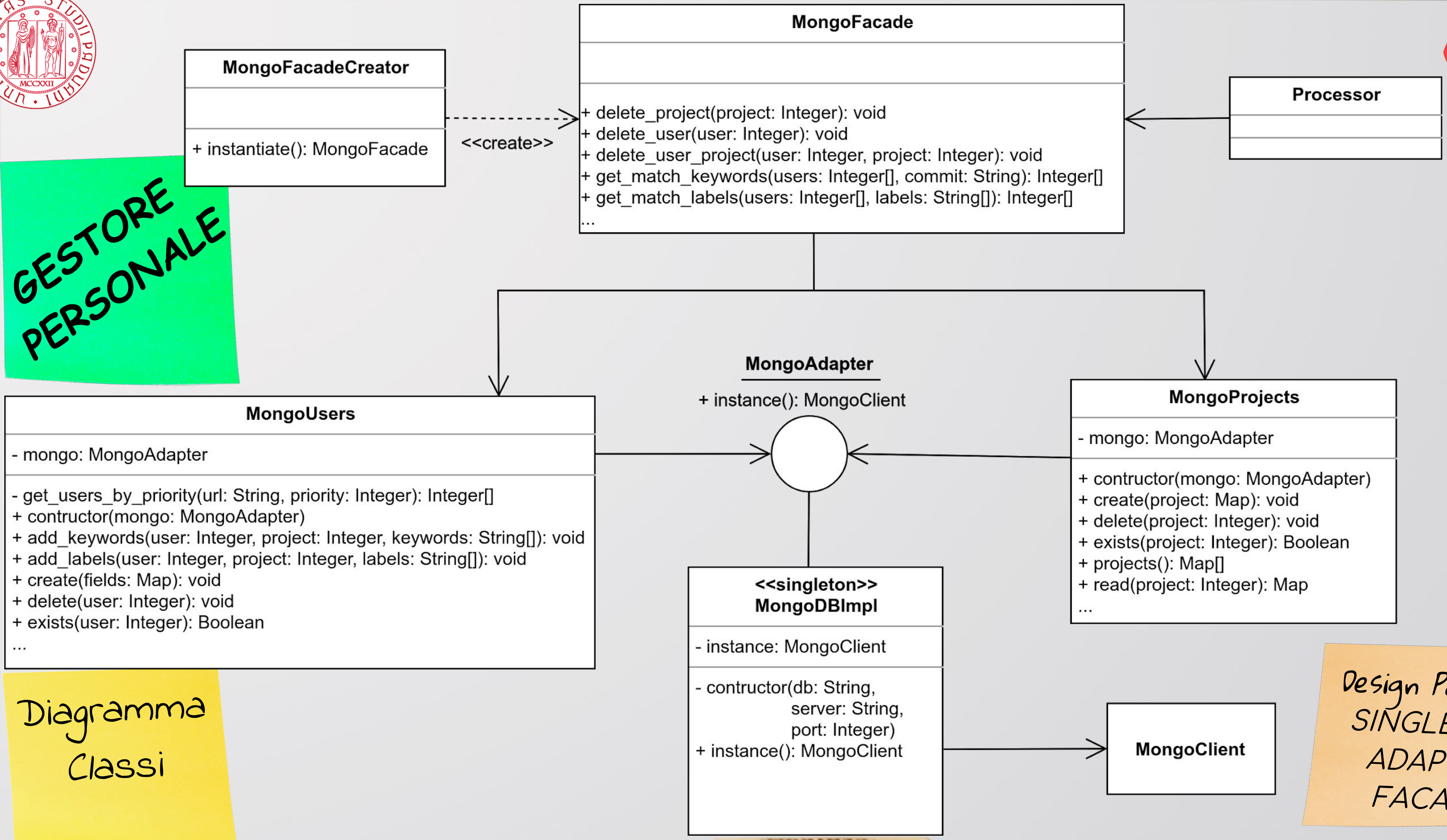
```
# filterUsersByTopic(
      users: List, object: String): List
```

RedmineProcessor

```
# filterUsersByTopic(
      users: List, object: String): List
```

GESTORE PERSONALE

Diagramma Classi



MongoFacadeCreator

+ instantiate(): MongoFacade

<<create>>

MongoFacade

+ delete_project(project: Integer): void
+ delete_user(user: Integer): void
+ delete_user_project(user: Integer, project: Integer): void
+ get_match_keywords(users: Integer[], commit: String): Integer[]
+ get_match_labels(users: Integer[], labels: String[]): Integer[]
...

