CS 411 / Software Architecture Design Term Project:

"OkHttp"
Group 14
Project Final Report

Project Group Members:
Ertan ADAY
Oktay AYAR
Özgür Can ERDOĞAN
Cihangir MERCAN
Arif Can TERZİOĞLU

Course Instructor:
Bora GÜNGÖREN

Teaching Assistant:
Sinem SAV

Abstract

In this document, the reader will find problems that Java and Android developers encounter while using HTTP protocol, and the need for a technology to solve those problems. After that, the reader can find the approaches to solve the stated problems and carry out the need for a new technology to solve those problems. This new technology's name is named as OkHttp. After problem statement and approach sections, the reader can learn how the requirements are analysed in Requirements Analysis section. Then, the reader can find information about analysis of technical problems. Afterwards, in this document, the reader can find the Domain Analysis in which OkHttp is used, which are mostly technical domains.

In the Problem Statement section, the will find the problem with the use of HTTP requests in general purpose APIs. In here, the reader can find information about already existing technologies that are used to solve those problems, and see that existing technologies cannot fully solve those problems, and different than the technologies on the market OkHttp promises to solve those problems fully.

In the approach section, the reader can find OkHttp's approach to solve the stated problems. OkHttp's approach is first analyse the requirements, then analyse the technical problems, and finally analyse the domain.

In the requirements analysis section, the reader will learn how OkHttp analysed the requirements, and see that the following steps are taken: The stakeholders are identified, and found that stakeholders are mostly developers, or people that are closely related to software development. That means, the stakeholders are mainly software developers, software engineers, software architects, and software testers. After stakeholder analysis, the reader can find functional and nonfunctional requirements that are determined by regarding the stakeholders. In the following subsections, the reader can find use case scenarios, and related diagrams to understand the uses of OkHttp.

In Technical Problems Analysis section, the reader can find the technical problems related with OkHttp, such as authentication, privacy, response time, maintaining sustainability, and maintaining reliability.

In Domain Analysis section, the reader can find solution domains and the usage of them in order to solve the technical problems stated in previous section. In addition, the reader can find related domain model, along with related architectural views and styles. In summary, module view, component & connector view, and allocation views are used.

Next, the reader can find Design Patterns that is used for OkHttp. There are two related design patterns: Observer Pattern and Façade Pattern.

After all of those, the reader can find the evaluation technique for OkHttp's architecture. The ATAM (Architectural Tradeoff Analysis Method) is used for evaluation of OkHttp's architecture.

1. Problem Statement

In today's world, a number of general-purpose APIs for making HTTP requests are provided. Through these APIs, developers can manage downloading, making simple HTTP and HTTPS requests, or precisely tune your request to the specific requirements of your server infrastructure. When dealing with these cases, developers encounters problems regarding
HTTP API's.

One of widely used library is Apache API and it is hard to use because functions like cache and compression is limited or hard to achieve tasks. Besides, Android developers cannot make request through UI thread due to restriction and software environmental conditions.
Doing HTTP efficiently makes applications load faster and saves bandwidth. Therefore, the need to use a library like OkHttp is emerging. OkHttp is a third party API developed by Square for sending and receive HTTP-based network request [3]. OkHttp makes request tasks and multi-threading tasks easier. OkHttp allows all requests to the same host the share a socket. It reduces request latency, shrinks download sizes and response caching prevents the repeated requests [2]. OkHttp handles the network problems related to connection from the behind.

In short, the main purpose of OkHttp is to provide easy to use request/response API for Java and Android developers with fluent builders and immutability. With both synchronous blocking calls and async calls with callbacks, it brings solutions to Android developers thread problems and with extensive cache and compression functions make API easy to use.

2. Approach

Our approach for OkHttp's architecture design will be done in three steps:

- Requirement Analysis,
- Technical Problem Analysis,
- Domain Analysis

Requirement analysis is the first step for software development design process. Requirement analysis is determining the important parts of the project. In requirement analysis, communication with the client determines the requirements of the system. In requirement analysis step, we define stakeholders, functional and nonfunctional requirements. These requirements are specified for the stakeholders according to their needs. Then, we define use case scenarios and use case model accordingly to clarify the functionality of the system. Next, we will show dynamic model with sequence diagrams.After dynamic model, technical problem analysis will define and address the technical problems about Okhttp. Each problem will be defined and then evaluated according to Okhttp. Finally, in domain analysis, we will describe identification of the domains. Then we will describe the knowledge sources which will be evaluated in this step.

