



MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

CNG 495

FALL - 2023

STAGE 2 REPORT

Team Members: Bartu Can PALAMUT, Doğukan AKDAĞ, Nurberat GÖKMEN

ID Numbers: 2386225, 2452928, 2453231

Github Cloning URL: <https://github.com/bartucank/PhotoHocamBackEnd.git>

Table Of Contents

1. Introduction	4
1.1 Project Purpose	4
1.2 Cloud Delivery Model	4
1.3 Cloud Provider	5
1.4 Project Contribution	5
1.4.1 Tasks and Milestones	5
1.5 Project Development Environment	6
1.6 Sequence Diagrams	7
1.7 Data Flow Diagram	9
2. Progress	9
2.1 Week 5 (30 Oct -3 Nov)	9
2.2 Week 6 (5 - 9 Nov)	11
2.3 Week 7 - 8 (13 - 19 Nov)	15
2.3 Week 11 (4 - 8 Dec)	24
3. References	29

Table Of Figures

Figure 1 Sequence Diagram for Registration Process	7
Figure 2 Sequence Diagram for Send Photo Process	7
Figure 3 Sequence Diagram for Open Photo Process.....	8
Figure 4 Data flow diagram of PhotoHocam	9
Figure 5 Spring Initializr	9
Figure 6 Project Packages.....	10
Figure 7 User Entity	11
Figure 8 User Repository	11
Figure 9 UserDetails	12
Figure 10 JWT Provider / Generator.....	13
Figure 11 Authentication Filtering.....	13
Figure 12 Authentication Entry Point.....	14
Figure 13 Security Config.....	14
Figure 14 FriendRequest Entity	15
Figure 15 FriendRequest Repository	16
Figure 16 FriendRequest Repository	17
Figure 17 approveFriendRequest service	18
Figure 18 Image Entity.....	19
Figure 19 Image Repository.....	20
Figure 20 Send Image Service.....	21
Figure 21 Delete Image Service	22
Figure 22 Imagedto Class	22
Figure 23 Get Image List Service	23
Figure 24 SQLCloud	24
Figure 25 SQLCloud configs	25
Figure 26 SQLCloud auth configs.....	25
Figure 27 Connecting SQL Cloud Database via DBeaver	26
Figure 28 Database Tables	26
Figure 29 App Engine Configs.....	27
Figure 30 App Engine Configs.....	27
Figure 31 App Engine Logs.....	28
Figure 32 App Engine Deployment.....	28
Figure 33 App Engine Logs.....	29

1. Introduction

1.1 Project Purpose

Our goal is to develop a dynamic application designed for seamless photo sharing. In the application we developed, users will be able to register, and send them friend requests. If the user to whom the user sent a friend request accepts the request, a private chat will be opened between each other. In this chat, users will be able to send photos to each other. To enhance privacy, received photos can be viewed only once. We've implemented a database to store photos in base64 format until they are accessed, ensuring that the photos captured are both conveniently accessible and securely preserved. Also, in this database, we store user informations.

1.2 Cloud Delivery Model

We are planning to use Software as a Service (SaaS) for our Snapchat clone project, and the chosen provider will be Google Cloud. We've selected SaaS because we are planning to develop a java application as a back-end server. After we develop this server, we make an artifact for this java application. We will deploy this artifact to Google Cloud App Engine.

Google Cloud App Engine minimizes our involvement with infrastructure management, allowing us to concentrate exclusively on application development. This choice aligns with our project's specific focus on sending and receiving photos. On the other hand, for choosing this cloud delivery model, we are planning to develop a flutter application as a client side of this project for users. If we update an existing service in this client application installed on the users' phone, we save the users from maintenance trouble by updating it in the cloud where we installed the back-end, without the need for the users to update the application.

1.3 Cloud Provider

We are planning to use Google Cloud as a provider. We use two services of Google Cloud which are App Engine and Cloud SQL. We are going to deploy our back-end application to App Engine. Also we connect our application to Cloud SQL for database operations.

1.4 Project Contribution

We are planning for everyone to contribute an equal amount to the different parts of the project, so everyone will get an equal amount of experience from the different parts of the project and improve themselves. Also, as you can see below, our milestones and tasks for each user are listed.

1.4.1 Tasks and Milestones

ID	DESCRIPTION OF TASK	ASSIGNED PERSON
1	Preparation user authentication services. The user can register and login to the system. When a user logs in to the system, JWT token will be provided to user. With this token, users can use the application. <ul style="list-style-type: none"> • User entity should be implemented. • Services and interfaces that are written on Spring Security Documentation¹ should be implemented. • Login and Register endpoints should be implemented. 	BARTU CAN PALAMUT
2	Preparation "Friendship" services. <ul style="list-style-type: none"> • Users send a friend request to any user. Service should validate username that entered from user. If this username validate, "FriendRequest" should be created. • UserRequest entity should have sender and receiver. If the second user accept that request, this user should be added to sender user's friend list and sender user should be added to this user's friend list. • UserRequest entity should be deleted after completing process to reduce unnecessary data. 	DOĞUKAN AKDAĞ
3		

	Preparation “Image” services. <ul style="list-style-type: none"> • Users should be able to send photos to any user they are friends with. • Service that takes image, converts it to base64 format, then saves to the database needed. • If a user opens that image, this image should be deleted. So, there is a service that deleting image needed. 	NURBERAT GÖKMEN
4	Preparation onboarding screens with flutter. <ul style="list-style-type: none"> • Welcome page, login and register page of the application should be prepared. 	NURBERAT GÖKMEN
5	Preparation camera screen with flutter. <ul style="list-style-type: none"> • “Flutter Camera Plugin” 2 package should be implemented. • A camera screen should be implemented. On that screen the user can open front end back cameras. Also, the user can take a photo and send this photo to the user. 	BARTU CAN PALAMUT
6	Preparation chat screen with flutter. <ul style="list-style-type: none"> • If the user has a received image that s/he have not opened before, it should be listed on this page. • Users can open that image. 	DOĞUKAN AKDAĞ
7	Cloud configurations. Google Cloud App Engine should be configured for this app.	ALL TEAM MEMBERS
8	Deploying. This application should be deployed to this cloud.	ALL TEAM MEMBERS

Tasks with ID 1,2,3, 7 and 8 are done already. Tasks with ID 4,5,6 will be prepared in the next report.

1.5 Project Development Environment

We are planning to develop the front-end side of the project with Flutter framework which is based on Dart programming language. We are going to deploy our application on both iOS and Android. We are planning to develop the back-end side of the project with Spring boot framework which is based on Java programming language.

1.6 Sequence Diagrams

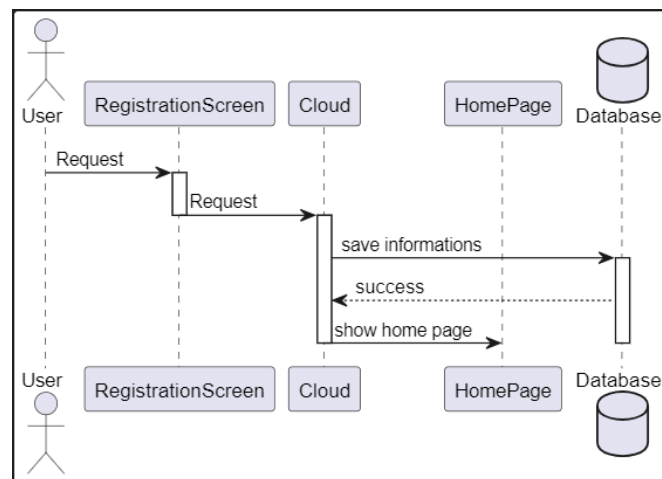


Figure 1 Sequence Diagram for Registration Process

Figure 1 shows that sequence diagram for the registration process of our project.

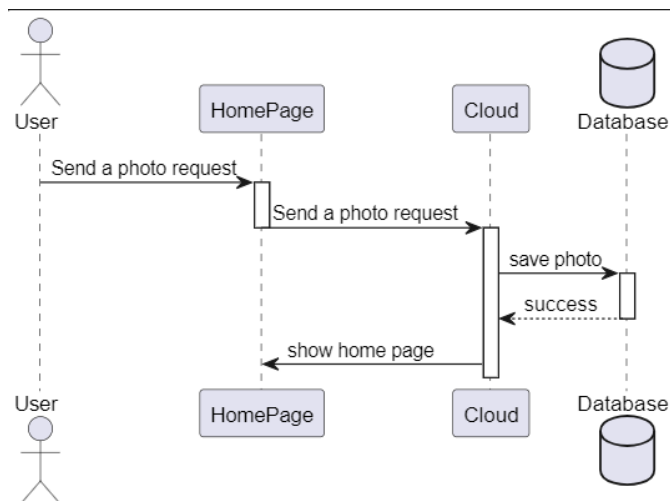


Figure 2 Sequence Diagram for Send Photo Process

Figure 2 shows that sequence diagram for sending photo process of our project.