Processor

MongoAdapter

+ instance(): MongoClient

MongoUsers

- mongo: MongoAdapter

- get_users_by_priority(url: String, priority: Integer): Integer[]
+ contructor(mongo: MongoAdapter)
+ add_keywords(user: Integer, project: Integer, keywords: String[]): void
+ add_labels(user: Integer, project: Integer, labels: String[]): void
+ create(fields: Map): void
+ delete(user: Integer): void
+ exists(user: Integer): Boolean
...

MongoProjects

- mongo: MongoAdapter

+ contructor(mongo: MongoAdapter)
+ create(project: Map): void
+ delete(project: Integer): void
+ exists(project: Integer): Boolean
+ projects(): Map[]
+ read(project: Integer): Map
...

<<singleton>>
MongoDBImpl

- instance: MongoClient

- contructor(db: String,
  server: String,
  port: Integer)
+ instance(): MongoClient

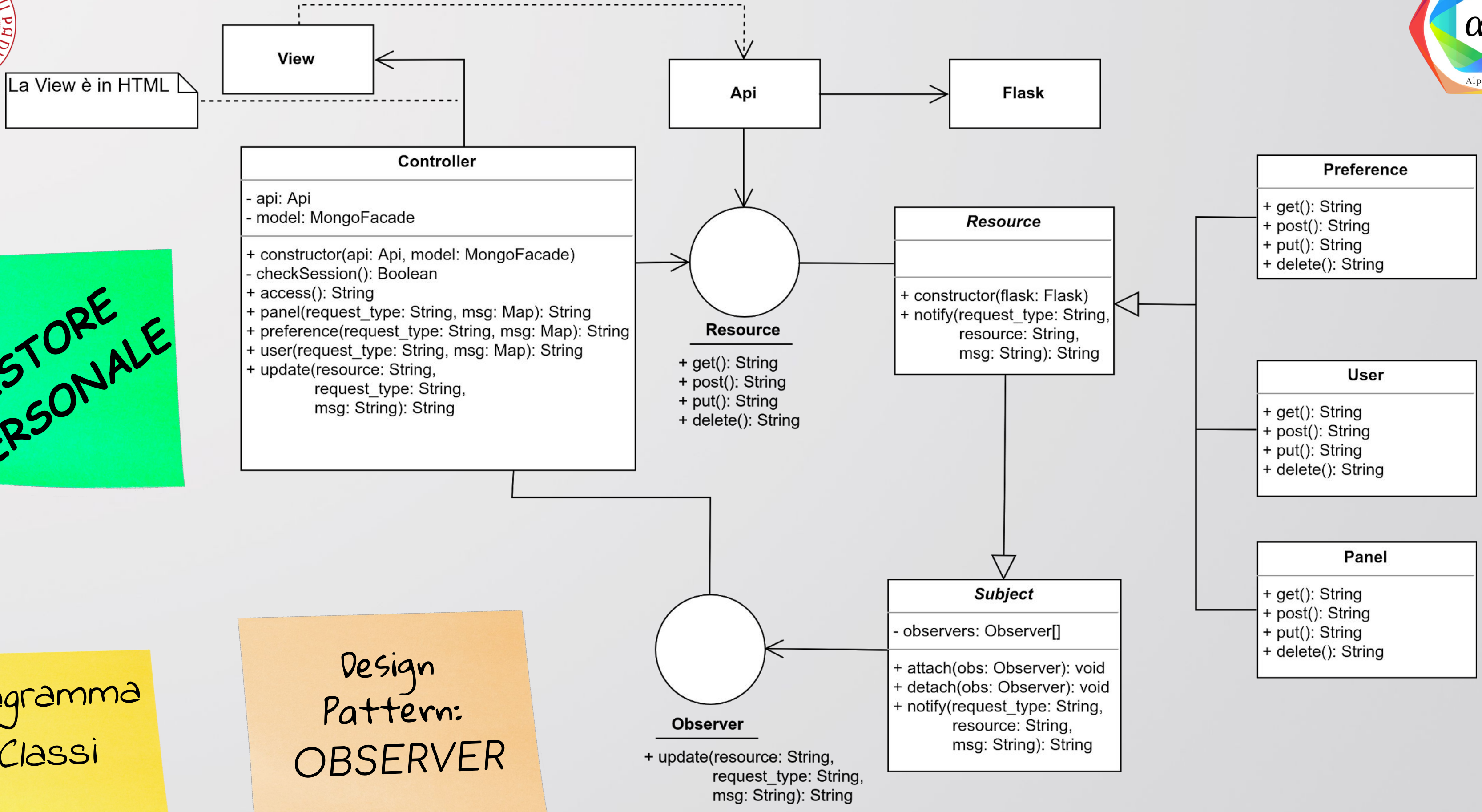
MongoClient

Design Pattern:
SINGLETON,
ADAPTER,
FACADE

GESTORE PERSONALE

Diagramma Classi

Design Pattern: OBSERVER



La View è in HTML

View

Api