3. Requirements Analysis:

3.1- Stakeholder Identification

3.1.1- Software Developers (Users)

They are the main users of the system. Software developers who develops a software on Android and Java.

Concerns of the Software Developer:

· Performance and Stability:

Software developers will expect to use the library with high response speed. Respond and request speed performance should be stable and fast.

· Usability:

Software developers will expect to learn library quickly. They will want good documentation about the library.

· Content Update:

Software developers desires to use a library which gets regular content updates which makes the system catch up with new technologies.

3.1.2- Software Engineers (Developer of the Okhttp)

The software engineers implement the system. They are responsible to implement the system according to needs of the stakeholders.

Concerns of the Software Developer:

· Performance of the System:

Software engineers will want to create a library with high performance.

· Reliability:

Software engineers should create library without any problems. They want to get rid of all the problems that are affecting the performance of the system.

3.1.3- Software Architect (Developer of the Okhttp)

The Software architects are responsible for designing the software architecture of the system. Concerns of the Software Architecture:

· Ease of integration:

Software architects wants to create a design which allows the software engineers to add new contents or make update to the system.

· Flawless Design:

The software architects want to make a design which satisfies all the needs of stakeholders.

· Usability:

Software architects want to create a design which is easy to understand for the engineers of the system.

3.1.4- Tester (Developer of the Okhttp)

They are responsible for testing all the functionalities of the system. They are responsible for executing test cases and its results.

3.2- Functional Requirements

- OkHttp can be used in Android and Java applications by either downloading the jar and adding it to the project build path or adding the dependency to the gradle/maven.
- Through an OkHttpClient, User can make a GET request and retrieve the remote data.
- User can add query parameters to the GET request URL.
- User can make a POST request and process the data to the identified resource.
- User can provide the data in the body of the request as a plain text, json or xml.
- User can also make a PUT, DELETE, PATCH and HEAD requests through request builder.
- All requests return a response to a User with a code (like 200 for success or 404 for not found), headers, and its own optional body.
- User can add headers to the requests (like specifying return type as json).
- Through calls, user can make a synchronous request which blocks the thread until the response is readable.
- User can make an asynchronous request which enqueues the request on any thread and get called back on another thread when the response is readable.

- User can attach tag to the requests. It can be used later to cancel the requests with the specific tag.
- User can cache requests and responses with the cache directory in which user can write a limit on the cache's size.

3.3- Non-functional Requirements

3.3.1- Efficient

HTTP is the way modern applications network. It's how we exchange data & media. Doing HTTP efficiently makes your stuff load faster and saves bandwidth.
OkHttp is an HTTP client that's efficient by default:

- HTTP/2 support allows all requests to the same host to share a socket.
- Connection pooling reduces request latency (if HTTP/2 isn't available).
- Transparent GZIP shrinks download sizes.
- Response caching avoids the network completely for repeat requests.

3.3.2- Easy

Using OkHttp is easy. Its request/response API is designed with fluent builders and immutability. It supports both synchronous blocking calls and async calls with callbacks.

3.3.3- Fast

OkHttp perseveres when the network is troublesome: it will silently recover from common connection problems. If your service has multiple IP addresses OkHttp will attempt alternate addresses if the first connect fails. This is necessary for IPv4+IPv6 and for services hosted in redundant data centers. OkHttp initiates new connections with modern TLS features (SNI, ALPN), and falls back to TLS 1.0 if the handshake fails. Also, it uses Okio for fast I/O and resizable buffers.

3.3.4- Secure

Security protocols, which have been developed in order to provide security for http requests, such as SSL and TLS are implemented by OkHttp.

3.4 Use Case Scenarios

Use-Case 1

GET Request at Java Maven project

Primary Actor: Bora

Main Success Scenario

1. Bora starts a new Java Maven Project at Eclipse IDE.
2. Bora adds a dependency for OkHttp to the pom.xml.
3. Bora wants to get information about the movie Memento as a JSON.
4. Bora finds a RESTful web service called OMDb API (http://www.omdbapi.com).
5. Bora starts to build a GET request.
6. Bora sets the URL as a http://www.omdbapi.com.
7. Bora adds query parameters: “apikey” = “1e8f599a” and “t” = “memento”. Therefore, URL becomes http://www.omdbapi.com/?apikey=1e8f599a&t=memento.
8. Bora adds a header: “Accepts” = “application/json; q=0.5”.
9. Bora executes the request through OkHttpClient.
10. Bora successfully gets the response.
11. Bora prints out the response code and sees 200.
12. Bora prints out the response as a Java String and sees all the necessary information about Memento.

