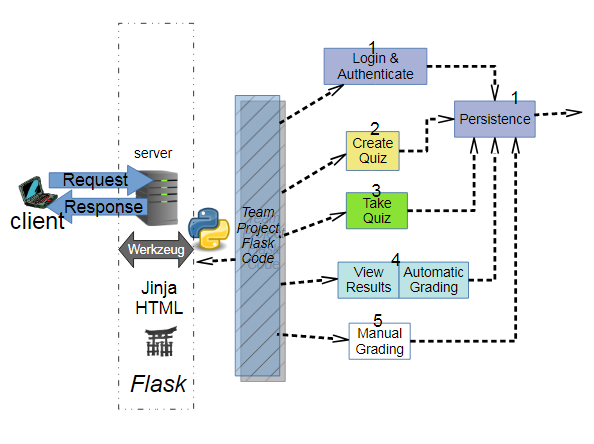
**A description (up to two pages plus diagrams) of the project, describing main functionality and software architecture, depicting each module and its responsibilities.**

The Project deals with the construction of a quiz application. The responsibilities for this project have been distributed in several abstraction layers.

* A front end that interacts with the principal user. It provides the user with an interface through which it can use its functionality of logging into the system, creating a quiz and taking a quiz. For this I have primarily used JSON which is a javascript object notation used for communication between the browser and server. Along html/bootstrap template to provide a user friendly interface for the principal users. The Flask app.py is used to pass information from the lower layer of abstraction to the server and vice versa.
* The Login/Authenticate, creating quiz and take quiz modules are handled by the User.py, Quiz.py and quesbank.py modules. They provide a second layer of abstraction where after receiving data from the lower level of abstraction for example the persistence layer, it processes that data and sends it off to the flask app.py which is used to communicate with the server.
* The persistence layer provides an interface which can be used by the second layer of abstraction (create quiz etc.) to store and retrieve data from the database.

**App.py:**

This module provides a flask front end for the entire project. It provides routes to handle login, registration, the main dashboard, logout, getting a question, deleting a question, adding a quiz, getting a quiz, deleting a quiz, editing a quiz, saving a quiz attempt and retrieving a quiz attempt. Data received from the browser is in JSON format. It is then converted into python dictionaries by using the request.json method. The data received from this method is passed along to the quiz/user classes for processing and is stored in the same format in the data base using persistence.

**Programmer 1: Login, authenticate, persistence**

* password authentication
* two classes of privilege: student and instructor
* new account configuration
* storage and retrieval for other module

For the login and authentication, I have created the user class in the user.py file which deals with the registration and login of the primary actors. Each of the primary actors have a varied level of privilege and hence while registering their accounts, it is made sure if they are an instructor or a student. While they create their passwords, their passwords are linked with a hash function and then stored. Then while logging again a check is done to make sure which account type is associated with the particular user. Once the credentials are verified, only then access is granted into the system.

For persistence of the entire system, I have created the persistence class in the persist.py file. It deals with the storage and retrieval of all the components that require storage in a database in the quiz system. The other programmers instead of dealing with the storage components (like a database), deal with my persistence class which helps them store their data for example for creating a quiz and later retrieving the quiz for the purpose of either taking the quiz or modifying it(adding or removing questions). The persistence modules assigns unique IDs to the data being stored to avoid any collision. These unique IDs are assigned internally and none of the other programmers need to be aware of their functionality.

**Programmer 2: Create Quiz**

* instructor only
* manage question bank
  + MCQ's questions, choices and answers
  + modify,copy existing questions
  + MCQ can have multiple correct answers
* manage quiz content
  + instructor access limited to owner of quiz
  + add/remove quiz questions
  + start and end time
  + students taking the quiz
  + modify and copy quiz

My program makes sure that the create quiz functionality is only available to the instructor. During the login and authentication in the user.py module. The account type is determined and the user is logged into the application. The student and instructor account types are provided two different dashboards and user interfaces. Only the instructor dashboard is provided with the functionality to create a quiz as seen in app.py. In the **quiz.py** module the setauthors(), removeauthors() and check\_quiz\_access() provide the functionality of setting authors and removing authors. The instructors are authors by default.

The question bank is used to create questions so while creating a quiz, the questions can be just added on to a particular quiz. The quesbank class deals with the functionality of the question bank. It provides methods for adding a question to the bank, getting all the questions from the question bank, updating a question and verifying the question answer.

The quiz class assures that the creating a quiz functionality is only available to the instructor as mentioned above. It includes methods which create a quiz, can update a quiz, delete a quiz and set start and end time.

**Programmer 3: Take Quiz**

* check access
  + student and time
* navigate questions

Only registered students can log into the application. Students are taken to the student dashboard where they can chose a quiz if it has been created and attempt it.

They can chose the their answers by clicking on the checkbox and hence make a selection. Once they have made the selections they can submit their attempt. Their attempt is stored with the storeQuizForUser method in the user class and can be retrieved with he retrieve\_quiz\_for\_user method.