Flask

Controller

- api: Api
- model: MongoFacade

+ constructor(api: Api, model: MongoFacade)
- checkSession(): Boolean
+ access(): String
+ panel(request_type: String, msg: Map): String
+ preference(request_type: String, msg: Map): String
+ user(request_type: String, msg: Map): String
+ update(resource: String, request_type: String, msg: String): String

Resource

+ get(): String
+ post(): String
+ put(): String
+ delete(): String

Resource

+ constructor(flask: Flask)
+ notify(request_type: String, resource: String, msg: String): String

Preference

+ get(): String
+ post(): String
+ put(): String
+ delete(): String

User

+ get(): String
+ post(): String
+ put(): String
+ delete(): String

Panel

+ get(): String
+ post(): String
+ put(): String
+ delete(): String

Subject

- observers: Observer[]

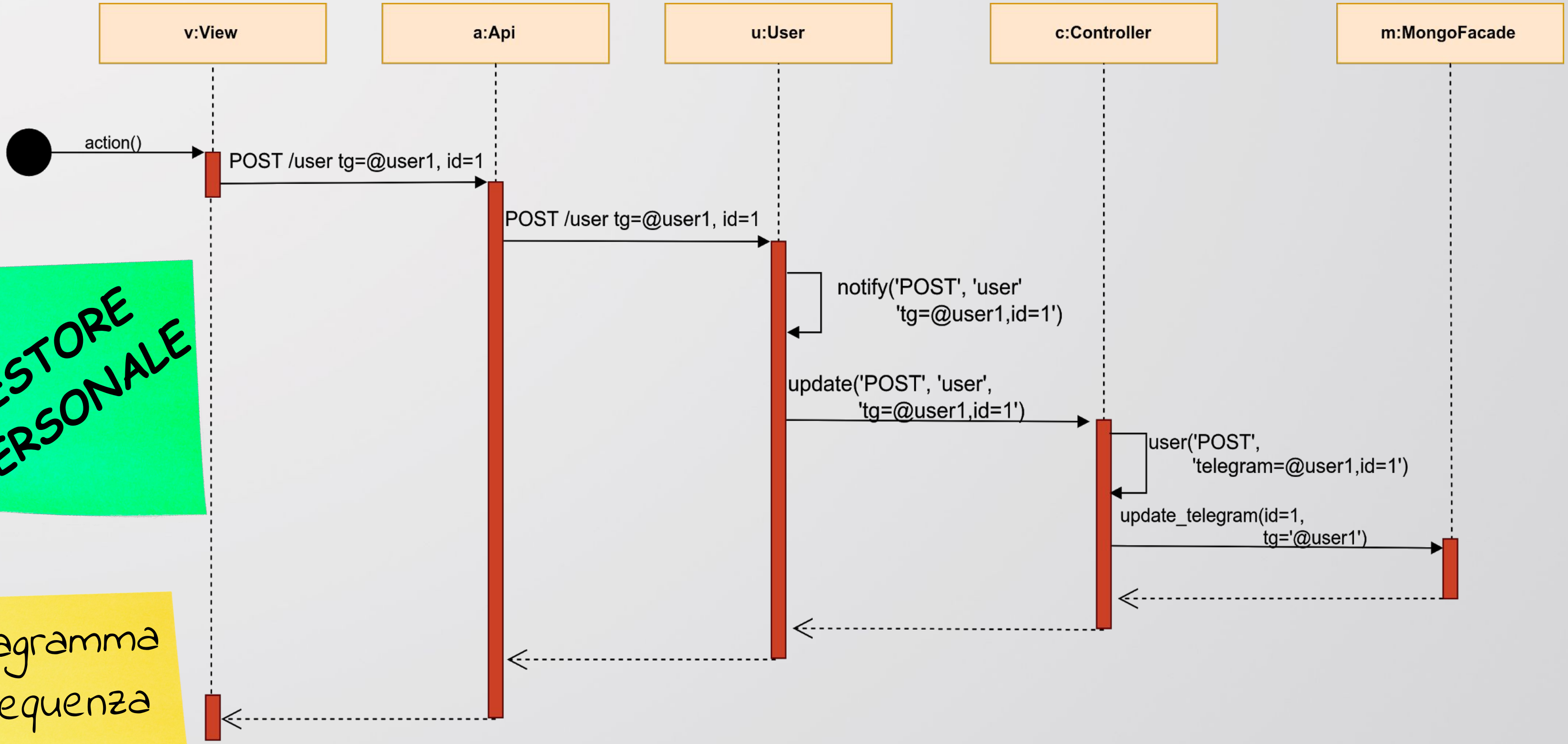
+ attach(obs: Observer): void
+ detach(obs: Observer): void
+ notify(request_type: String, resource: String, msg: String): String