Alternate Flow

7a. Bora forgets to add apikey parameter to the URL.

 7a1. Bora does step 8 and step 9.

 7a2. Bora fails to get Memento as a JSON and sees a JSON that indicates error.

 7a3. Bora prints out the response code and sees 401.

Use-Case 2

POST Request at Java EE application

Primary Actor: Bora

Main Success Scenario

1. Bora wants to test his Java EE application.
2. Bora needs to send a POST request to a URL: http://46.101.243.149/test-post/register which accepts a JSON in the body that includes a username, an email and a password.
3. Bora prepares a JSON as a String.
4. Bora sets the URL of a request as a http://46.101.243.149/test-post/register.
5. Bora sets the request body as a media type JSON.
6. Bora executes the request through OkHttpClient.
7. Request succeeds.
8. Bora prints out the response code and sees 200.

9. Bora prints out the response as a String and sees “new record is added to database.”.

Alternate Flow

3a. Bora forgets to add an email to the JSON String.

- 3a1. Bora does steps 3, 4 and 5.
- 3a2. Request succeeds.
- 3a3. Bora prints out the response code and sees 200.
- 3a4. Bora prints out the response as a String and sees “email is not set.”.

<u>**Use-Case 3**</u>

PUT Request to Elastic Search

Primary Actor: Bora

Main Success Scenario

1. Bora wants to do mapping for his index at the Elastic Search.
2. Bora needs to send a PUT request to a URL: http://localhost:8080/my_index/ which accepts a JSON that includes mappings.
2. Bora prepares a JSON as a String like the example at here: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html.
3. Bora sets the URL of a request as a http://localhost:8080/my_index/.
4. Bora sets the request body as a media type JSON.
5. Bora executes the POST request through an OkHttpClient.
6. Request succeeds.
7. Bora prints out the response code and sees 200.

Alternate Flow

-

<u>**Use-Case 4**</u>

DELETE Request to Elastic Search

Primary Actor: Bora

Main Success Scenario

1. Bora has a collection of tweets at: http://localhost:8080/twitter/_search.
2. Bora wants to delete the first tweet which is in http://localhost:8080/twitter/tweet/1.
2. Bora sets the URL as a http://localhost:8080/twitter/tweet/1.
5. Bora executes the DELETE request without a body through an OkHttpClient.
6. Request succeeds.
7. Bora prints out the response code and sees 200.

Alternate Flow

-

<u>**Use Case 5:**</u>

Adding Headers to Request at Java Project

Primary Actor: Bora

Main Success Scenario

1- Bora starts a new Java project.

2- Bora creates a new request with a header.

3- Later in time, Bora decides to add multiple headers to the request.

4- Bora changes his request to a chain request through an OkHttp interceptor.

5- Bora adds an authorization header to the request.

7- Bora gets the OAuth mode, and adds that mode as a string together with authorization mode.

8- Bora builds the request through request builder wrapper class.

9- Bora prints out the response code to the console.

Alternate Flow

-

<u>**Use-Case 6**</u>

Asynchronous GET Request at Android Application

Primary Actor: Bora

Main Success Scenario

1. Bora starts an Android Project.
2. Bora creates a new TextView object.
2. Bora adds a dependency for OkHttp to the build.gradle.
3. Bora synchronously gets information about the movie Memento as a JSON.
4. Bora sets the TextView as a String JSON.
4. Bora sees that the TextView is not updated on the start of application.
5. Bora makes the request asynchronously.
6. On the callback function, when response is returned, Bora sets the TextView.
7. This time, Bora sees that the TextView is updated.

Alternate Flow

-

<u>**Use Case 7:**</u>

Adding TAG to a Request

Primary Actor: Bora

Main Success Scenario

1- Bora wants to set a tag to his request for the ease of operations to the request (like cancelling, etc.)

2- Bora sets a tag name as a string, like “dummy tag”.

3- Bora builds the request with the tag “dummy tag”.

4- After a while, Bora realizes that he needs to cancel the requests he made with “dummy tag”.

