# Solution M4: NoSQL

One possible solution to the homework tasks could look like this:

* First, we will prepare the web application files
* The **index.html** file has a few cosmetic changes. The new version is:

<html ng-app="mongo">

  <head>

    <title>Guestbook</title>

    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>

    <script src="controllers.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.13.0/ui-bootstrap-tpls.js"></script>

  </head>

  <body ng-controller="MongoCtrl">

    <div style="width: 50%; margin-left: 20px">

      <h2>Guestbook</h2>

    <form>

    <fieldset>

    <input ng-model="msg" placeholder="Messages" class="form-control" type="text" name="input"><br>

    <button type="button" class="btn btn-primary" ng-click="controller.onMongo()">Submit</button>

    </fieldset>

    </form>

    <div>

      <h2>Messages</h2>

      <div ng-repeat="msg in messages track by $index">

        {{msg}}

      </div>

    </div>

    </div>

  </body>

</html>

* In similar fashion, the **controllers.js** file has a few changes:

var mongoApp = angular.module('mongo', ['ui.bootstrap']);

/\*\*

 \* Constructor

 \*/

function MongoController() {}

MongoController.prototype.onMongo = function() {

    this.scope\_.messages.push(this.scope\_.msg);

    var value = this.scope\_.msg;

    this.http\_.get("guestbook.php?cmd=set&key=messages&value=" + value)

            .success(angular.bind(this, function(data) {

                this.scope\_.mongoResponse = "Inserted.";

            }));

};

mongoApp.controller('MongoCtrl', function ($scope, $http, $location) {

        $scope.controller = new MongoController();

        $scope.controller.scope\_ = $scope;

        $scope.controller.location\_ = $location;

        $scope.controller.http\_ = $http;

        $scope.controller.http\_.get("guestbook.php?cmd=get&key=messages")

            .success(function(data) {

                console.log(data);

                $scope.messages = data.data.split(",");

            });

});

* The modified version of the **guestbook.php** file is the following:

<?php

error\_reporting(E\_ALL);

ini\_set('display\_errors', 1);

if (isset($\_GET['cmd']) === true) {

  $host = "mongo-host";

  $connection = new MongoClient( "mongodb://".$host.":27017" );

  $db = $connection->selectDB('test');

  $collection = $db->selectCollection('messages');

  header('Content-Type: application/json');

  if ($\_GET['cmd'] == 'set') {

    $collection->insert(array("message" => $\_GET['value'], "published" => date("Y-m-d H:i:s")));

    print('{"message": "Inserted."}');

  }

  else {

    $results = $collection->find();

    $value = "";

    foreach($results as $result) {

       $value = $value.$result['message'].",";

    }

    $value = substr($value, 0, -1);

    print('{"data": "' . $value . '"}');

  }

}

else {

  phpinfo();

} ?>

* All three files are stored in a folder called **web**
* A copy of this folder is expected to be found on the Kubernetes node (minikube) in the following path **/home/docker/web**
* Next, we will prepare a file named **seed.js** to create the collection in our MongoDB database:

db.createCollection('messages');

* The file is stored in a folder called **db**
* A copy of this folder is expected to be found on the Kubernetes node (minikube) in the following path **/home/docker/db**
* As a next step, we can prepare a set of YAML files to deploy our application on Kubernetes
* First, we will create a **web-pod.yml** file for the front-end with the following content:

apiVersion: v1

kind: Pod

metadata:

  name: hw-web-pod

  labels:

    app: hw-web

spec:

  containers:

  - name: hw-web-container

    image: shekeriev/php-apache-mongo

    ports:

    - containerPort: 80

    volumeMounts:

    - mountPath: /var/www/html/

      name: web-data

  volumes:

  - name: web-data

    hostPath:

      path: /home/docker/web

      type: Directory

* Next, we will create a **web-svc.yml** file for the front-end service with the following content:

apiVersion: v1

kind: Service

metadata:

  name: hw-web-svc

  labels:

    app: hw-web

spec:

  type: NodePort

  ports:

  - port: 80

    nodePort: 30001

    protocol: TCP

  selector:

    app: hw-web

* Then, we will create a **db-pod.yml** file for the back-end with the following content:

apiVersion: v1

kind: Pod

metadata:

  name: hw-db-pod

  labels:

    app: hw-db

spec:

  containers:

  - name: hw-db-container

    image: mongo

    ports:

    - containerPort: 27017

    volumeMounts:

      - mountPath: /docker-entrypoint-initdb.d

        name: db-data

  volumes:

  - name: db-data

    hostPath:

      path: /home/docker/db

      type: Directory

* ***Please note that if your CPU is eld enough and doesn’t support Intel AVX instruction set, you won’t be able to use the latest version (5.x) of MongoDB and you should add the 4.4.8 tag (mongo:4.4.8)***
* And finally, we will create a **db-svc.yml** file for the back-end service with the following content:

apiVersion: v1

kind: Service

metadata:

  name: mongo-host

  labels:

    app: hw-db

spec:

  ports:

  - port: 27017

    protocol: TCP

  selector:

    app: hw-db

* Now, we can test our application
* minikube start
* scp -r web docker@<minikube-ip>:/home/docker/web (password tcuser)
* scp -r db docker@<minikube-ip>:/home/docker/db
* Execute the following command to spin up the app

**kubectl create -f . -R**

* Check the progress with

**kubectl get pods,svc**

* Open a browser tab and navigate to http://<minikube-ip>:30001
* Once done, delete the application with

**kubectl delete -f . -R**