*Verteilte Systeme im Sommersemester 2021*

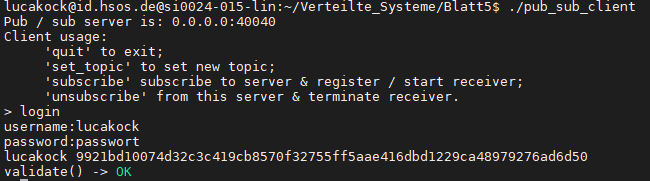
Steffen Herweg, Matr. Nr. 873475

Luca Fabio Kock, Matr. Nr. 879534 Osnabrück, 18.05.2021

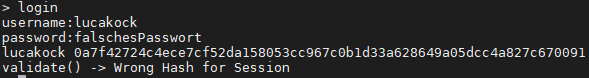
# Aufgabenblatt 5

**Tests:**

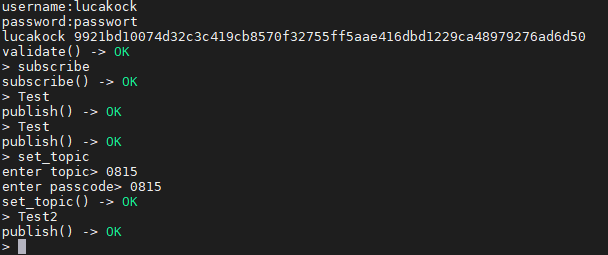
Erfolgreiche Anmeldung:

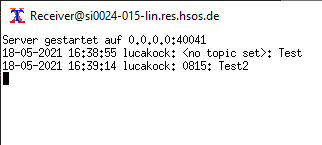


Fehlerhaftes Passwort bei Anmeldung:



Kommunikationsbeispiel:





Pub\_sub\_Client.cc:

std::string hash(std::string first, std::string second){

    std::string str = first + ";" + second;

    std::stringstream hash\_val;

    unsigned char result[SHA256\_DIGEST\_LENGTH];

    SHA256((unsigned char\*) str.c\_str(), str.length(), result);

    for (int i = 0; i < SHA256\_DIGEST\_LENGTH; i++) {

        char tmp[3];

        sprintf(tmp, "%02x", result[i]);

        hash\_val << tmp;

    }

    return hash\_val.str();

}

                // TODO: Topic fuer Server vorbereiten ...

                request.mutable\_opttopic()->set\_passcode(passcode.c\_str());

                request.mutable\_opttopic()->set\_topic(topic.c\_str());

                request.mutable\_sid()->set\_id(sid);

                request.set\_hash\_string(hash(std::to\_string(sid),digest));

                // TODO: Hashwert erzeugen

                // TODO: RPC abschicken ...

                Status status = stub\_->set\_topic(&context, request, &reply);

                // Status / Reply behandeln

                this->handle\_status("set\_topic()", status, reply);

            }

            else if (cmd.compare("subscribe") == 0)

            {

                /\* Ueberpruefen, ob Binary des Receivers existiert \*/

                if (access(receiverExecFile, X\_OK) != -1)

                {

                    /\* Receiver starten \*/

                    if ((rec\_pid = fork()) < 0)

                    {

                        std::cerr << "Cannot create process for receiver!\n";

                    }

                    else if (rec\_pid == 0)

                    {

                        /\* Der Shell-Aufruf \*/

                        /\* xterm -fa 'Monospace' -fs 12 -T Receiver -e ...pub\_sub\_deliv \*/

                        /\* kann nicht 1:1 uebertragen werden. Bei Aufruf via exec() \*/

                        /\* verhaelt sich das Terminal anders. \*/

                        /\* Alternative: Aufruf von xterm ueber ein Shell-Skript. \*/

                        /\* Allerdings haette man dann 2 Kind-Prozesse. \*/

                        execl("/usr/bin/xterm", "Receiver", "-fs", "14", receiverExecFile, (char \*)NULL);

                        /\* -fs 14 wird leider ignoriert! \*/

                        exit(0); /\* Kind beenden \*/

                    }

                    /\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                    /\* Platzhalter wie oben lokal erstellen ... \*/

                    ClientContext clientContext;

                    PubSubParam request;

                    ReturnCode response;

                    // TODO: Receiver Adresse setzen ...

                    request.mutable\_optaddress()->set\_ip\_address(get\_receiver\_ip());

                    request.mutable\_optaddress()->set\_port(40041);

                    request.mutable\_sid()->set\_id(sid);

                    request.set\_hash\_string(hash(std::to\_string(sid),digest));

                    // TODO: RPC abschicken ...