5- Bora searches all the requests with “dummy tag”, and then cancels those requests.

6- Bora builds the request through request builder wrapper class.

7- Bora prints out the response code to the console.

Alternate Flow

-

Use Case 8:

Adding CACHE CONTROL to a Request

Primary Actor: Bora

Main Success Scenario

1- Bora wants to have an eye on requests’ cache control.

2- To do that, he sets the requests’ Cache-Control header

3- Any existing Cache-Control headers already present is replaced by Bora’s Cache-Control header.

4- Bora enters Cache-Control directives.

5- Bora prints out the response code to the console.

Alternate Flow

-

3.5 Use Case Model

In the model below, we gave the use cases, the methods to use, and their relationship with each other is given. This is a model that summarizes all use cases in one model for reader to see the whole picture in one diagram.

Figure 3.5.1: User case model of system

3.6- Dynamic model

3.6.1-Sequence Diagrams

#1

GET request to get movie Memento as JSON

Primary Actor: Bora

Scenario

Bora gets the information about the movie Memento as a JSON at his project from OMDb API with the URL: http://www.omdbapi.com/?t=memento&apikey=1e8f599a.

Description

First, Bora initializes a HttpUrl.Builder object by giving URL http:/www.omdbapi.com/. Bora calls two methods on this object to add two query parameters. Therefore, he builds the final URL as a Java String. Then, with using Request.Builder, he calls two methods: url(urlString) and addHeader("Accept", "application/json; q=0.5"). Finally, he calls build() method and he creates a Request object. Then, through OkHttpClient object, he starts a new call with the request created and executes it: client.newCall(request).execute(). Hence, this call returns a Response object to the Bora. Now, Bora can see the response code with response.code() method.

Figure 3.6.1.1: Sequence diagram of use case scenario - 1

#2

POST request to register

Primary Actor: Bora

Scenario

Bora registers to a website: http://46.101.243.149/test-post/register by giving a username, an email and a password.

Description

First, by using Java String, Bora creates a JSON String which includes his username, email and password. He uses static method RequestBody.create(MediaType.JSON, jsonString.toString()) to create a RequestBody object. Then, with Request.Builder, he calls two methods: url(http://46.101.243.149/test-post/register) and post(requestBody) and builds the Request. Then, through OkHttpClient object, he starts a new call with the request created and executes it: client.newCall(request).execute(). Hence, this call returns a Response object to the Bora. Now, Bora can see the response code with response.code() method.

Figure 3.6.1.2: Sequence diagram of use case scenario - 2

#3

PUT request to Elastic Search
Primary Actor: Bora
Scenario
Bora does a mapping for his index at the Elastic Search like here:
https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html.

Description

While coding, the only difference from post is: Bora calls put(requestBody) instead of post(requestBody) on the Request.Builder this time.

Figure 3.6.1.3: Sequence diagram of use case scenario - 3

#4

DELETE request to Elastic Search

Primary Actor: Bora

Scenario

Bora deletes the first tweet from his Elastic Search and sees the result of this delete operation.

Description
Bora sets the url of the request: http://localhost:8080/twitter/tweet/1. Then he calls delete() on the Request.Builder. He executes a new call and gets the response as a JSON.

Figure 3.6.1.4: Sequence diagram of use case scenario - 4

#5

Request with headers
Primary Actor: Bora
Scenario
Bora makes a request with multiple headers.
Description

Bora set an Interceptor on the OkHttpClient. Then, he creates a chain request and he adds headers on this chain.

```
client.setInterceptor(new Interceptor() {
        public Response intercept(Interceptor.Chain chain) {
                Request chainRequest = chain.request();
                // add multiple headers to request here
                return chain.proceed(chainRequest);
        }
});
```

Figure: 3.6.1.5: Sequence diagram of use case scenario - 5

#6

Asynchronous GET to update view

Primary Actor: Bora

Scenario

Bora uses Asynchronous GET to update a TextView at his Android Project.

Description

First, with the same way, Bora creates a Request object but this time, he does not call execute() directly. He calls it like this:

```
client.newCall(request).enqueue(new Callback() {
        public void onResponse(Call call, final Response response) {
                // update TextView here with the response
        }
});
```

Hence, he updates the TextView when the response is returned at the callback function.

Figure: 3.6.1.6: Sequence diagram of use case scenario - 6

#7

Request with a tag

Primary Actor: Bora

Scenario

Bora builds a request with tag “dummy tag”. Then, he cancel requests with the tag: “dummy tag”.

