Klausur
„Entwicklung webbasierter Anwendungen“
Wintersemester 2025

Nachname	**Vorname**	**Matrikelnummer**	**Note**

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**
Punkte	20	15	15	20	10	20
Ihre Punkte						
						Gesamt

Hinweise

- **Bearbeitungszeit:** 90 Minuten
- **Erlaubte Hilfsmittel:** Ein beidseitig handbeschriebenes DIN-A4-Blatt

- **Lesen** Sie die gesamte Klausur vollständig, bevor Sie mit der Bearbeitung beginnen.
- Achten Sie auf **Vorder- und Rückseite**!
- **Schreiben** Sie klar und deutlich. Unleserliche Antworten werden nicht bewertet!
- Bitte achten Sie auf präzise und knappe Formulierungen.
- Verwenden Sie für Ihre **Lösungen** die vorgesehenen Antwortfelder. Sollte Ihnen der Platz nicht reichen, nutzen Sie das Zusatzpapier auf der letzten Seite.
- **Codequalität:** Schreiben Sie wartbaren und standardkonformen Code, der den Vorgaben der Veranstaltung entspricht.

- **Abgabehinweis:** Die Klammerung der Aufgabenblätter darf nicht entfernt werden. Es liegt in Ihrer Verantwortung sicherzustellen, dass alle Blätter vollständig abgegeben werden. Im Falle eines versehentlichen Lösens der Klammerung beschriften Sie alle Blätter mit ihrem Namen.

Die Aufgabe allgemein: „H_DA – Wetterdienst“
Gegenstand dieser Klausur ist die Umsetzung ausgewählter Teilfunktionen einer Wetterdienst-Webanwendung.

Für die technische Realisierung kommen PHP und MySQL im Backend sowie HTML, CSS und JavaScript im Frontend zum Einsatz. Es sollen keine Frameworks verwendet werden.

Im Anschluss folgen zusätzlich einige Wissensfragen zu grundlegenden Webtechnologien.

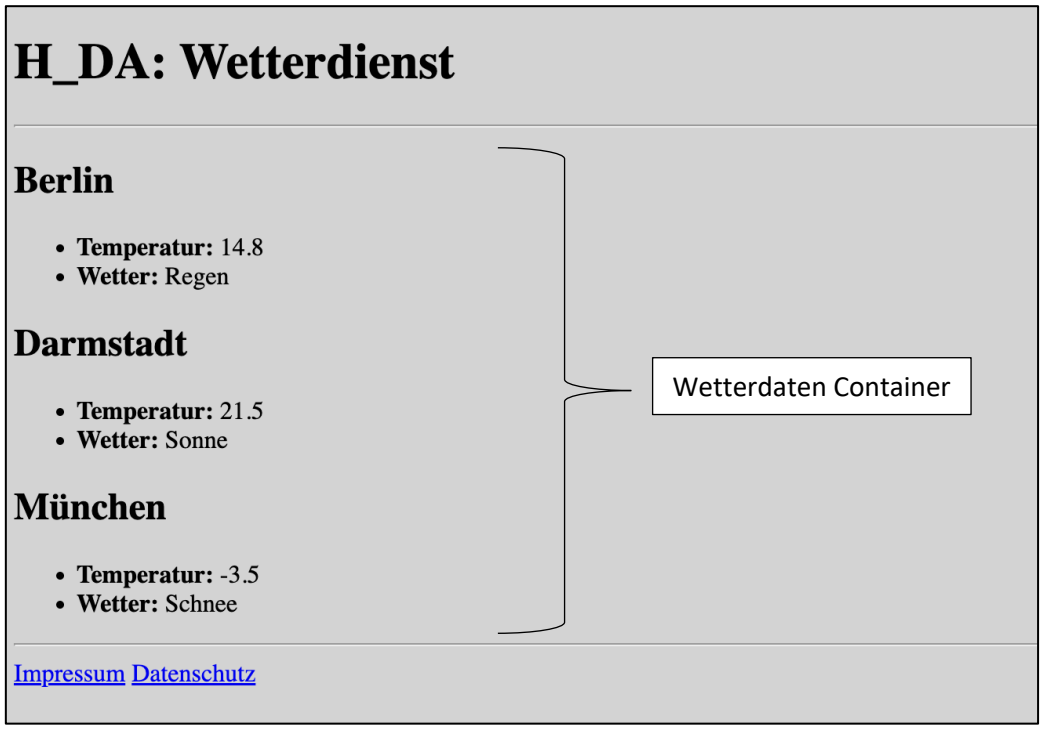

Abbildung 1: Layout

Datenbankstruktur
Die folgende Abbildung zeigt die Struktur unserer Datenbank. Sie bildet die Grundlage für alle nachfolgenden Aufgaben und Abfragen.