                    Status status = stub\_->subscribe(&clientContext, request, &response);

                    // TODO: Status / Reply behandeln ...

                    this->handle\_status("subscribe()", status, response);

                    if(!(response.value() == pubsub::ReturnCode\_Values\_OK) && rec\_pid > 0){

                        if (kill(rec\_pid, SIGTERM) != 0)

                            std::cerr << "Cannot terminate message receiver!\n";

                        else

                            rec\_pid = -1;

                    }

                }

                else

                {

                    std::cerr << "Cannot find message receiver executable!\n";

                    std::cerr << "Press <return> to continue";

                    char c = getc(stdin);

                    continue;

                }

            }

            else if ((cmd.compare("quit") == 0) ||

                     (cmd.compare("unsubscribe") == 0))

            {

                /\* Receiver console beenden \*/

                if (rec\_pid > 0)

                {

                    if (kill(rec\_pid, SIGTERM) != 0)

                        std::cerr << "Cannot terminate message receiver!\n";

                    else

                        rec\_pid = -1;

                }

                /\* Bei quit muss ebenfalls ein unsubscribe() gemacht werden. \*/

                /\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                /\* Platzhalter wie oben lokal erstellen ... \*/

                ClientContext clientContext;

                PubSubParam request;

                ReturnCode response;

                // TODO: Receiver Adresse setzen ...

                request.mutable\_optaddress()->set\_ip\_address(get\_receiver\_ip());

                request.mutable\_optaddress()->set\_port(40041);

                request.mutable\_sid()->set\_id(sid);

                request.set\_hash\_string(hash(std::to\_string(sid),digest));

                // TODO: RPC abschicken ...

                Status status = stub\_->unsubscribe(&clientContext, request, &response);

                // TODO: Status / Reply behandeln ...

                this->handle\_status("unsubscribe()", status, response);

                /\* Shell beenden nur bei quit \*/

                if (cmd.compare("quit") == 0)

                    break; /\* Shell beenden \*/

            }

            else if(cmd.compare("login") == 0){

                std::cout << "username:";

                std::string username;

                getline(std::cin,username);

                std::cout << "password:";

                std::string password;

                getline(std::cin, password);

                {

                    ClientContext clientContext;

                    UserName request;

                    SessionId response;

                    request.set\_name(username.c\_str());

                    Status status = stub\_->get\_session(&clientContext, request, &response);

                    sid = response.id();

                    digest = hash(username,password);

                    std::cout << username << " " << digest << std::endl;

                }

                {

                    ClientContext clientContext;

                    PubSubParam request;

                    ReturnCode response;

                    request.mutable\_void\_();

                    request.mutable\_sid()->set\_id(sid);

                    request.set\_hash\_string(hash(std::to\_string(sid),digest));

                    Status status = stub\_->validate(&clientContext, request, &response);

                    this->handle\_status("validate()",status,response);

                }

            }

            else  /\* kein Kommando -> publish() aufrufen \*/

            {

                /\* TODO: Hier den Request verschicken und Ergebnis auswerten! \*/

                /\* Platzhalter wie oben lokal erstellen ... \*/

                ClientContext clientContext;

                PubSubParam request;

                ReturnCode response;

                // TODO: Message setzen ...

                request.mutable\_optmessage()->set\_message(cmd.c\_str());

                request.mutable\_sid()->set\_id(sid);

                request.set\_hash\_string(hash(std::to\_string(sid),digest));

                // TODO: RPC abschicken ...

                Status status = stub\_->publish(&clientContext, request, &response);

                // TODO: Status / Reply behandeln ...

                this->handle\_status("publish()", status, response);

Pub\_Sub\_server.cc:

std::string hash(std::string digest, int sessionId){

    std::string str = std::to\_string(sessionId) +";"+ digest;

    std::stringstream hash\_val;

    unsigned char result[SHA256\_DIGEST\_LENGTH];

    SHA256((unsigned char\*) str.c\_str(), str.length(), result);

    for (int i = 0; i < SHA256\_DIGEST\_LENGTH; i++) {

        char tmp[3];

        sprintf(tmp, "%02x", result[i]);

        hash\_val << tmp;

    }

    return hash\_val.str();

}

  Status subscribe(ServerContext \*context, const PubSubParam \*request,

                   ReturnCode \*reply) override

  {

    int sid = request->sid().id();

    try{

      std::string username = validSessions.at(sid);

      std::string digest = hashes.at(username);

      std::string expectedHash = hash(digest,sid);

      if(expectedHash == request->hash\_string()){

        std::string receiver = stringify(request->optaddress());

        bool created = subscribers.emplace(receiver, PubSubDelivService::NewStub(grpc::CreateChannel(receiver, grpc::InsecureChannelCredentials()))).second;

        std::cout << "Created subscriber now " << subscribers.size();

        if(created){

          reply->set\_value(pubsub::ReturnCode\_Values\_OK);