Description

With the same way before, Bora builds a request with the addition of the method: tag(“dummy tag”). He queues the call. Then, at some point, he searches for requests with tag equals to “dummy tag” at the client.dispatcher().queuedCalls() and client.dispatcher().runningCalls() with the enhanced for loop. When it is found, he calls the method: call.cancel() to cancel.

Figure 3.6.1.7: Sequence diagram of use case scenario - 7

#8
Cache
Primary Actor: Bora
Scenario
Bora adds cache control header to the requests, in order to check requests’ cache control efficiency.
Description
Bora builds the request with the addition of the method:

.cacheControl(new CacheControl.Builder().maxAge(1, TimeUnit.DAYS).build()). In this way, he replaces the headers that are already existing with “maxAge(1, TimeUnit.DAYS)”. As a result, Bora specifies the directives for caching mechanisms.

Figure: 3.6.1.8: Sequence diagram of use case scenario - 8

4- Technical Problem Analysis

P1) How to authenticate in client-server side

Okhttp will communicate and authenticates with web services through OAuth protocol. Therefore, system should meet the requirements of OAuth protocol.

P2) How to protect user's’ credentials

Okhttp will be a library for Java and Android applications. Okhttp enables users to make response – request calls on the internet (web sites). Therefore, user credentials must be protected from other people in order to prevent hacking.

P3) How to provide short response time

Users can make several requests which is crucial to their system at the same time. Responses of these requests should be short enough to maintaining high performance.

P4) How to maintain sustainability of the system when changes occur over the time
Internet and systems that are used on internet are open to change in a short time. Therefore, Okhttp should be adaptable to changes.

P5) How to maintain reliability of the system
When the changes occur on the internet services, performance of the system should meet the stakeholders' needs.

P6) How to use OkHttp
OkHttp should be embedded into Android and Java SE / EE applications.

5- Domain Analysis

Since OkHttp library is a solution for a technical issue, domain analysis is mostly focused on technical domains. Next chapters will give more detailed information about domain analysis.

5.1- Domain Identification

- Client-Server Authentication
- Security
- Performance
- Concurrency (Multi-threading)
- Android Application Development
- Java SE and EE Application Development

To solve technical problems, we need solution domain elements which are associated with technical problems. In table below, each problem and its solution domain is described with priority of the solution domain. Highest priority is 1 and as number increases priority level decreases.

Problem	**Solution Domain**	**Priority**
How to authenticate in client-server side	Client-Server Authentication	2
How to protect TCP package content	Security	1
How to provide short response time	Performance	1
How to manage multi-threading	Concurrency Performance	1
How to integrate Android applications	Android Application Development	1
How to integrate Java SE and EE application	Java SE and EE Application Development	1

Table 5.1.1: Domain Problems Table

5.1.1- Client-Server Authentication

Client-server authentication is important for most of contemporary applications. There are many methods to provider client-server authentication such as Oauth. OkHttp should provide implementation or modules in order to establish client-server authentication. Therefore, knowledge of these protocols should be obtained.

These knowledge sources are required in order to solve problems in this domain.

ID	**Knowledge Source**	**Form**
KS1	Advanced Client/Server Authentication in TLS [A. Hess, J. Jacobson, H. Mills, R. Wamsley, E. Seamons, B. Smith]	Article
KS2	OAuth: The Big Picture [Greg Brail]	E-Book

KS3	The OAuth 2.0 Authorization Framework: Bearer Token Usage [M. Jones & D.Hardt, IETF]	Article
KS4	OAuth Web Authorization Protocol [Barby Leiba, IEEE]	Article
KS5	An ID-based client authentication with key agreement protocol for mobile client–server environment on ECC with provable security [H. Debiao, C. Jianhua, Hu Jin]	Article

Table 5.1.1.1: Knowledge source table for Client-Server Authentication domain

5.1.2- Security

Security is very important in order to protect users' information from people having bad intent. Security protocols, which have been developed in order to provide security for http requests, such as SSL and TLS should be implemented by OkHttp. Knowledge sources related to this domain is presented below.