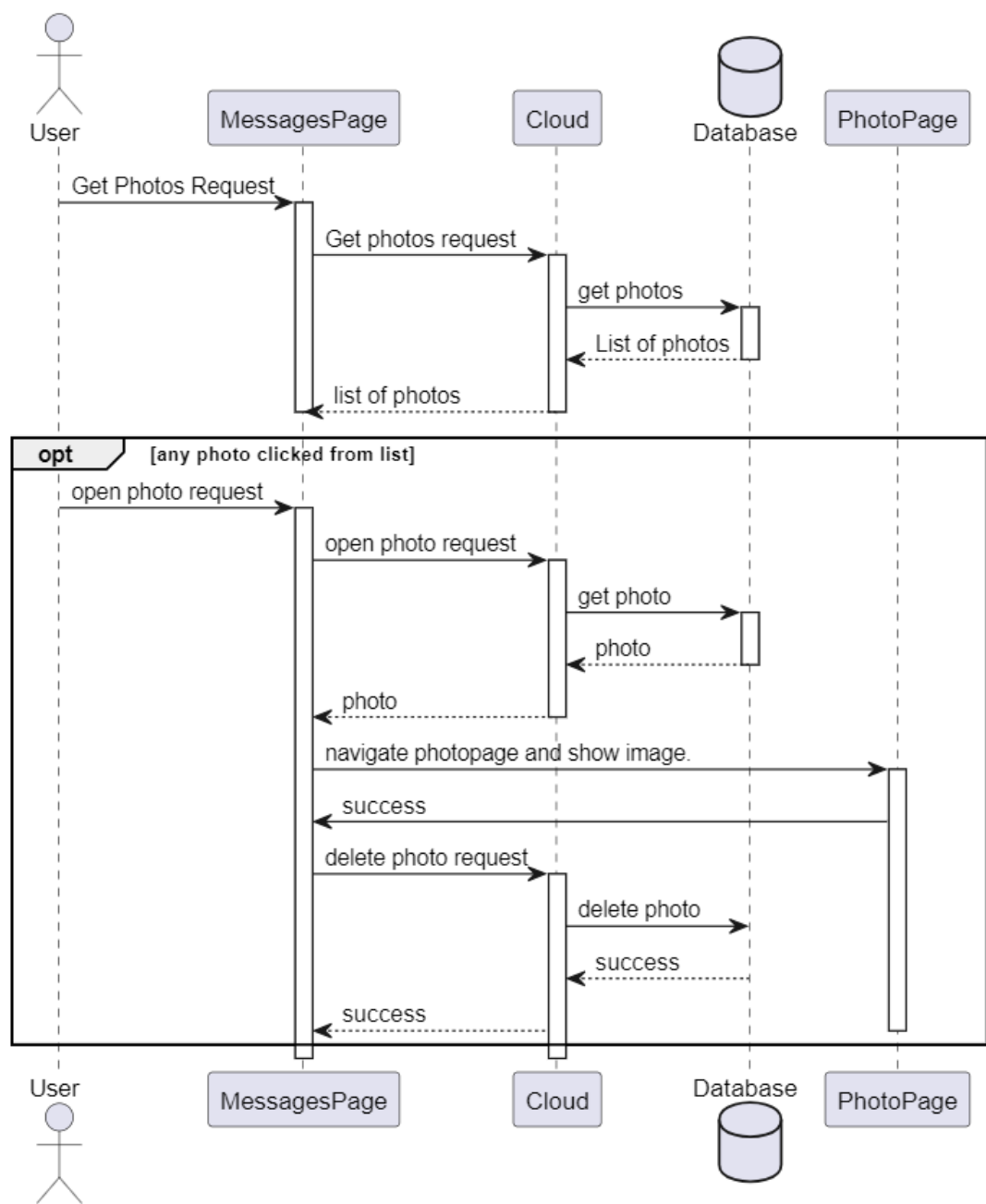


Figure 3 Sequence Diagram for Open Photo Process

Figure 3 shows that sequence diagram for opening photo process of our project.

1.7 Data Flow Diagram

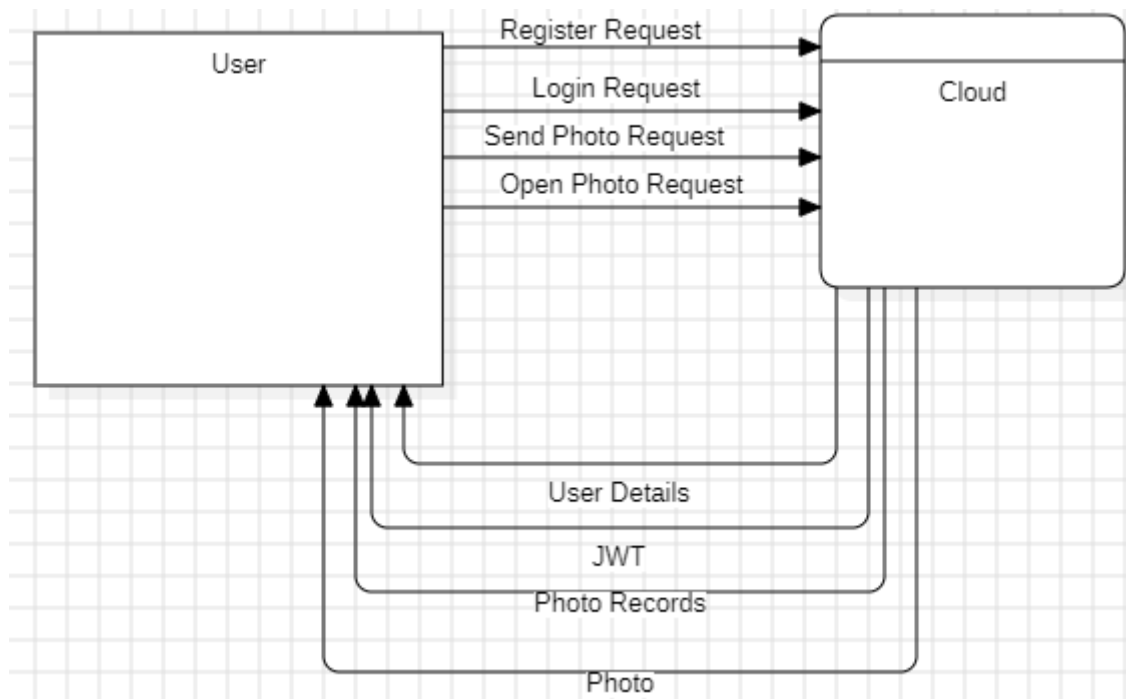


Figure 4 Data flow diagram of PhotoHocam

Figure 4 shows that context level data flow diagram of our project.

2. Progress

2.1 Week 5 (30 Oct -3 Nov)

Beginning of development, we set up basic spring boot configurations on Spring Initializr. This tool is published by Spring Framework's team. This tool basically set up necessary spring dependencies, also we can add some dependencies.

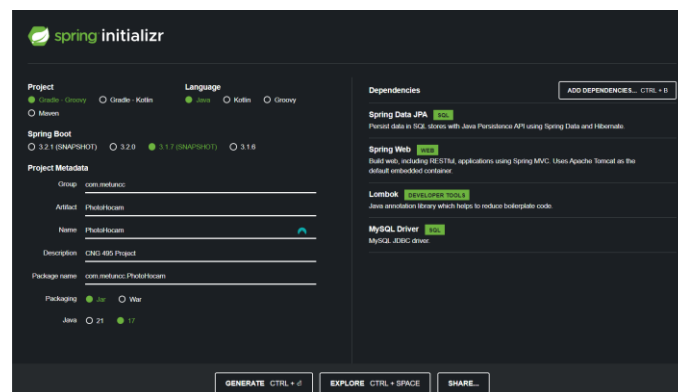


Figure 5 Spring Initializr

As you can see from Figure X, we add some dependencies for this project. Spring Data JPA will help us to SQL operations. Spring Web will help us to create web services. Lombok will help to create method, such as getters, setters, and constructors. Because we need testing locally with MySQL, we added MySQL driver to this application. Then we generate a basic project file from this page.