Observer

+ update(resource: String, request_type: String, msg: String): String

GESTORE PERSONALE

Diagramma Sequenza



v:View

a:Api

u:User

c:Controller

m:MongoFacade

action()

POST /user tg=@user1, id=1

POST /user tg=@user1, id=1

notify('POST', 'user'
'tg=@user1,id=1')

update('POST', 'user',
'tg=@user1,id=1')

user('POST',
'telegram=@user1,id=1')

update_telegram(id=1,
tg='@user1')

CONSUMER

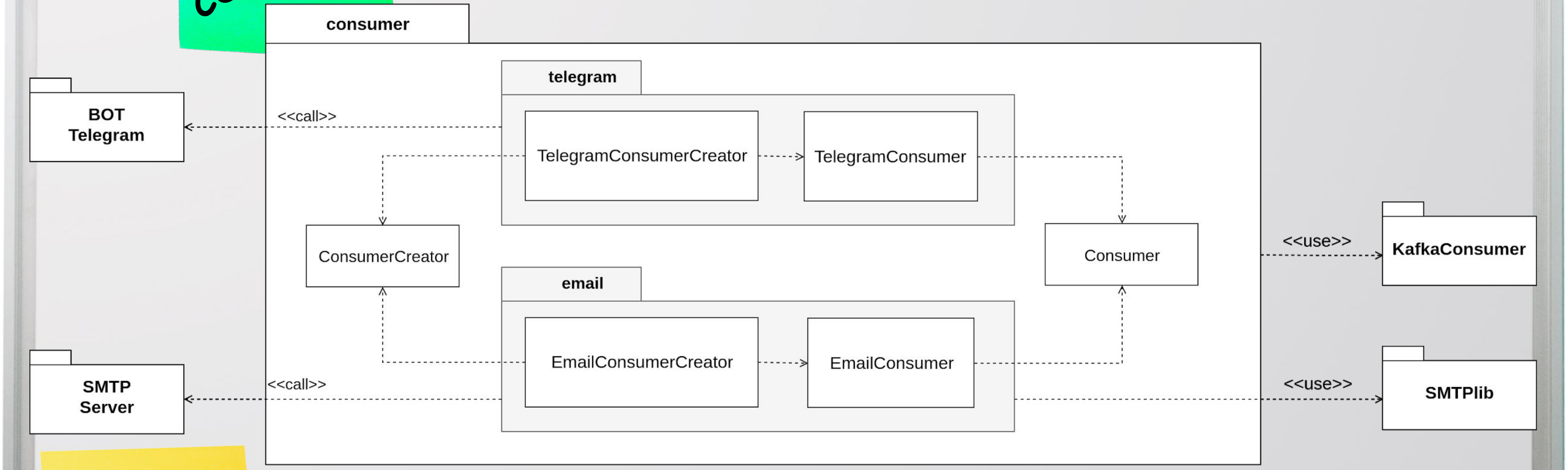
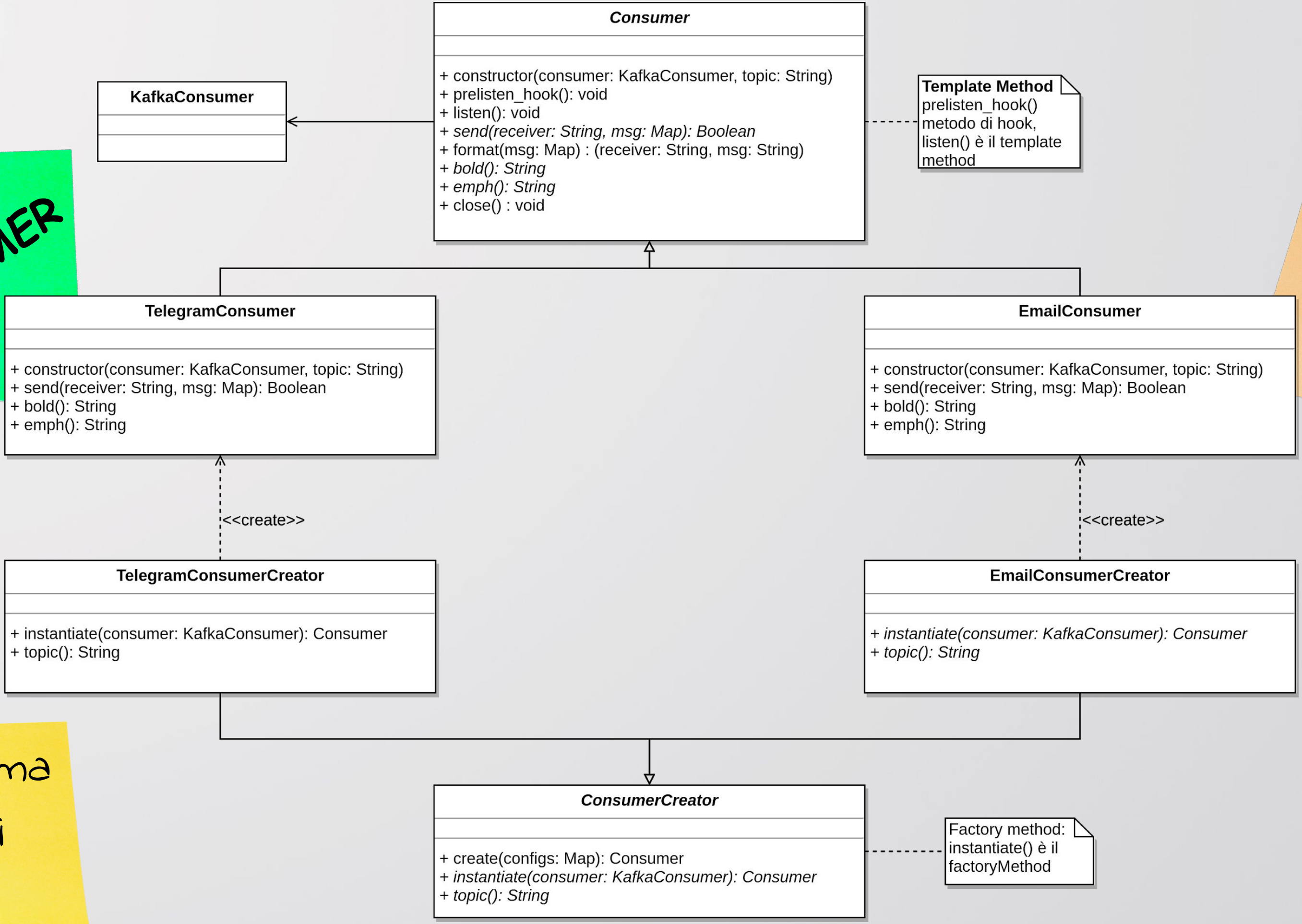


Diagramma Classi

CONSUMER



Consumer

+ constructor(consumer: KafkaConsumer, topic: String)
+ prelisten_hook(): void
+ listen(): void
+ *send(receiver: String, msg: Map): Boolean*
+ format(msg: Map) : (receiver: String, msg: String)
+ *bold(): String*
+ *emph(): String*
+ close() : void