ID	**Knowledge Source**	**Form**
KS6	SSL and TLS: Designing and Building Secure Systems [Eric Rescoria]	Book
KS7	The Transport Layer Security (TLS) Protocol Version 1.2 [T. Dierks, IETF]	Article
KS8	The Secure Sockets Layer (SSL) Protocol Version 3.0 [A. Freier & P. Karlton, IETF]	Article
KS9	Inside SSL: the secure sockets layer protocol [W. Chou, IEEE]	Article
KS10	The Transport Layer Security (TLS) Protocol Version 1.1 [T. Dierks, IETF]	Article

Table 5.1.2.1: Knowledge source table for Security domain

5.1.3- Concurrency (Multi-Threading)

OkHttp should handle multiple http requests concurrently. Therefore, OkHttp should implement multi-threading concept. Since OkHttp is implemented with Java, multi-threading APIs provided by Java SDK. Knowledge resources related to this domain is presented below.

ID	**Knowledge Source**	**Form**

KS11	Multithreaded Programming with Java [B. Lewis & D.J. Berg]	Book
KS12	Java Concurrency in Practice	E-Book
KS13	Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs	Book
KS14	Multithreading in Java: Performance and Scalability on Multicore Systems	Article
KS15	Java SE SDK	Existing Service

Table 5.1.3.1: Knowledge source table for Concurrency domain

5.1.4- Performance

OkHttp should complete http requests and dispatch response of requests quickly. Since OkHttp library is implemented with Java. Java programming language should be used by being careful about performance. Knowledge sources related to this domain is presented below.

ID	**Knowledge Source**	**Form**
KS16	Java Performance, 1st Edition	Book
KS17	Java Performance Tuning (2nd Edition)	Book

Table 5.1.4.1: Knowledge source table for Performance domain

5.1.5- Android Application Development

OkHttp library should be embedded in android applications. Therefore it should be compatible with Android SDK. Even though Android SDK is developed with Java programming language, there are differences between Android SDK and Java SE. Therefore, OkHttp should be optimized for Android SDK. Knowledge sources related to this domain is presented below.

ID	**Knowledge Source**	**Form**
KS18	Android Application Development	Book
KS19	Head First Android Development	E-Book
KS20	Android SDK	Existing Service

Table 5.1.5.1: Knowledge source table for Android Application Development domain

5.1.6- JAVA SE and EE Application Development

OkHttp library should be embedded in Java SE and EE applications. Therefore it should be compatible with Java SE and EE. It should be optimized for Java SE and EE. Knowledge sources related to this domain is presented below.

ID	**Knowledge Source**	**Form**
KS20	Java EE Development with Eclipse - Second Edition	Book
KS21	Java EE SDK	Existing Service
KS22	Java SE SDK	Existing Service

Table 5.1.6.1: Knowledge source table for Java SE / EE Application Development domain

5.2 Derived Concepts

Solution Domain	**Solution Domain Concept**
Client-Server Authentication	Authentication Manager
Security	SSL / TLS Certificate Manager
Concurrency	Multi-thread Manager
Performance	Http Request Duration Controller
Android Application Development	Android Application Manager
Java SE / EE Application Development	Java SE / EE Application Manager

Table 5.2.1: Derived concepts table

5.3 Domain Model

Figure 5.3.1. Domain Model

5.3.1 Architectural Views and Styles

5.3.1.1 Module View

i. Decomposition Style

All classes in OkHttp library is placed into a single package and these all classes is part of same system. Therefore, it can be inferred that OkHttp has a monolithic structure. Therefore, it cannot be decomposed into sub-modules.

ii. Generalization Style

OkHttp provides interfaces (protocols) in order to customtimize some functionalities and make library more compatible with applications. It can be seen that generalization style is used here and generalization diagram of system has been given below.

Figure 5.3.1.1.1. Generalization Style

As it is seen from the generalization diagram that there are 9 interfaces that OkHttp provides. OkHttp has default implementation for some of these interfaces. Explanation of these interfaces is provided below.

Callback: This interface is used in order to dispatch response of a request. Developers are able to receive response of their responses by implementing this interface.

Call: This interface is used in order to maintain a request process. OkHttp provides a default implementation for this interface. Developers also can implement their own Call interface.

Interceptor: This interface is used in order to observe and modify an ongoing request such as changing request headers. OkHttp provides a default implementation for this interface. Developers also can implement their own Interceptor interface.

Interceptor.Chain: This is an inner interface of Interceptor interface. It represent single step that Interceptor interface performs. Developers can modify and observer an ongoing request by implementing this interface.

