Universitatea Tehnică a Moldovei

# RAPORT

La lucrarea de laborator Nr. 1
La disciplina : PAD

**Efectuat**: st. gr FI-141 Cernei Eugeniu
**Verificat**: lect. superior Alex Gavrișco

Chișinău 2017

## Obiective

- •Studiul agenților de mesagerie;
- •Elaborarea unui protocol de comunicare al agentului de mesaje;
- •Tratarea concurentă a mesajelor;
- •Alegerea protocolului de transport (în dependență de scopul/domeniul de aplicare al agentului de mesaje);
- •Alerea și elaborarea strategiei de păstrare a mesajelor;

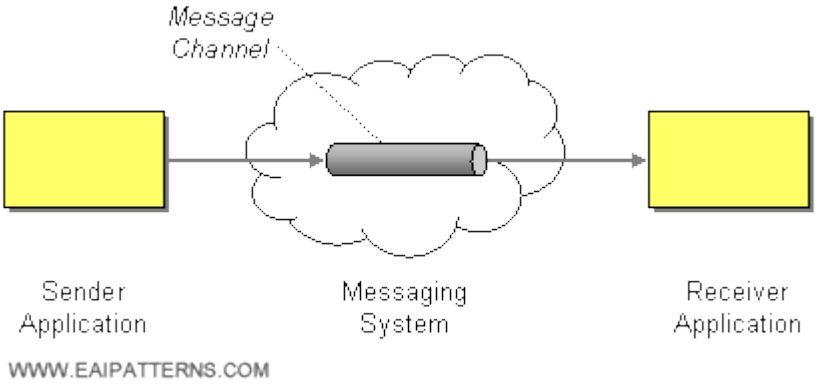
## Sarcinile și baremul

Sarcina de bază este minim necesar pentru această lucrare de laborator. Toate sarcinile reprezintă o continuitate. Adică, sarcina pentru nota 6 reprezintă cîteva sarcini care se vor baza pe sarcina pentru nota 5.

**Sarcina de bază - Implementarea unei cozi de mesaje(nota 5)**

- •Elaborarea protocolului de comunicare. Descrie protocolul într-un fișier markdown și salvează-l în mapa „docs” din repozitoriul tău;
- •Alegerea protocului de transport și argumentarea alegerii;
- •Implementarea cozii de mesaje (utilizînd colecții de date concurente)

Implementează o coadă de mesaje care poate avea atît multipli productători (expeditori) de mesaje, cît și multipli consumatori de mesaje.




Sistemul trebuie să permită:

- •plasarea unui mesaj în coadă (concurent de mai mulți producători);
- •consumarea mesajelor (concurent de mai mulți consumatori);

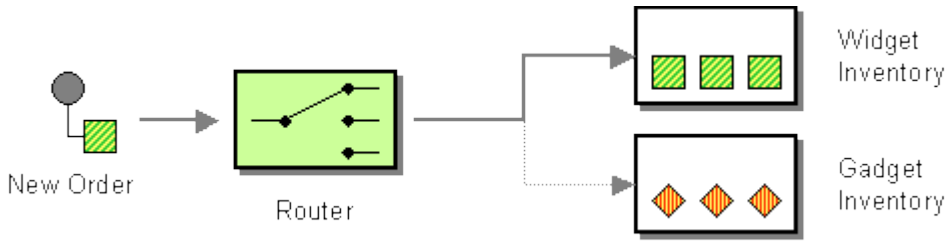
**Implementarea mecanismului de stocare a mesajelor (Nota 6)**

Stocarea mesajelor cu scopul asigurării robusteții cozii de mesaje. Adică, sistemul trebuie să:

- •Serializeze coada de mesaje;
- •Să o stocheze persistent (memorie secundară);
- •În cazul întreruperii lucrului sistemului să restabilească (deserializeze) coada de mesaje din copia stocată.

**Implementarea mecanismului de rutare a mesajelor (Nota 7)**

Este necesar de a adăuga în sistem suportul la multiple cozi de mesaje (atît pentru producători, cît și pentru consumatori). Cît și a mecanismului de rutare a mesajelor la cozi.