Template Method
prelisten_hook()
metodo di hook,
listen() è il template
method

KafkaConsumer

TelegramConsumer

+ constructor(consumer: KafkaConsumer, topic: String)
+ send(receiver: String, msg: Map): Boolean
+ bold(): String
+ emph(): String

EmailConsumer

+ constructor(consumer: KafkaConsumer, topic: String)
+ send(receiver: String, msg: Map): Boolean
+ bold(): String
+ emph(): String

TelegramConsumerCreator

+ instantiate(consumer: KafkaConsumer): Consumer
+ topic(): String

EmailConsumerCreator

+ *instantiate(consumer: KafkaConsumer): Consumer*
+ *topic(): String*

ConsumerCreator

+ create(configs: Map): Consumer
+ *instantiate(consumer: KafkaConsumer): Consumer*
+ *topic(): String*

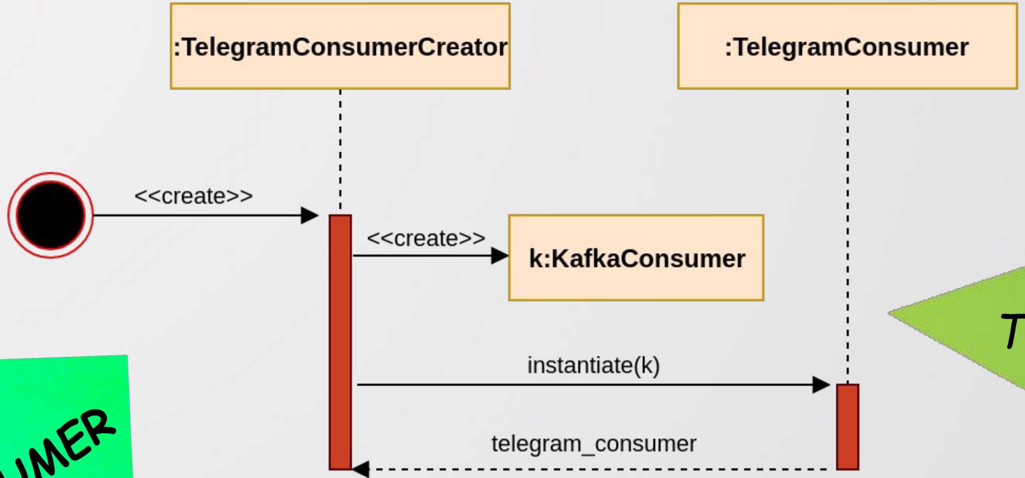
Factory method:
instantiate() è il
factoryMethod

Design Pattern: TEMPLATE METHOD

Diagramma Classi

CONSUMER

Diagrammi Sequenza



:TelegramConsumerCreator

:TelegramConsumer

<<create>>

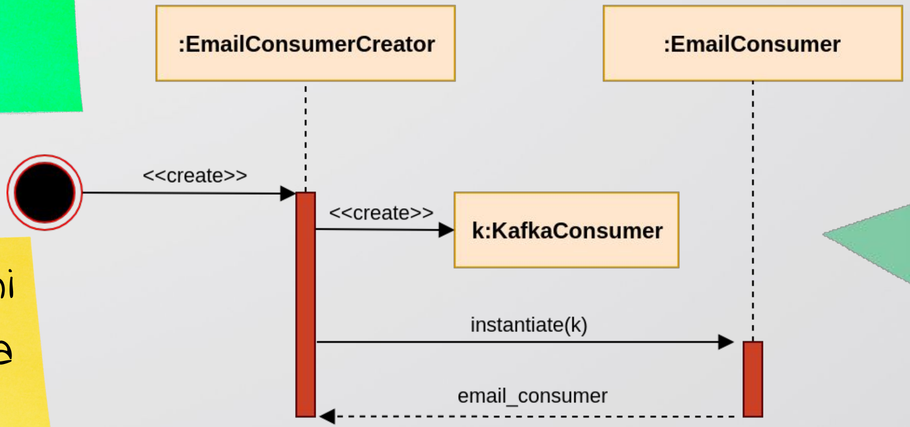
<<create>>

k:KafkaConsumer

instantiate(k)