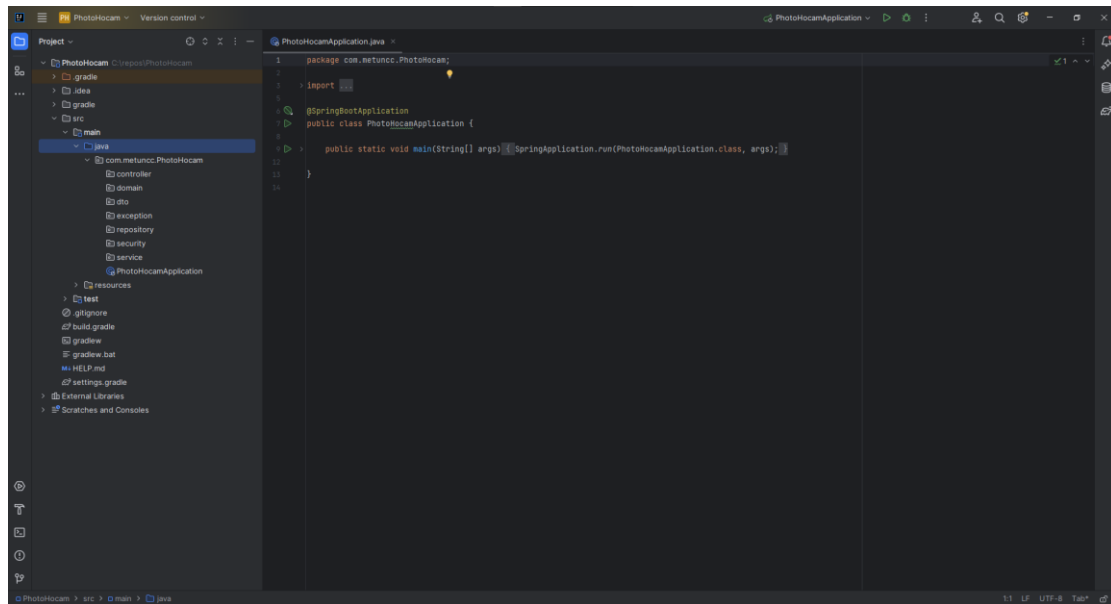
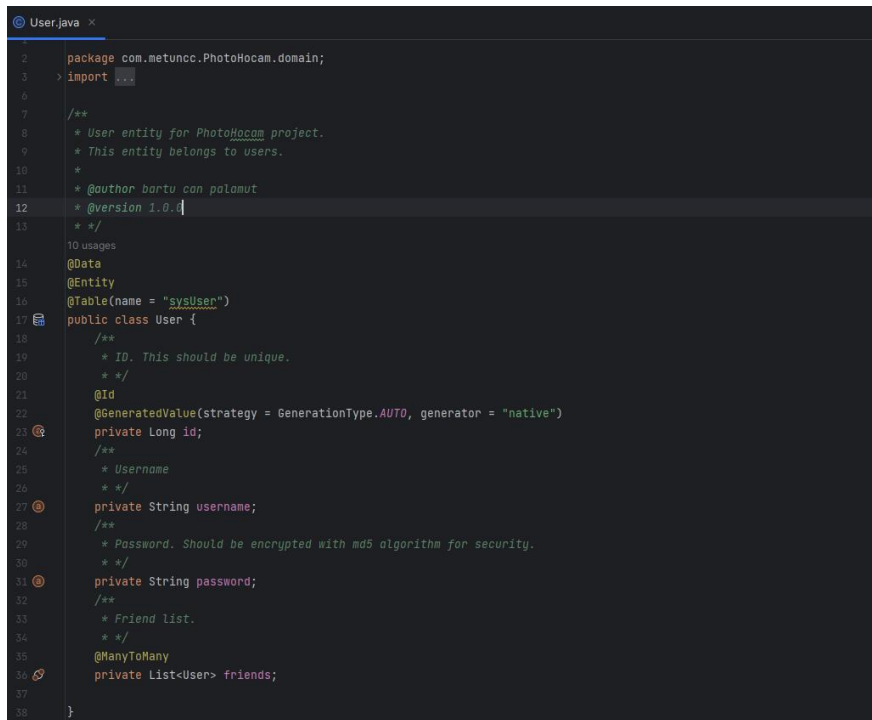


Figure 6 Project Packages

We created some packages in our project to keep it tidy. The “domain” package will contain entities that actually represents tables on database. The “controller” package will contain http requests. The “dto” package, data transfer object, package will contain basic classes. The classes in this package will actually contain objects that carry data. The “service” package will contain service class/eses. We make our services on this/these class/eses. The “exception” class will contain sample exception class for our project. We are planning to create our back-end server as transactional structured. It means that, if any error that don’t want by us happen, we want to roll back the database easily. Spring boot has some annotations. One of them, we can say to framework "Roll back when this exception throws". We are planning to roll back operation when our exception throws. The “repository” package will contain repository interfaces that help us to make database operations. Last package, “security” will contain security configs for authentication processes.

2.2 Week 6 (5 - 9 Nov)

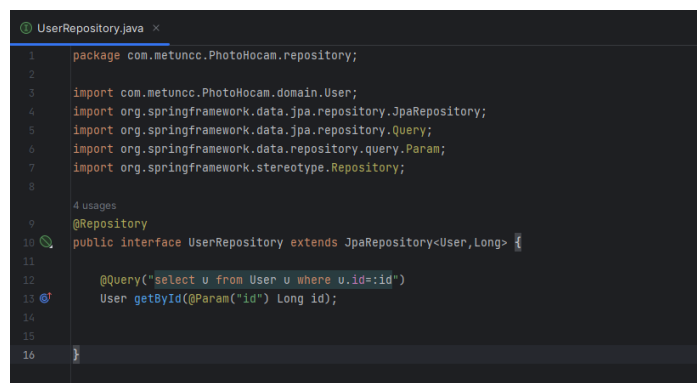
In Week 6, I created a User entity that represents users of our system.



```
1 package com.metuncc.PhotoHocam.domain;
2
3 > import ...
4
5
6
7 /**
8  * User entity for PhotoHocam project.
9  * This entity belongs to users.
10  *
11  * @author bartu can palamut
12  * @version 1.0.1
13  */
14 @Data
15 @Entity
16 @Table(name = "sysUser")
17 public class User {
18     /**
19      * ID. This should be unique.
20      */
21     @Id
22     @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
23     private Long id;
24     /**
25      * Username
26      */
27     private String username;
28     /**
29      * Password. Should be encrypted with md5 algorithm for security.
30      */
31     private String password;
32     /**
33      * Friend list.
34      */
35     @ManyToMany
36     private List<User> friends;
37 }
38
```

Figure 7 User Entity

As you can see from Figure X, on our user entity, there are “id”, “username”, “password” and “friends” attributes. We encrypt user’s password MD5 algorithm before saving to database for security. Also, as you can see above, there is an attribute called “friends”. This attribute holds a list of users whose friend of this user.



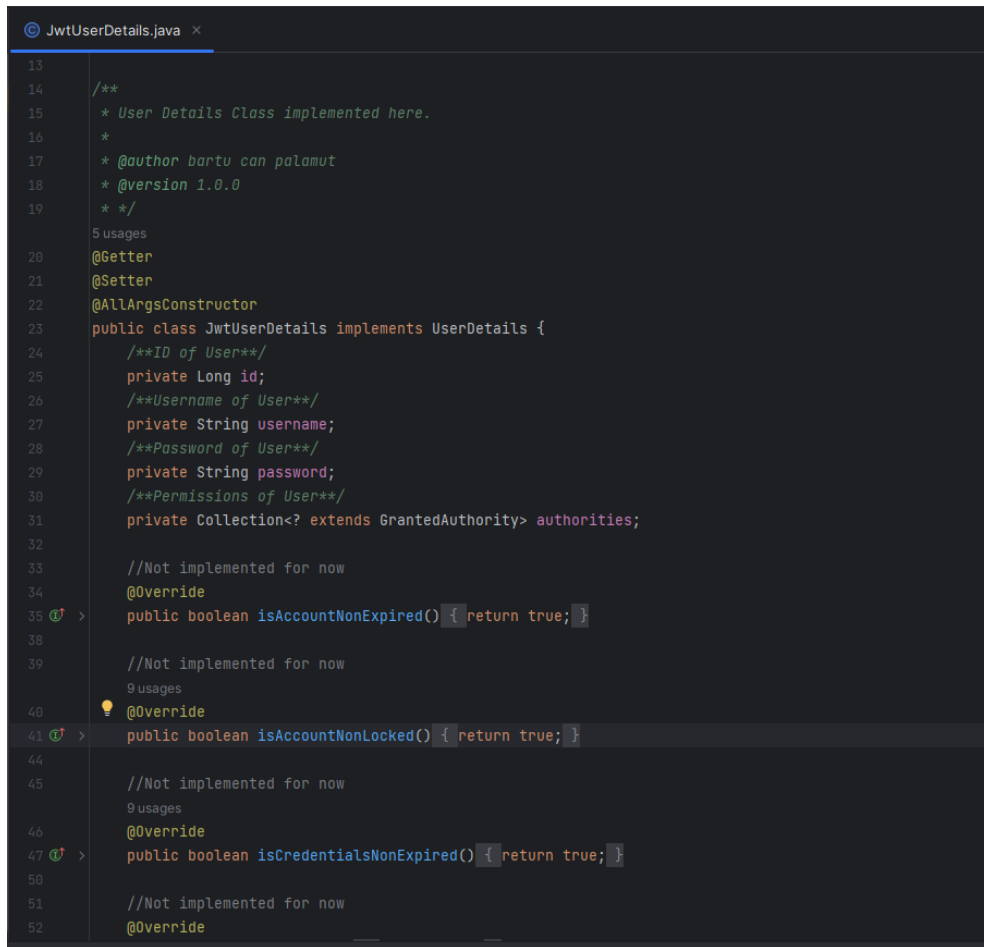
```
1 package com.metuncc.PhotoHocam.repository;
2
3 import com.metuncc.PhotoHocam.domain.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7 import org.springframework.stereotype.Repository;
8
9 4 usages
10 @Repository
11 public interface UserRepository extends JpaRepository<User, Long> {
12     @Query("select u from User u where u.id=:id")
13     User getById(@Param("id") Long id);
14
15
16 }

```

Figure 8 User Repository

After that I created a simple repository for this class, and I created a very simple SQL command that takes a specific user from the database by id.