WebSocket: This interface is used in order manage lifecycle a web socket. OkHttp provides a default implementation for this interface. Developers also can implement their own WebSocket interface. OkHttp developers recommend that WebSocket should be created by WebSocket.Factory (Details of this interface is provided below.) implementation.

WebSocket.Factory: This interface is used to create a factory for WebSocket. OkHttp provides a default implementation for this interface. Developers also can implement their own WebSocket.Factory interface.

CookieJar: This interface is used in order to provide policy and persistence for HTTP cookies. OkHttp provides a default implementation for this interface. Developers also can implement their own CookieJar interface.

Connection: This interface is used in order to manage sockets and streams of an HTTP connection. OkHttp provides a default implementation for this interface. Developers also can implement their own Connection interface.

Authenticator: This interface is used in order to create an authorization layer for requests. Developers can create their custom authentication layer by implementing this interface.

iii. Uses Style

Since OkHttp has monolithic structure, only classes are interacting with each other. Therefore, uses relationship between classes has been discussed in this report. Uses diagram of system is provided below. This diagram contains major uses relationships in OkHttp library.

Figure 5.3.1.1.2 Uses Style

OkHttpClient has major uses relationship with 4 other classes or interfaces. This uses relationships are mostly related with an HTTP request lifecycle.

First of all, **OkHttpClient** uses **Request** class instance in order to start an HTTP request. Request class represents an HTTP request. While an HTTP request is ongoing OkHttpClient uses instance of class which implements Call interface. As it is mentioned before, Call interface is used in order to maintain request process.

OkHttpClient uses Response class instance in order to represent response of HTTP request. It uses instance of Callback interface in order to dispatch this response to callee.

5.3.1.2 Component and Connector View

i. Client-Server

Figure 5.3.1.2.1 Client Server

In OkHttp, the communication between client and server is easy. Users can use OkHttpClient to make calls. They will send their requests and get the responses. Calls can either be synchronous or asynchronous. If they choose, they can also cache responses. For OkHttp, client-server style has many benefits. First and foremost, it allows reuse of the common services delivered by server. Also, it is convenient for performance, security and scalability.

ii. Pipe and Filter

Figure 5.3.1.2.2 Pipe and Filter

In OkHttp, Pipe and Filter view above shows that how events occur serially when users make requests. With regard to the request type, some of these stages may be skipped. For example, GET requests do not often need ReqestBody. In addition, some of them might have a types. For example, RequestBody can be JSON or XML data type and Calls can be synchronous or asynchronous.

5.3.1.3 Allocation View

i. Deployment View

Figure 5.3.1.3.1 Deployment View of OkHttp

In this view allocation, a relation of OkHttp software has shown with applications. Since internal structure of OkHttp beyond the scope of us and deployment diagrams are mostly used for mapping of C&C in the software architecture to the hardware of the platform, in this diagram application that uses OkHttp is demonstrated. Main module in our system is OkHttp Core in other word API. OkHttp can be used for two different platforms; Android and Java. All those artefacts (apk,jar and ear) will communicate with server to run the functionalities of the application through OkHttp

Core via Request – Response. Through TCP/IP socket(HTTP request), connection between server and application is established.

ii. Install View

Figure 5.3.1.3.2 Install View of OkHttp

The install style is used to show requirements for software to operate on a specific platform in terms of dependencies. Resulting files have to be packaged to be installed on the target production platform. In our case, platforms are Android SDK and Java JDK. Files need to be organized so as to retain control and maintain the integrity of the system build and package process [1].
OkHttp is an HTTP client that runs on Android and Java. Java SDK and Android SDK is required for operation of OkHttp. Cookie jar is needed for providing policy and persistence for HTTP cookies. TCP/IP socket is internal endpoint for sending or receiving data at a single node in a computer network. In OkHttp all request uses same host to share a socket. To do so, HTTP/2 support is required. More advanced download management, Transparent GZIP must also be provided. Interceptors are a powerful mechanism that can monitor, rewrite, and retry calls. Interceptor APIs are available through the library APIs. For cache responses, library needs a cache directory that you can read and write to, and a limit on the cache size.

iii. Work Assignment View

Figure 5.3.1.3.3 Work Assignment View of OkHttp

There are two separate teams that are developers of OkHttp and software developers use OkHttp. Software developers' team consists of three parts. Since OkHttp supports two platforms, Android Developers and Java developers use library in their application. Testers of the applications are responsible for testing the OkHttp related functions on the system.The software engineers implement the system. They are responsible to implement the system according to needs of the stakeholders. The Software architects are responsible for designing the software architecture of the system. Testers of OkHttp are responsible for testing all the functionalities of the OkHttp library.