Cerințe concrete:

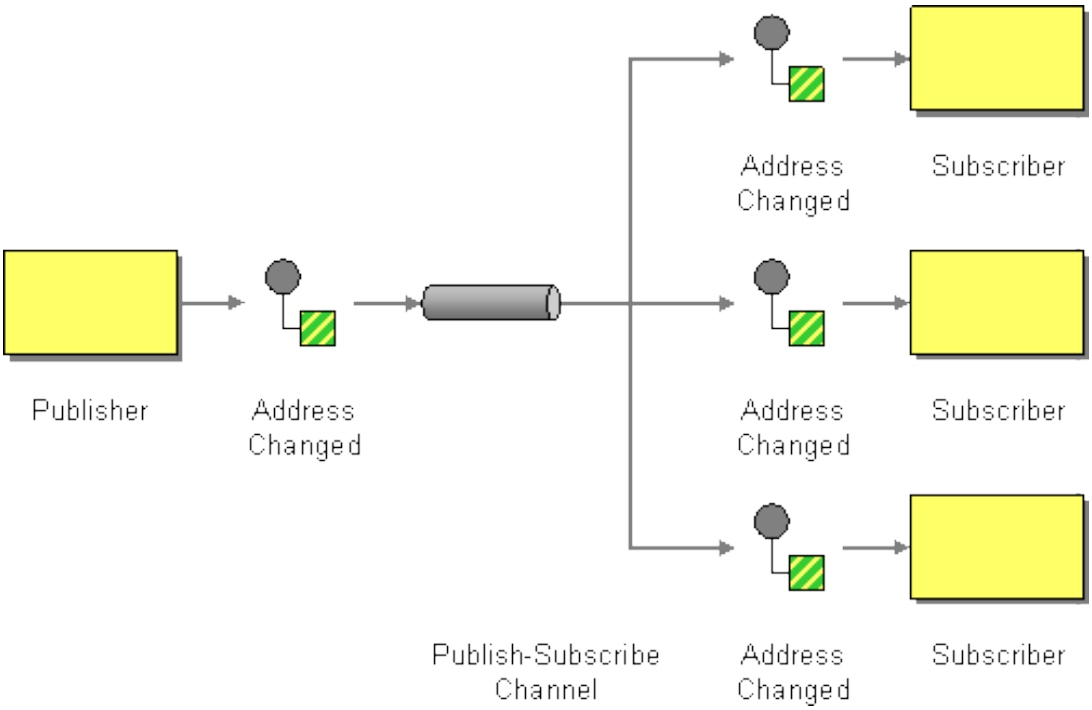
- •Ajustarea protocolului agentului de mesaje cu scopul de a adăuga mecanismul de rutare;
- •Adăugarea multiplelor cozi (de dorit să fie dinamic - în dependență de cererea producătorului);
- •Consumarea dintr-o coadă non existentă trebuie să producă eroare explicită, prevăzută de protocol.

Bonus points (+1):

- •Asigură compatibilitatea protocolului cu clienții din etapa precedentă (fără mecanism de rutare);
- •Implementează cozi persistente și non-persistente. Acele persistente vor rămîne chiar după consumarea tuturor mesajelor, non-persistente se distrug după consumarea ultimului mesaj.

**Implementarea patternului de publisher-subscriber (Nota 8)**

Taskurile precedente presupun că consumatorul cerea explicit de fiecare dată un mesaj din coadă. Pattern-ul publisher-subscriber permite consumatorului o singură dată să se „aboneze” (subscribe) la mesaje după un anumit criteriu, și la apariția acestui mesaj, dacă sunt consumatori, el este automat scos din coadă și transmis consumatorilor respectivi.




Rolul producătorului rămîne același (se schimbă doar denumirea). Însă rolul consumatorului este mai simplu - el trebuie doar să se aboneze, să mențină conexiunea deschisă și să aștepte mesaje respective.

- •Ajustarea protocolului agentului de mesaje;
- •Adăugarea mecanismului de abonare (subscribe);
- •Subscribe la o coadă non existentă, la fel produce o eroare prevăzută de protocol.