According to the Spring Security Documentation, we need to prepare some implementations for our project for creating security rules.



```
13
14  /**
15   * User Details Class implemented here.
16   *
17   * @author bartu can palamut
18   * @version 1.0.0
19   */
20  @Getter
21  @Setter
22  @AllArgsConstructor
23  public class JwtUserDetails implements UserDetails {
24      /**ID of User**/
25      private Long id;
26      /**Username of User**/
27      private String username;
28      /**Password of User**/
29      private String password;
30      /**Permissions of User**/
31      private Collection<? extends GrantedAuthority> authorities;
32
33      //Not implemented for now
34      @Override
35      public boolean isAccountNonExpired() { return true; }
36
37      //Not implemented for now
38
39      //Not implemented for now
40      @Override
41      public boolean isAccountNonLocked() { return true; }
42
43      //Not implemented for now
44
45      //Not implemented for now
46      @Override
47      public boolean isCredentialsNonExpired() { return true; }
48
49      //Not implemented for now
50
51      //Not implemented for now
52      @Override
```

Figure 9 UserDetails

First of all, I implemented the “UserDetails” interface. This interface comes from Spring Security dependency. In this interface, there are some methods, but we don’t want to use these methods. I just want to make a simple security chain because on our system, there is no user-blocking, role based authentication. I created the “JwtUserDetails” class and I added some attributes on this class, such as id of the user, username of the user, password of the user and permission of the user. We are not planning to use role-based authentication but based on documentation, I have to implement the “getAuthorities” method, so that we add this attribute here too. I set the default role to everyone in the future. After that, I created some classes for generating, validating and filtering authentication based on Json Web Token. As you can see below, based on Spring Security, there are some operations that I already mentioned above. These operations are necessary for a simple authentication process for Spring Boot.

```

12
13  /**
14   * That class generates JWT token for specific user with SECRET and expired time (in sec).
15   */
16  4 usages 1 bartucank
17  @Component
18  public class JwtProvider {
19
20      @Value("${res.security.secret}")
21      private String SECRET;
22
23      @Value("${res.security.exp}")
24      private long EXP;
25
26  1 usage 1 bartucank
27  public String generateJwtToken(Authentication auth){
28      JwtUserDetails userDetails = (JwtUserDetails) auth.getPrincipal();
29      Date expDate= new Date(new Date().getTime() + EXP);
30      return Jwts.builder()
31          .setSubject(Long.toString(userDetails.getId()))
32          .setIssuedAt(new Date())
33          .setExpiration(expDate)
34          .signWith(SignatureAlgorithm.HS512,SECRET)
35          .compact();
36  }

```

Figure 10 JWT Provider / Generator

As you can see above, based on our configs which are secret key and expired second, json web token generates for user here.

```

14  1 bartucank
15  @Override
16  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
17      throws ServletException, IOException {
18      try {
19          String jwtToken = null;
20          String bearer = request.getHeader("Authorization");
21          if (StringUtils.hasText(bearer) && bearer.startsWith("Bearer ")){
22              jwtToken = bearer.substring("Bearer ".length() + 1);
23          }
24          Boolean isValidToken = false;
25          try {
26              Jwts
27                  .parser()
28                  .setSigningKey(SECRET)
29                  .parseClaimsJws(jwtToken);
30              isValidToken = true;
31          } catch (Exception e) {
32              isValidToken = false;
33          }
34          if (StringUtils.hasText(jwtToken) && isValidToken) {
35              Long id = Long.parseLong(
36                  Jwts
37                      .parser()
38                      .setSigningKey(SECRET)
39                      .parseClaimsJws(jwtToken)
40                      .getBody().getSubject());
41              User user = userRepository.findById(id);
42              UserDetails userDetails = user.toJwtUserDetails();
43              if (Objects.nonNull(userDetails)) {
44                  UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(user, null, userDetails.getAuthorities());
45                  auth.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
46                  SecurityContextHolder.getContext().setAuthentication(auth);
47              }
48          } catch (Exception e) {
49              return;
50          }
51      }
52      filterChain.doFilter(request, response);

```

Figure 11 Authentication Filtering

As you can see above, every request that is sent to our application is filtering here. Based on Spring Security documentation, I parsed JWT here then validated that token. Also, I matched user that are on JWT, with our database. If any error happens here, we return without setting Security Context Holder with Authentication, so that we ignore that requests.

```

1 package com.metuncc.PhotoHocam.security;
2
3 > import java.io.IOException;
4
5
6
7
8
9
10
11 /**
12  * This class represents an Authentication Entry Point for JWT authentication.
13  * Actually AuthenticationEntryPoint interface implemented here.
14  * If there is an unauthorized attempt, our service return 401 http status with is UNAUTHORIZED.
15  * @see org.springframework.security.web.AuthenticationEntryPoint
16  * */
17
18 @Component
19 public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {
20
21     @Override
22     public void commence(HttpServletRequest request, HttpServletResponse response,
23         AuthenticationException authException) throws IOException {
24         response.sendError(HttpServletResponse.SC_UNAUTHORIZED, authException.getMessage());
25     }
26 }

```

Figure 12 Authentication Entry Point

As you can see above, based on Spring Security documentation, we override commence method. If any request that is unauthorized, we return HTTP Status 401.

```

47 }
48
49 @Bean
50 public CorsFilter corsFilter() {
51     UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
52     CorsConfiguration config = new CorsConfiguration();
53     config.setAllowCredentials(false);
54     config.addAllowedOrigin("*");
55     config.addAllowedHeader("*");
56     config.addAllowedMethod("OPTIONS");
57     config.addAllowedMethod("HEAD");
58     config.addAllowedMethod("GET");
59     config.addAllowedMethod("PUT");
60     config.addAllowedMethod("POST");
61     config.addAllowedMethod("DELETE");
62     config.addAllowedMethod("PATCH");
63     source.registerCorsConfiguration(pattern: "/*", config);
64     return new CorsFilter(source);
65 }
66
67
68 @Bean
69 public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
70     httpSecurity
71         .cors().configure<HttpSecurity>()
72         .and().httpSecurity()
73         .csrf().disable()
74         .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and()
75         .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
76         .authorizeRequests().expressionInterceptUrlRegistry()
77         .antMatchers("/api/login").permitAll()
78         .antMatchers("/api/register").permitAll()
79         .antMatchers("/api/**").hasAuthority("user")
80         .anyRequest().authenticated();
81     httpSecurity.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
82     return httpSecurity.build();
83 }
84
85