telegram_consumer

TelegramConsumerCreator



:EmailConsumerCreator

:EmailConsumer

<<create>>

<<create>>

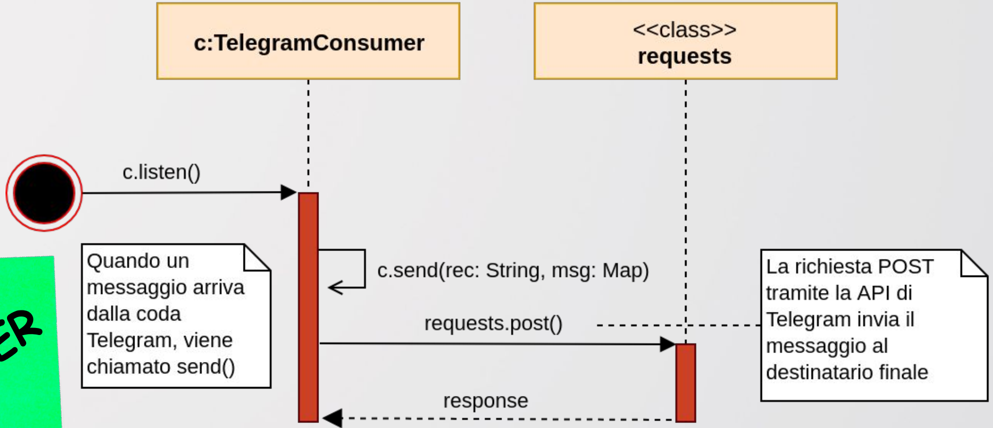
k:KafkaConsumer

instantiate(k)

email_consumer

EmailConsumerCreator

CONSUMER



c:TelegramConsumer

<<class>>
requests

c.listen()

c.send(rec: String, msg: Map)

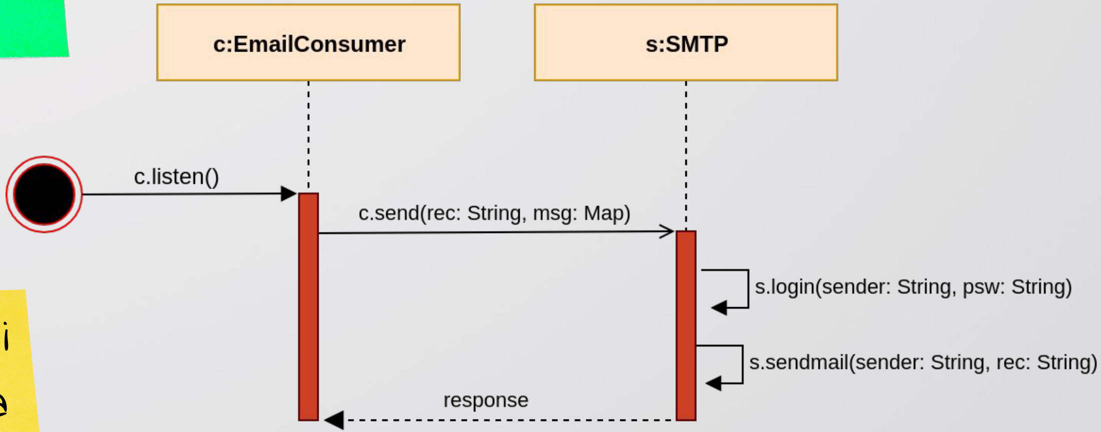
requests.post()

response

Quando un messaggio arriva dalla coda Telegram, viene chiamato send()

La richiesta POST tramite la API di Telegram invia il messaggio al destinatario finale

TelegramConsumer



c:EmailConsumer

s:SMTP

c.listen()

c.send(rec: String, msg: Map)

s.login(sender: String, psw: String)

s.sendmail(sender: String, rec: String)

response

EmailConsumer

Diagrammi Sequenza

Riepilogo Design Pattern

Creazionali

- FACTORY METHOD
- SINGLETON
- ABSTRACT FACTORY

Strutturali

- FACADE
- ADAPTER

Comportamentali

- OBSERVER
- TEMPLATE METHOD

Architetturali

- MVC
- DEPENDENCY INJECTION

Alcuni principi OOP

Composition over inheritance

Single responsibility

Dependency inversion

- “The object-oriented version of spaghetti code is, of course, ‘lasagna code’. Too many layers.”

~ Roberto Waltman