**Task 1**

**What is an API**

It stands for application programming interface and can be described as software “function”. It is the way that software communicates with other software, an interface intended for machine to machine communication.

API can be used for both web services and offline service (e.g. windows search bar), the APIs used in HTTP is called web API and there exists different methods (e.g. RPC, RMI, REST).

**REST & RESTful web service:**

the name “Representational state transfer” is intended to evoke an image of how a well-designed web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use. REST architecture consists of constraints, **architecture style** for designing loosely coupled applications over HTTP. REST does not enforce any rule regarding how it should be implemented at lower level, it just put high level design guidelines and leave you to think of your own implementation. REST defines **6 architectural constraints** which make any web service a true RESTful API:

1. **Uniform interface**

As the constraint name itself applies, you MUST decide APIs interface for resources inside the system which are exposed to API consumers and follow religiously. A resource in the system should have only one logical URI and that should provide a way to fetch related or additional data.

Also, the resource representations across system should follow certain guidelines such as naming conventions, link formats or data format (xml or/and json).

1. **Client–server**

This essentially means that client application and server application MUST be able to evolve separately without any dependency on each other. A client should know only resource URIs and that’s all. Today, this is normal practice in web development so nothing fancy is required from your side. Keep it simple.

1. **Stateless**

Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

If client application needs to be a stateful application for the end user, where user logs in once and do other authorized operations thereafter, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.

1. **Cacheable**

Caching brings performance improvement for client side, and better scope for scalability for a server because the load has reduced.

In REST, caching shall be applied to resources when applicable and then these resources MUST declare themselves cacheable. Caching can be implemented on the server or client side.

1. **Layered system**

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Client does not need to know what is on server C either, all the client knows is what they are required to know.

1. **Code on demand (optional)**

Well, this constraint is optional. Most of the time you will be sending the static representations of resources in form of XML or JSON. But when you need to, you are free to return executable code to support a part of your application e.g. clients may call your API to get a UI widget rendering code.

GIT is developed by Linus Torvalds, the developer of Linux. GIT is a version control system used for tracking changed files among multiple programmers. Primarily used in the development of big projects because of its source code management and responsitories. GIT can track changing history of the entire project, including every file individually. In case some error occurred during developing, programmers could find it from history list directly rather than look through the whole code.

**Task 2**

**Rest API in project implementation:**

The project is called adventure club, in the case of making it a legitimate business this would require some degree of convenience for the user. Thus, implementing Google APIs is the easiest way to get a hold of fully functional and easily integrated APIs. In the case of adventure club, the first that come to mind are Google maps and hangout. Making it easier for user to find the adventure location and communicate with the rest of the team. Of course it depends on the size of the venture and the business module, there may be many more functionalities requires and thus more APIs that can work in a cohesive way. However, the primary needs that come to mind for such a webservice are information communication and communication between its members.

George Iskef

References:

Applied Web architecture (lectures)

Client-server communication (lectures)

GoogleAPIs