```

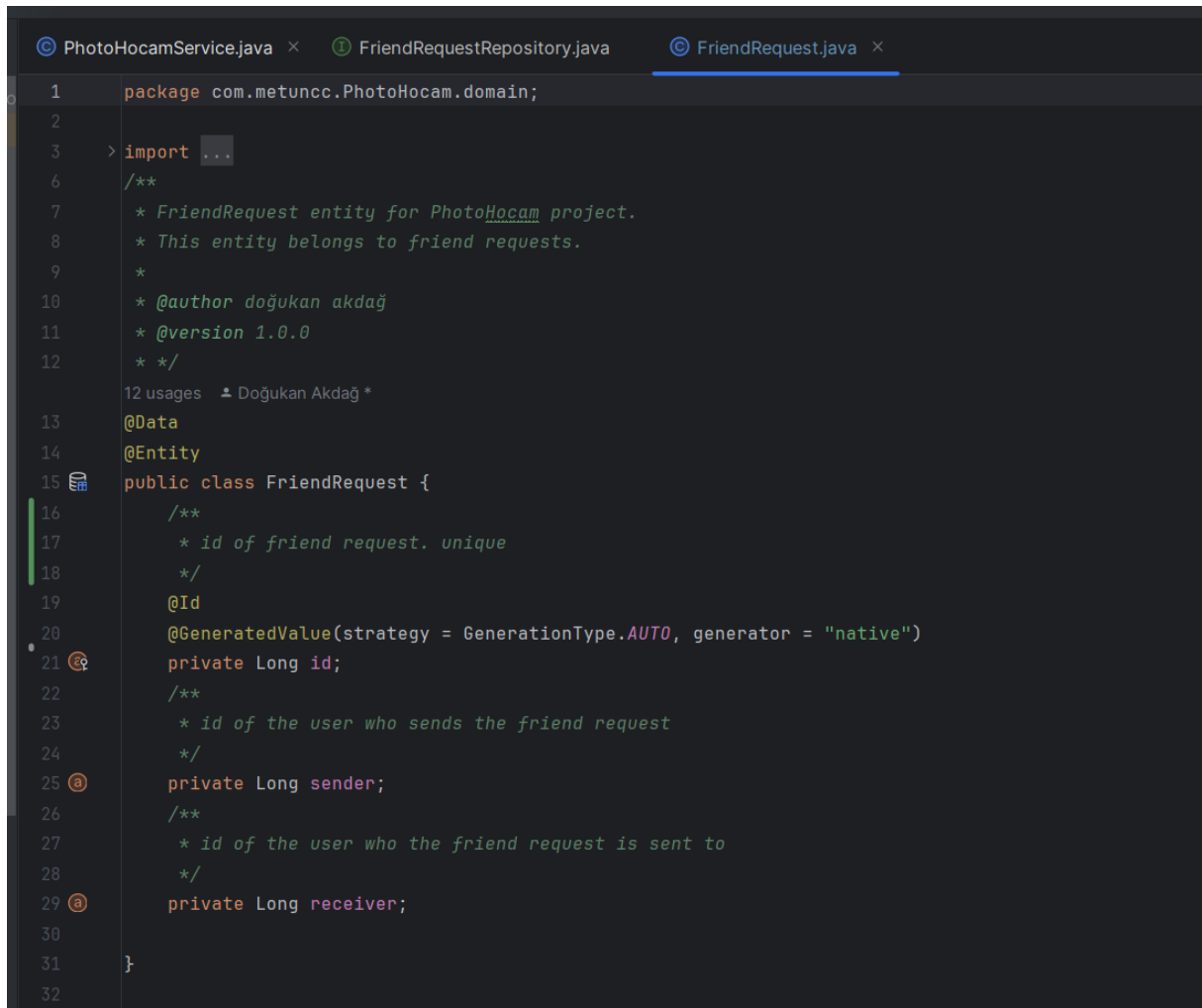
Figure 13 Security Config

Then finally I created a SecurityConfig class that actually manages requests sent by clients to our system. As you can see below, I added necessary cors configs as allowed. For example, I added methods “PUT, POST, DELETE, PATCH, GET, OPTINS, HEAD” as allowed methods. Also, I set

allowed origin and header as “*”. We are not planning to set a specific origin because we do not know whether Google will give us a dynamic url. Then I created a “filterChain” method that enables the cors configurer, enabling jwtAuthenticationEntryPoint that I created already. Also I disabled csrf because we do not have any SSL certificate. Then I set some endpoints to enable without any token that are login and register. Rest of the all endpoints need jwt token.

2.3 Week 7 - 8 (13 - 19 Nov)

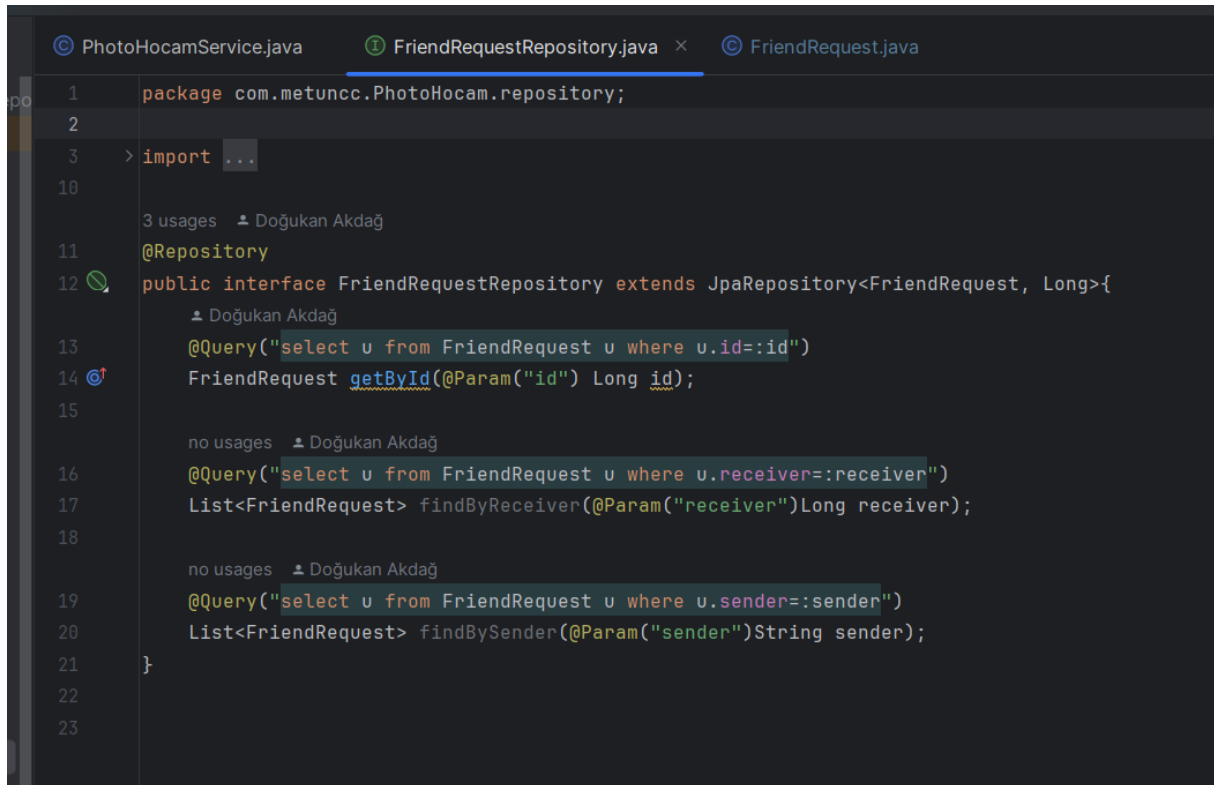
In Week 7, Nurberat and Doğukan worked separately. Bartu tested authentication services.



```
1 package com.metuncc.PhotoHocam.domain;
2
3 > import ...
4
5 /**
6  * FriendRequest entity for PhotoHocam project.
7  * This entity belongs to friend requests.
8  *
9  *
10 * @author doğukan akdağ
11 * @version 1.0.0
12 * */
13 12 usages  ⤴ Doğukan Akdağ *
14 @Data
15 @Entity
16 public class FriendRequest {
17     /**
18      * id of friend request. unique
19      */
20     @Id
21     @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
22     private Long id;
23     /**
24      * id of the user who sends the friend request
25      */
26     private Long sender;
27     /**
28      * id of the user who the friend request is sent to
29      */
30     private Long receiver;
31 }
32
```

Figure 14 FriendRequest Entity

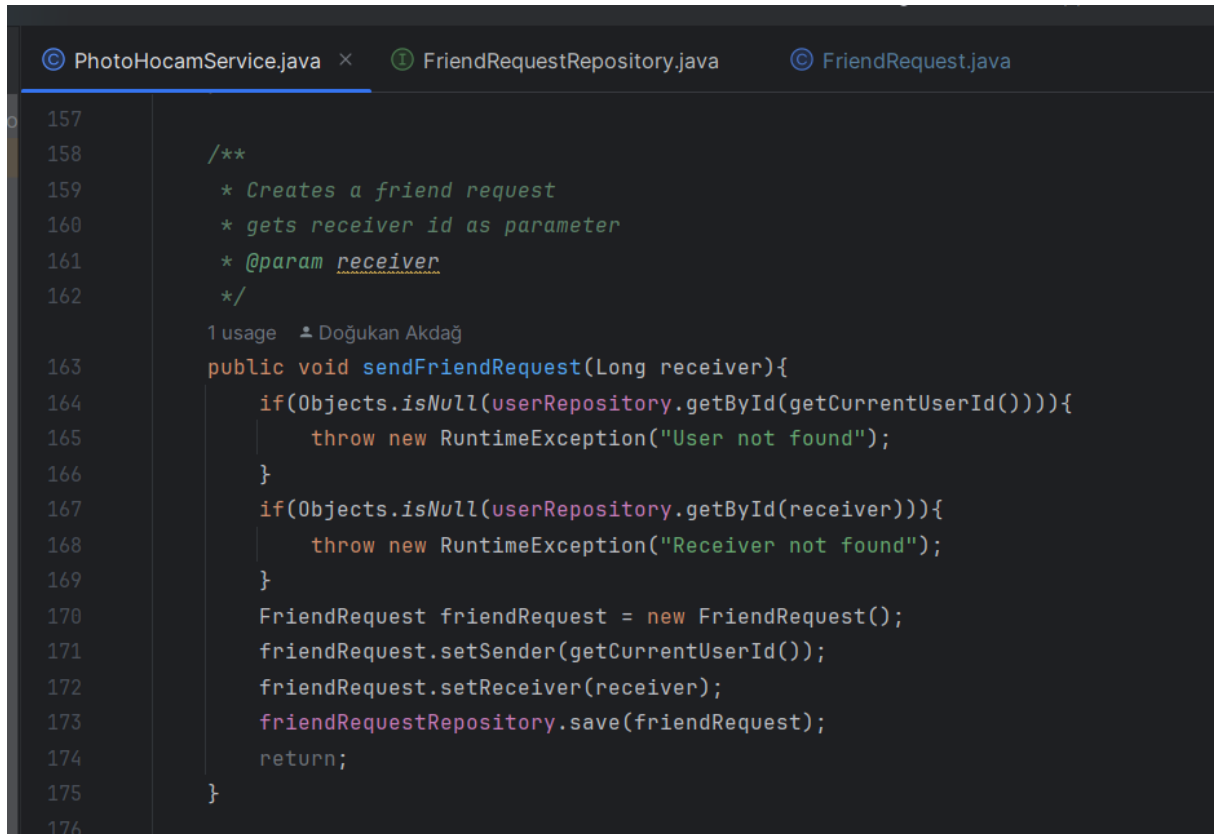
I have created a FriendRequest entity that holds an unique id, id of the user who sends the friend request and also id of the user who the friend request is sent to.



```
1 package com.metuncc.PhotoHocam.repository;
2
3 > import ...
10
11 @Repository
12 public interface FriendRequestRepository extends JpaRepository<FriendRequest, Long>{
13     @Query("select u from FriendRequest u where u.id=:id")
14     FriendRequest getById(@Param("id") Long id);
15
16     @Query("select u from FriendRequest u where u.receiver=:receiver")
17     List<FriendRequest> findByReceiver(@Param("receiver")Long receiver);
18
19     @Query("select u from FriendRequest u where u.sender=:sender")
20     List<FriendRequest> findBySender(@Param("sender")String sender);
21 }
22
23
```