6. Design Pattern

In this section, two patterns will be examined in detailed for better understanding Okkhttp. These two patterns are observer pattern, and client-server.

6.1. Observer Pattern

Observer pattern is used for realization of changes in observer's subject. It allows system to be aware of any change and let observer's object to make appropriate changes. In the OkHttp, observer pattern can be clearly indicated. In order to check changes in the observerJSON (JSON File), observer interface of OkHttp can make http request via makeRequest. After each request there will be a response from the server (getResponse) whether or not there were a change. If request returns any sort of change on the server (subjectJSON), subject interface will be notified. As a response, the subject interface will notify back observer interface.

Figure 6.1.1. Observer Pattern

6.2. Façade Pattern

Façade pattern is applicable for complex software libraries. Façade pattern hides the complexity of the system and provides an interface to the user. By using this interface, user can access and use the complex system. Http request respond system is difficult and unimportant for casual software developers who just wants to complete their tasks. For this purpose, façade patterns provides easy interfaces for the software developers.

In this pattern, OkHttp has some fundamental methods such as AsynchronousRequest, AsynchronousResponse, HeaderBuilder, synchronousRequest, synchronousResponse, Get, Post. These methods are provided through interface for the software engineers. By using these methods users can use the http request-response services. In other words, façade pattern provides easy usage for OkHttp's abstract methodologies.

Figure 6.1.1. Façade Pattern

7. Evaluation of Software Architecture

Evaluation of the software architecture is necessary for understanding the outcomes of design decisions for the system. There are various methods for Evaluation of Software Architecture, such as SAAM (Software Architecture Analysis Method) and ATAM (Architectural Tradeoff

Analysis Method). Since the development of OkHttp is not done by us, and OkHttp is an already developed system, SAAM is not favorable for evaluation of OkHttp's architecture, And ATAM is the best fit to do so.

7.1. ATAM (Architectural Tradeoff Analysis Method)

The consequences of architectural decisions based on multiple attribute refinements for different quality attributes are investigated with the help of ATAM (Architectural Tradeoff Analysis Method). Thanks to the same method, the relationships between each quality attributes are also understood. In the below table, the reader can find quality attributes, related refienements and the rationale for that refinement. The rationales are derived from the architecture of OkHttp and scenarios. The contributors of ATAM are the decision makers of the project and the stakeholders of the architecture.

Quality Attribute	**Attribute Refinement**	**Rationale**
Performance	Response Time	Gives Response Less than 200 ms.
	Latency	Connection pooling reduces latency.
	Efficiency	Response caching minimizes the repeat of requests.
	Resizability	Uses Okio for input/output and buffers.
Usability	Ease of Use	Supports both synchronous and asynchronous blocking calls with callbacks.
Security	Privacy	SSL and TLS are implemented.
Reliability	Network Resilience	If connection handshake fails, new connections with modern features are initiated.

Table 7.1.1 ATAM for OkHttp

8. Conclusion

In conclusion, in this document, the problems encountered during Java and Android development using http library is identified. Then, OkHttp's approach to solve this problem is stated. After that, a requirement analysis is done, along with stakeholder identification, functional / non-functional requirement identification and use case scenarios, use case models, and dynamic model. Furthermore, technical problem analysis is done, and after that, domain analysis is done. After all of these, design patterns of OkHttp is identified. Finally, the architecture of OkHttp is evaluated with ATAM (Architectural Tradeoff Analysis Model).

References

[1] N. Esquenazi, "codepath/android_guides", GitHub, 2017. [Online]. Available: https://github.com/codepath/android_guides/wiki/Using-OkHttp. [Accessed: 26- Nov- 2017].

[2] "OkHttp", Square.github.io, 2017. [Online]. Available: http://square.github.io/okhttp/. [Accessed: 26- Nov- 2017].

[3] M. Abdelgawad, "Software architecture with SOA modeling Flavor", Slideshare.net, 2017. [Online]. Available: https://www.slideshare.net/MohamedZakarya2/software-architecture-with-soa-modeling-flavor-62550658. [Accessed: 19- Dec- 2017].