**Implementarea rutării avansate a mesajelor (Nota 9)**

Pînă la această etapă, deși existau multiple cozi, atît producătorul cît și consumatorul puteau lucra doar cu una. Sarcina este de a adăuga posibilitatea rutării avansate a mesajelor (expedierea în multiple cozi, abonarea la multiple cozi).

Ajustează protocolul și implementează următoarele sarcini:

- rutarea după numele cozii (posibilitatea de a specifica un pattern);
- posiblitatea de a enumera explicit cozile;

Exemplu rutării după numele cuzii: Să presupunem că convenția la dumele cozii e următoarea- <numele companiei>.<numele produsului>.<tipul mesajelor - eroare, info, etc>

Atunci, vor fi disponibile următoarele variații:

- Google.* - toate mesajele pentru toate produsele a companiei „Google”
- Google.Nest.* - toate mesajele pentru produsul „Nest” a companiei „Google”
- Google.*.error - toate messajele de tip „error” pentru compania „Google”
- *.*.critical-erorr - toate mesajele pentru toate companiile de tip „critical-error”
- etc.

Acesta e doar un exemplu. Puteți oferi posibilitatea de a specifica o expresie regulată (RegEx) la expediere/abonare.

**Implementarea mecanismului „last will and testament” (Nota 10)**

Mecanismul „last will and testament” (impementat în protocolul mqtt) este utilizat pentru a notifica despre deconectarea anormală abonatului (subscriber). Adică, sunt situații cînd este necesar să cunoști dacă deconectarea abonatului a fost una așteptată sau nu. Însă pentru aceasta sistemul trebuie să deosebească noțiunea de deconectare planificată și anormală.

Sarcinile:

- Ajustează protocolul ca subscriberii, înainte de deconectare să transmită un mesaj specific;
- Ajustează protocolul ca subscriberii la conectare să transmită: mesajul și coada pentru „last will and testament”;
- Ajustează sistemul ca acesta să detecteze deconectări anormale și să transmită mesajul „last will and testament” (dacă este cazul).

## Realizarea task-urilor

În cadrul laboratorului au existat două versiuni majore a proiectului:

**Message Queue:**

Există 3 module diferite: Sender, Receiver, Server.
Comunicarea se face pe baza la mesaje JSON, care au câmpurile Type și Info. Type poate să fie POST și GET.

| | |
|---|---|
| Sender: | Sender-ul trimite mesaje la server. |
| Receiver: | Receiver-ul trimite un mesaj GET. Ca răspuns el primește un mesaj de la server. |
| Server: | Serverul acceptă mesajele POST, le adaugă în coadă. La venirea unui mesaj GET el trimite un mesaj din coadă ca răspuns. |

De asemenea, a fost implementată salvarea cozii pe disk, o dată la 5 secunde, și încărcarea cozii din fișier la pornirea serverului.

**Publish-Subscriber:**

Există 2 module: broker și client.
Ambele pot fi configurate din linia de comandă.
Pentru a vedea parametrii liniei de comandă, se poate folosi flagul -h (help) .
Descrierea specificațiilor mesajelor poate fi găsită în mapa /docs/specs.md

Serverul acceptă mesaje de tip PUBLISH, SUBSCRIBE. La venirea unui mesaj PUBLISH el îl transmite tuturor SUBSCRIBE-rilor ce ascultă la această coadă. Mesajul PUBLISH conține parametrul Queue, care poate avea un format REGEX (conectarea la mai multe cozi).

Clientul poate fi de tip sender sau receiver, și are rolul de a afișa în consolă mesajele trimise / primite.

## Concluzie:

În cadrul acestei lucrări am studiat structura la Message Queue, patternul Publish-Subscriber și Message Broker. De asemenea am implementat aceste arhitecturi, conform task-urilor din sarcina laboratorui. La moment nu a fost obținut nici un rezultat, deoareace lucrarea de laborator încă nu a fost susținută la profesor. Cunoștințele obținute probabil pot fi utile în viața reală, în cadrul unui proiect back-end.