Figure 15 FriendRequest Repository

I have created a FriendRequest repository for the class. Then I have created simple SQL queries to find friend requests by the id of the friend request, id of the sender and id of the receiver.



```
157
158     /**
159      * Creates a friend request
160      * gets receiver id as parameter
161      * @param receiver
162      */
163     1 usage  Doğukan Akdağ
164     public void sendFriendRequest(Long receiver){
165         if(Objects.isNull(userRepository.getById(getCurrentUserId()))){
166             throw new RuntimeException("User not found");
167         }
168         if(Objects.isNull(userRepository.getById(receiver))){
169             throw new RuntimeException("Receiver not found");
170         }
171         FriendRequest friendRequest = new FriendRequest();
172         friendRequest.setSender(getCurrentUserId());
173         friendRequest.setReceiver(receiver);
174         friendRequestRepository.save(friendRequest);
175         return;
176     }
```

Figure 16 FriendRequest Repository

I have created a service for sending friend requests. It takes the id of the receiver as a parameter. Id of the sender is acquired using the `getCurrentUserId` service that Bartu wrote which gets the user id from the authentication. After checking if the sender and receiver users exist in the repository a friend request gets created and saved to the repository. Otherwise an exception is thrown.

```

169      * adds sender and receiver to each others friend list
170      * deletes friend request
171      * @param id
172      */
173      public void approveFriendRequest(Long id){
174          FriendRequest friendRequest = friendRequestRepository.getById(id);
175          if(Objects.isNull(friendRequest)){
176              throw new RuntimeException("Friend request not found");
177          }
178          User sender = userRepository.getById(friendRequest.getSender());
179          if(Objects.isNull(sender)){
180              throw new RuntimeException("User not found");
181          }
182          User receiver = userRepository.getById(friendRequest.getReceiver());
183          if(Objects.isNull(receiver)){
184              throw new RuntimeException("User not found");
185          }
186          if(Objects.isNull(sender.getFriends())){
187              sender.setFriends(new ArrayList<>());
188          }
189          sender.getFriends().add(receiver);
190          userRepository.save(sender);
191
192          // if(Objects.isNull(receiver.getFriends())){
193          //     receiver.setFriends(new ArrayList<>());
194          // }
195          // receiver.getFriends().add(sender);
196          // userRepository.save(receiver);
197
198          friendRequestRepository.delete(friendRequest);
199      }

```

Figure 17 approveFriendRequest service

Then I finally created a service that approves friend requests. It takes the id of the friend request as a parameter. First I check if the friend request with the parameter id exists in the repository. Then I check if the sender and receiver exist in the user repository and throw an exception if they don't. After pulling the users from the user repository using the sender and receiver ids, if the user's friend lists are empty it creates a new empty list for them. Then it adds both users to each other's friend list. Finally it deletes the friend request from the repository. But after testing we encountered a bug that will put our system in a loop which we explained below in week 11. To prevent it we removed the other half of our adding friend operations as you can see in the commented out lines.

```

* @author Nurberat Gökmen
* @version 1.0.0
*/
13 usages  nurgokmen *
@Entity
@Data
public class Image {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
    /**
     * unique id for each image
     */
    private Long id;

    /**
     * user who sends the image
     * Every user can send an image to more than one receiver
     */
    @ManyToOne
    private User sender;

    /**
     * user who receives the image
     * each user can receive an image from more than one friend
     */
    @ManyToOne
    private User receiver;

    @Lob
    @Type(type = "org.hibernate.type.ImageType")
    /**
     * to store the binary data of the image. for large objects
     */
    private byte[] data;
}

```

Figure 18 Image Entity

I have created an Image entity. Image entity holds a private and unique id. A user type receiver and sender. Also, I used @Lob in order to handle large sizes of data. In the first weeks I used receiver and sender as Long type because we haven't yet opened the git repository so I wasn't able to access the User entity Bartu has created. But later on after opening the git repository, and merged everyone's work, so I changed the receiver and sender type from long to User.

3 usages nurgokmen

@Repository

```
public interface ImageRepository extends JpaRepository<Image, Long> {
```

nurgokmen

```
@Query("select i from Image i where i.id=:id")
```

```
Image getById(@Param("id") Long id);
```

1 usage nurgokmen

```
@Query("select i from Image i where i.receiver.id=:user")
```

```
List<Image> getImages(@Param("user") Long user);
```

Figure 19 Image Repository

I have created an Image repository which extends the JpaRepository. In this repository, I included two SQL queries: One query is for getting a single image from its unique id and the second query is for obtaining the list of images sent to a specific user. The list of images is retrieved by its receiver's id.

```

    * @return true if sent operation is successful, false if receiver not found
    */
    1 usage  2 nurgokmen *
    public boolean sendImage(MultipartFile file, Long userID) {

        User receiver = userRepository.getById(userID);
        User sender = userRepository.getById(getCurrentUserId());

        if (receiver == null) {
            return false;
        }
        try {
            Image img = new Image();
            var deflater = new Deflater();
            deflater.setLevel(Deflater.BEST_COMPRESSION);
            deflater.setInput(file.getBytes());
            deflater.finish();

            ByteArrayOutputStream outputStream = new ByteArrayOutputStream(file.getBytes().length);
            byte[] tmp = new byte[4*1024];
            while (!deflater.finished()) {
                int size = deflater.deflate(tmp);
                outputStream.write(tmp, 0, size);}
            try {
                outputStream.close();
            } catch (Exception e) {
            }
            img.setData( outputStream.toByteArray());
            img.setReceiver(receiver);
            img.setSender(sender);
            imageRepository.save(img);
        } catch (IOException e) {
            return false;
        }

        return true;    }

```

Figure 20 Send Image Service

I have created a sendImage service which takes a multipartFile which is an image in our case and a long id, which is the unique id of the receiver. This method handles the sending of an image operation. The method checks the sender and the receiver by getting their ids from the user repository. If the id given as parameter in this method is not matching with any of the ids in the user repository, the operation can not proceed. Then, I created an Image object to represent the file in operation. I used deflater in order to compress the file. After compression the image bytes are written into the ByteArrayOutputStream. Later on, I sent the image data, receiver information and sender information to the Image repository. I used try catch block so if any exception occurs during this process, the method will return false.

```

/**
 * deletes the image specified by its ID
 * @param id
 */

no usages  🧑 nurgokmen
public void deleteImage(Long id){

    imageRepository.deleteById(id);

}

```

Figure 21 Delete Image Service

I created a delete Image method in order to delete the images by their ids. This method takes a long id which is the id of the image as a parameter.

```

/**
 * this class is used for transferring the data of the image within the application
 */

8 usages  🧑 nurgokmen
@Data
public class Imagedto {

    /**
     * username of the user sent the image
     */
    private String username;

    /**
     * list of the images sent from the specified user
     */
    private List<Image> imageList;
}

```

Figure 22 Imagedto Class

This java class I implemented is a data transfer object class. It contains a username and a list of images. It contains the images sent by the specified username and the usernames which sent the images.

```

/**
 * gets a list of the image objects sent by a user. Based on the sender's username.
 * @return list of Imagedto, containing the username of the sender and the images they sent
 */
1 usage  nurgokmen
public List<Imagedto> getImageList(){

    Long me = getCurrentUserId();
    List<Image> imgs = imageRepository.getImages(me);
    List<Imagedto> result = new ArrayList<>();
    for (Image img : imgs) {

        Boolean check = false;
        for (Imagedto imagedto : result) {
            if(imagedto.getUsername().equals(img.getSender().getUsername())){
                check = true;
                imagedto.getImageList().add(img);
            }
        }
        if(!check){
            Imagedto imagedto = new Imagedto();
            imagedto.setUsername(img.getSender().getUsername());
            imagedto.setImageList(new ArrayList<>());
            imagedto.getImageList().add(img);
        }
    }

    return result;
}

```

Figure 23 Get Image List Service

I have created a getImageList service. This method is used to get the list of images sent by the currently logged in user. It returns the list of images from the Image Repository sent by the current user's username. This method iterates through the images and adds the image found to the list checking by the sender's username. If the sender's username is included in the list then the method adds the current image to the Imagedto list. If the sender's username is not included in the list then the method creates a new Imagedto for that sender.. The method returns a list of Imagedto objects, objects contain the list of images and the username of the sender for that images.