        }else{

          reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_REGISTER);

        }

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

      }

    }

    catch(std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  Status unsubscribe(ServerContext \*context, const PubSubParam \*request,

                     ReturnCode \*reply) override

  {

    int sid = request->sid().id();

    try{

      std::string username = validSessions.at(sid);

      std::string digest = hashes.at(username);

      std::string expectedHash = hash(digest,sid);

      if(expectedHash == request->hash\_string()){

        std::string receiver = stringify(request->optaddress());

        int removed = subscribers.erase(receiver);

        if(removed > 0){

          reply->set\_value(pubsub::ReturnCode\_Values\_OK);

        }else{

          reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_UNREGISTER);

        }

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

      }

    }

    catch(std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  Status publish(ServerContext \*context, const PubSubParam \*request,

                 ReturnCode \*reply) override

  {

    int sid = request->sid().id();

    try{

      std::string username = validSessions.at(sid);

      std::string digest = hashes.at(username);

      std::string expectedHash = hash(digest,sid);

      if(expectedHash == request->hash\_string()){

        std::cout << "DELIVERING to " << subscribers.size() << std::endl;

        // TODO: Nachricht an alle Subscriber verteilen

        ClientContext clientContext;

        EmptyMessage empty;

        Message requestOut;

        requestOut.set\_message((username + ": " + topic + ": " + request->optmessage().message()));

        for (auto& subscriberPair : subscribers) {

          Status status = subscriberPair.second->deliver(&clientContext, requestOut, &empty);

          handle\_status("deliver()",status);

        }

        reply->set\_value(pubsub::ReturnCode\_Values\_OK);

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

      }

    }

    catch(std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  Status set\_topic(ServerContext \*context, const PubSubParam \*request,

                     ReturnCode \*reply) override

  {

    int sid = request->sid().id();

    try{

      std::string username = validSessions.at(sid);

      std::string digest = hashes.at(username);

      std::string expectedHash = hash(digest,sid);

      if(expectedHash == request->hash\_string()){

        if(request->opttopic().passcode().compare(PASSCODE) == 0){

          // TODO: Topic setzen und Info ausgeben

          topic = request->opttopic().topic();

          reply->set\_value(pubsub::ReturnCode\_Values\_OK);

        }

        else{

          reply->set\_value(pubsub::ReturnCode\_Values\_CANNOT\_SET\_TOPIC);

        }

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

      }

    }

    catch(std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  Status get\_session(ServerContext \*context, const UserName\* request, SessionId \*reply){

    int sessionId = clock();

    pendingSessions.emplace(sessionId, request->name());

    reply->set\_id(sessionId);

    return Status::OK;

  }

  Status validate(ServerContext \*context, const PubSubParam \* request, ReturnCode \*reply){

    int sid = request->sid().id();

    try{

      std::string username = pendingSessions.at(sid);

      if(validSessions.find(sid)==validSessions.end()){

        try{

          std::string digest = hashes.at(username);

          std::string expectedHash = hash(digest,sid);

          if(expectedHash == request->hash\_string()){

            pendingSessions.erase(sid);

            validSessions.emplace(sid,username);

            reply->set\_value(pubsub::ReturnCode\_Values\_OK);

          }else{

            reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

          }

        } catch (std::out\_of\_range ex) {

           reply->set\_value(pubsub::ReturnCode\_Values\_NO\_HASH\_FOR\_SESSION);

        }

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_USER\_ALREADY\_LOGGED\_IN);

      }

    } catch (std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  Status invalidate(ServerContext \*context, const PubSubParam \* request, ReturnCode \*reply){

    int sid = request->sid().id();

    try{

      std::string username = validSessions.at(sid);

      std::string digest = hashes.at(username);

      std::string expectedHash = hash(digest,sid);

      if(expectedHash == request->hash\_string()){

        validSessions.erase(sid);

        reply->set\_value(pubsub::ReturnCode\_Values\_OK);

      }else{

        reply->set\_value(pubsub::ReturnCode\_Values\_WRONG\_HASH\_FOR\_SESSION);

      }

    }

    catch(std::out\_of\_range ex){

      reply->set\_value(pubsub::ReturnCode\_Values\_SESSION\_INVALID);

    }

    return Status::OK;

  }

  void importHashes(){

    std::ifstream hashFile("./hashes.txt");

    if(!hashFile.is\_open()){

      std::cerr << "hashes.txt could not be opened!\n";

    }

    std::string username, digest;

    while(hashFile >> username>>digest){

      hashes.insert(std::pair<std::string,std::string>(username,digest));

    }

  }