2.3 Week 11 (4 - 8 Dec)

In Week 11, we met and opened a git repository. After that, we pushed codes that were already written separately. Also, since we did not open a git repository in the previous weeks, we corrected the values we assumed. For example, Bartu prepared user services, but Nurberat cannot access that entity, so Nurberat put attributes to Image class without User, she put “Long” value for sender and receiver. After that, we tested the whole project by calling some api calls on postman. When we were testing our system, we noticed some performance issues. For example, when the number of users and the number of friendships are large on the system, our system stays in loop, and after an amount of time it gives StackOverflow. We debug our system step by step and we noticed that we create a very non performance structure on friendship relations. For example, X and Y are friends. Y added to X’s friend list, also X added to Y’s friend. If we take X or Y from the database, spring boot starts a loop that takes X’s friend list which have Y, also Y’s friend list which have X. For fixing this issue, we remove half of the friendship operations. Current system; let X send a friendship request to Y and Y accept that request, Y will be added to X’s friend list. When Y takes its own friend list, our system will return its own friend list with other users who added Y to their own friend list. Then we opened a Google cloud account.

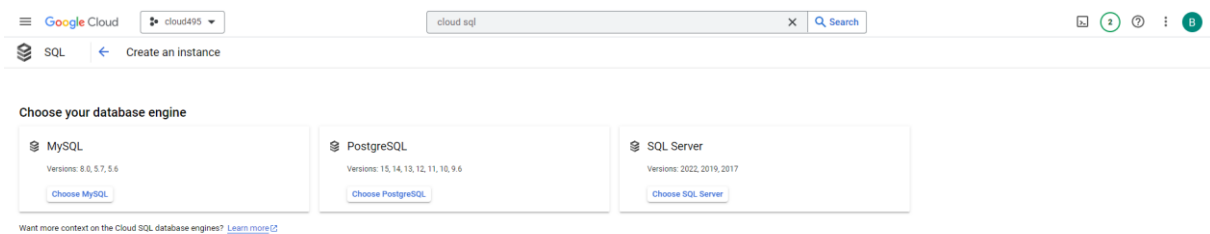


Figure 24 SQLCloud

As you can see above, there are three types of databases on GCloud. At that time we selected PostgreSQL because while developing, we are changing database MySQL to PostgreSQL.

Instance info

Instance ID *
cloudproject495

Use lowercase letters, numbers, and hyphens. Start with a letter.

Password *
[REDACTED] [GENERATE](#)

Set a password for the default admin user "postgres". [Learn more](#) [Show password](#)

PASSWORD POLICY

Database version *
PostgreSQL 15

Choose a Cloud SQL edition

A Cloud SQL edition determines foundational characteristics of your instance and cannot be changed later. Choose based on your price and performance needs. [Learn more](#)

☐ Enterprise Plus

- 99.99% availability SLA for eligible instances
- High-performance machines, up to 128 vCPUs
- Up to 35 days point-in-time recovery
- Data cache (optional)

☒ Enterprise

- 99.95% availability SLA for eligible instances
- General purpose machines, up to 96 vCPUs
- Up to 7 days point-in-time recovery

Choose a preset for this edition. Presets can be customized later as needed.

Sandbox

COMPARISON PRESETS

Choose region and zonal availability

Region: europe-west4 (Netherlands)

Pricing estimate

\$0.15 per hour (estimated, without discounts)

That's about \$3.72 per day.

Feature usage and traffic costs aren't included in estimate

SHOW COST BREAKDOWN

Summary

Cloud SQL Edition	Enterprise
Region	europe-west4 (Netherlands)
DB Version	PostgreSQL 15
vCPUs	2 vCPU
Memory	8 GB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	500 of 500
Disk throughput (MB/s)	Read: 4.8 of 240.0 Write: 4.8 of 240.0
IOPS	Read: 300 of 15,000 Write: 300 of 15,000

Figure 25 SQLCloud configs

Then we set up these configs for PostgreSQL server. As you can see above, since we are just a simple project, we set a lower package, called “Sandbox”, of GCloud. Also we selected the nearest location of Cyprus which is the Netherlands.

Connections

All instances > cloudproject495

cloudproject495

PostgreSQL 15

Instance is being updated. This may take a few minutes. While this operation is running, you may continue to view information about the instance. [VIEW ALL](#)

SUMMARY **NETWORKING** **SECURITY** **CONNECTIVITY TESTS**

Choose how you want your source to connect to this instance, then define which networks are authorized to connect. [Learn more](#)

You can use the Cloud SQL Proxy for extra security with either option. [Learn more](#)

Instance IP assignment

☐ Private IP
Assigns an internal, Google-hosted VPC IP address. Requires additional APIs and permissions. Can't be disabled once enabled. [Learn more](#)

☒ Public IP
Assigns an external, internet-accessible IP address. Requires using an authorized network or the Cloud SQL Proxy to connect to this instance. [Learn more](#)

Authorized networks

You can specify CIDR ranges to allow IP addresses in those ranges to access your instance. [Learn more](#)

[REDACTED] (Not saved)

ADD A NETWORK

Google Cloud services authorization

☐ Enable private path
Allows other Google Cloud services the BigQuery to access data and make queries over Private IP. [Learn more](#)

App Engine authorization

All apps in this project are authorized by default. You can use Cloud IAM to authorize apps in other projects. [Learn more](#)

Uploads and cloudsql operations

- Edited cloudproject495 2:52:52 PM GMT+2
- Created cloudproject495 2:52:10 PM GMT+2

Figure 26 SQLCloud auth configs

When the database is created, we add our ip address to connections because gcloud does not allow external connection. For our testing, we need to add our ip address to the connections section of gcloud.

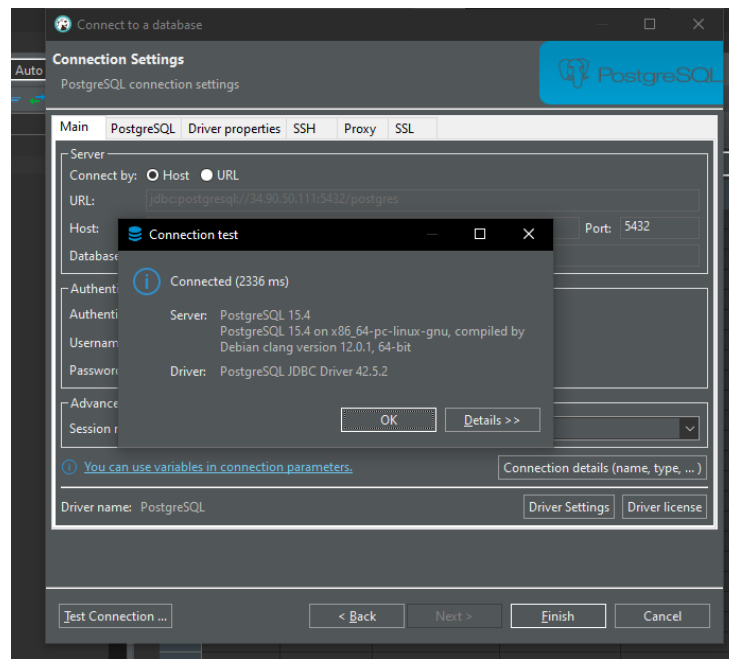


Figure 27 Connecting SQL Cloud Database via DBeaver

Then we connected the database with DBeaver. As you can see above, we connected the cloud database successfully. After that, we configured our project to connect our project to SQL Cloud. First of all, we read Google SQL Cloud documentation. Firstly, we added a necessary dependency called “spring-cloud-gcp-starter-sql-postgresql” to our project. Then we add some parameters to our application.properties file. This file sets some necessary parameters for Spring Boot when our application starts. After that we run our application for creating tables. Spring Boot & Hibernate generates tables based on entities which are created by us.

Table Name	Object ID	Owner	Tablespace	Row Count Estimate	Has Row-Level Security	Partitions	Partition by	Extra Options	Comment
friend_request	16,488	postgres	pg_default	-1	[]	[]			
image	16,493	postgres	pg_default	-1	[]	[]			
sys_user	16,500	postgres	pg_default	-1	[]	[]			
sys_user_friends	16,507	postgres	pg_default	-1	[]	[]			

Figure 28 Database Tables

As you can see above, tables were created successfully.

Then we created a Google App Engine application for deploying our application to the cloud.

Services

Versions

Instances

Task queues

Cron jobs

Security scans

Firewall rules

Quotas

Memcache


Search

Settings

Release Notes

Region

Select a region for your App Engine application. Please remember, once selected the region is permanently tied to the project.



Select a region *
europe-west6

Identity and API access

Select a service account
App Engine default service account

If no service account is selected the default App Engine service account will be used.

NEXT

Figure 29 App Engine Configs

We selected nearest location to our campus for the lowest ping.

Google Cloud

cloud495

app engine

Search

App Engine

Get started

Dashboard

Services

Versions

Instances

Task queues

Cron jobs

Security scans

Firewall rules

Quotas

Memcache

Search

Settings

Resources

Language
Java

Environment
Standard

Read App Engine Java Standard Environment [Documentation](#)

Visit [GitHub](#) for Java Standard Environment code samples.

ILL DO THIS LATER

Deploy with Google Cloud SDK

DOWNLOAD THE CLOUD SDK

Initialize your SDK
\$ gcloud init

Deploy to App Engine
\$ gcloud app deploy

Figure 30 App Engine Configs

On App Engine, application for Java applications, there are two different types of environments called Standard and Flexible. We read “App Engine Java Standard Environment Documentation” and we decided to use the standard environment because on that documentation, the standard environment has faster startup time.

Then we read GCloud Spring Framework documentation. We imported some dependencies to our application to connect to GCloud. However, based on the documentation, the given `spring-cloud-gcp-dependencies` version is 4.8.5-SNAPSHOT, but with this version we could not deploy our application, so we import another version of `spring-cloud-gcp-dependencies` which is 4.9.0.

Steps	Duration	BUILD LOG	EXECUTION DETAILS	BUILD ARTIFACTS
<div>1</div> Build Summary 3 Steps	00:01:35	<div> <input type="checkbox"/> Wrap lines <input type="checkbox"/> Show newest entries first T ↓ </div>		<div> EXPAND VIEW RAW </div>
<div>2</div> 0: fetch ~type=Manifest--location=gs://staging.cloud495-407311.ap...	00:00:02	<pre> 795 Step #2 - 'build': location: variable userDetails of type JutUserDetails 796 Step #2 - 'build': /workspace/src/main/java/com/netuncc/Photomocam/security/JutUserDetails.java:23: error: JutUserDetails is not abstract and does not override abstract method getUsername() 797 Step #2 - 'build': public class JutUserDetails implements UserDetails { 798 Step #2 - 'build': ^ 799 Step #2 - 'build': /workspace/src/main/java/com/netuncc/Photomocam/security/JutAuthenticationFilter.java:73: error: cannot find symbol 800 Step #2 - 'build': UserDetails userDetails = new JutUserDetails((user.getId(),user.getUsername()),user.getPassword()); 801 Step #2 - 'build': ^ 802 Step #2 - 'build': symbol: method getId() 803 Step #2 - 'build': location: variable user of type User 804 Step #2 - 'build': /workspace/src/main/java/com/netuncc/Photomocam/security/JutAuthenticationFilter.java:73: error: cannot find symbol 805 Step #2 - 'build': UserDetails userDetails = new JutUserDetails((user.getId(),user.getUsername()),user.getPassword()); 806 Step #2 - 'build': ^ 807 Step #2 - 'build': symbol: method getUsername() 808 Step #2 - 'build': location: variable user of type User 809 Step #2 - 'build': /workspace/src/main/java/com/netuncc/Photomocam/security/JutAuthenticationFilter.java:73: error: cannot find symbol 810 Step #2 - 'build': UserDetails userDetails = new JutUserDetails((user.getId(),user.getUsername()),user.getPassword()); 811 Step #2 - 'build': ^ 812 Step #2 - 'build': symbol: method getPassword() 813 Step #2 - 'build': location: variable user of type User 814 Step #2 - 'build': 42 errors 815 Step #2 - 'build': 816 Step #2 - 'build': FAILURE: Build failed with an exception. 817 Step #2 - 'build': 818 Step #2 - 'build': * what went wrong: 819 Step #2 - 'build': Execution failed for task ':compileJava'. 820 Step #2 - 'build': > Compilation failed; see the compiler error output for details. 821 Step #2 - 'build': 822 Step #2 - 'build': * Try: 823 Step #2 - 'build': > Run with --info option to get more log output. 824 Step #2 - 'build': > Run with --scan to get full insights. 825 Step #2 - 'build': 826 Step #2 - 'build': BUILD FAILED in 1m 7s </pre>		

Also we faced some issues. As you can see from Figure X, Lombok plugin which created our getters and setters automatically, did not work on gcloud. We add all getter and setters methods manually. After that we deployed again.

Figure 32 App Engine Deployment

```
Terminal Local (4) x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\repos\PhotoHocam> gcloud app logs tail
Waiting for new log entries...
2023-12-14 20:23:19 default[20231214t222057] 2023-12-14 20:23:19.099 INFO 12 --- [main] c.m.PhotoHocam.PhotoHocamApplication : Started PhotoHocamApplication in 9.084 seconds (JVM running for 10.829)
2023-12-14 20:27:09 default[20231214t222057] 2023-12-14 20:27:11.583 INFO 11 --- [main] c.m.PhotoHocam.PhotoHocamApplication : Starting PhotoHocamApplication using Java 17.0.8.1 on localhost with PID 11 (/worksp
"GET /favicon.ico HTTP/1.1" 503
2023-12-14 20:27:11 default[20231214t222057] 2023-12-14 20:27:11.586 INFO 11 --- [main] c.m.PhotoHocam.PhotoHocamApplication : No active profile set, falling back to default profiles: default
2023-12-14 20:27:11 default[20231214t222057] 2023-12-14 20:27:11.586 INFO 11 --- [main] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property
2023-12-14 20:27:11 default[20231214t222057] 2023-12-14 20:27:11.586 INFO 11 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-12-14 20:27:11 default[20231214t222057] 2023-12-14 20:27:11.586 INFO 11 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 108 ms. Found 3 JPA repository interface
s.
2023-12-14 20:27:12 default[20231214t222057] 2023-12-14 20:27:12.770 INFO 11 --- [main] trationDelegateBeanPostProcessorChecker : Bean 'org.springframework.security.access.expression.method.DefaultMethodSecurityExp
ressionHandler@52c3cb31' of type [org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-
proxying)
2023-12-14 20:27:13 default[20231214t222057] 2023-12-14 20:27:13.489 INFO 11 --- [main] trationDelegateBeanPostProcessorChecker : Bean 'methodSecurityMetadataSource' of type [org.springframework.security.access.met
hod.DelegatingMethodSecurityMetadataSource] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2023-12-14 20:27:13 default[20231214t222057] 2023-12-14 20:27:13.940 INFO 11 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 1010 (http)
2023-12-14 20:27:13 default[20231214t222057] 2023-12-14 20:27:13.940 INFO 11 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-12-14 20:27:13 default[20231214t222057] 2023-12-14 20:27:13.940 INFO 11 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.49]
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.041 INFO 11 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.041 INFO 11 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: Initialization completed in 2485 ms
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.396 INFO 11 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000284: Processing PersistenceUnitInfo [name: default]
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.487 INFO 11 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.32.Final
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.697 INFO 11 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5.1.2.Final)
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.843 INFO 11 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-12-14 20:27:14 default[20231214t222057] 2023-12-14 20:27:14.897 INFO 11 --- [main] c.g.cloud.sql.core.CoreSocketFactory : First Cloud SQL connection, generating RSA key pair.
2023-12-14 20:27:17 default[20231214t222057] 2023-12-14 20:27:17.066 INFO 11 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
```

Figure 33 App Engine Logs

Also we checked if our application was deployed successfully or not. As you can see above, it deployed successfully.

3. References

1. <https://docs.spring.io/spring-security/reference/index.html>
2. <https://pub.dev/documentation/camera/latest/>
3. <https://cloud.google.com/appengine/docs/legacy/standard/java>
4. https://cloud.google.com/appengine/docs/flexible/java?hl=en_US
5. <https://cloud.google.com/appengine/docs/the-appengine-environments>
6. <https://googlecloudplatform.github.io/spring-cloud-gcp/reference/html/>