- Die Datenbank heißt hda_weather
- Die Tabelle heißt data.
- Die ID wird autoinkrementiert

#	Name	Typ
1	id	int(11)
2	city	varchar(100)
3	temperature	decimal(4,1)
4	weather	varchar(50)

id	city	temperature	weather
1	Berlin	14.8	Regen
2	Darmstadt	21.5	Sonne
3	München	-3.5	Schnee

Abbildung 2: Datenbankstruktur mit Beispieldaten

Aufgabe 1: HTML & PHP (20 Punkte)

Setzen Sie das **semantische HTML** gemäß der Abbildung 1 um. Folgendes soll beachtet werden:

- Der Titel der Seite lautet: H_DA – Wetter. Die JS und CSS-Dateien heißen: exam.js und exam.css
- Der <head> soll so ergänzt werden, dass die Seite auf mobilen Geräten korrekt dargestellt wird.
- Der Wetterdaten-Container wird auf Basis der Daten aus der Tabelle data der Datenbank hda_weather mithilfe von **HTML und Embedded PHP (PHP/HTML-Mix)** generiert.
- Nutzen Sie außerdem bitte die **alternative PHP-Syntax** für mögliche Kontrollstrukturen.
- Gemäß der MVC-Architektur können Sie davon ausgehen, dass der Controller dem View die Daten aus der Datenbank bereits in der Variable **$data** als assoziatives Array zur Verfügung stellt.
- Wenn **$data** leer ist soll der Text „Keine Daten vorhanden“ statt der Wetterdaten erscheinen.
- Der Container für die Wetterdaten soll per JavaScript angesteuert werden können.
- Markieren Sie Abschnitte, welche ihrer Meinung nach in Partials ausgelagert werden sollten.

exam.view..php

exam.view..php

Aufgabe 2: PHP (15 Punkte)

Implementieren Sie die komplette Model-Klasse WeatherModel, welche vom BaseModel (Siehe Anhang) erbt. Das WeatherModel hat nur die Methode getAllData(). Die Methode soll alle Wetterdaten aus der Tabelle data aus der Datenbank hda_weather laden und diese als assoziatives Array an den Controller zurückgeben. Verwenden Sie für den Datenbankzugriff das vererbte Datenbankobjekt $this->db (MySQLi). Denken Sie an Error Handling, und richtige Rückgabewerte. Aktivieren Sie außerdem die strikte Typisierung und binden Sie alle notwendigen Dateien ein.

WeatherModel.php

Aufgabe 3: PHP und JavaScript (15 Punkte)

Das komplette DOM vom Wetterdaten-Container aus Aufgabe 1 wird nun alle drei Sekunden per clientseitigem Polling aktualisiert. Das Backend liefert hierfür alle Daten als JSON-String, den das Frontend per Fetch API anfragt und anschließend das DOM aktualisiert. Die Daten werden über das WeatherModel aus Aufgabe 2 abgefragt. Nutzen Sie für die JSON-Generierung die vererbte Funktionalität vom BaseController. (Siehe Anhang) Die Methode handleRequest() ist der Startpunkt.
Beispielhafte Struktur eines API-Response-Eintrags:
{"id":<int>,"city":"<string>","temperature":<float>,"weather":"<string>"}

Vervollständigen Sie den ApiController und JavaScript-Code.

ApiController.php

```php
<?php
declare(strict_types=1);

class ApiController extends BaseController {
    /**
     * Retrieve all data necessary for the response.
     *
     * @return array Data array
     */
    private function getData(): array {

    }

    /**
     * Generate the full response output.
     * This method may output HTML, JSON, or other formats as needed.
     *
     * @return void
     */
    public function generateResponse(): void {

    }

    /**
     * Start! controller->handleRequest() wird im Endpunkt api.php aufgerufen
     * Handles the full request lifecycle
     *
     * @return void
     */
    public function handleRequest(): void {

    }
}
```

exam.js

```
function requestData() {
    try {
        const response =  ('api.php');

        if (!response.ok) throw new Error('Netzwerkantwort war nicht ok');

        const data =

        process(data);

    }  {
        console.error('Übertragung fehlgeschlagen:', error);
    }
}

function process(data) {
    // Der Wetterdaten-Container aus Aufgabe 1
  output = document.getElementById('output');

    output.innerHTML = '';

    for (const item of data) {
        const heading =

        heading.innerText = item.city;
        output.

        const list = document.createElement('ul');

        const temperatureElement = document.createElement('li');

        = 'Temperatur: ' + item.temperature;
        list.appendChild(temperatureElement);

        const weatherElement =

        weatherElement.innerText =


        output.appendChild(list);
    }

}

document.addEventListener(  , () => {

});
```

Aufgabe 4: CSS (20 Punkte)

Setzen Sie die folgenden Layouts aus Abbildung 3 und Abbildung 5 mit **HTML, CSS und Flexbox** um. Das vollständige HTML-Grundgerüst ist nicht erforderlich; außerdem müssen nicht alle Boxen komplett ausgeschrieben werden!

A)

- Die Boxen sind 200px breit und 150px hoch.
- Die Schriftgröße in den Boxen beträgt das 1,5-Fache der Schriftgröße des Eltern-Elements.
- Die Container Hintergrundfarbe ist gray und die der Boxen lightgray.
- Der Text in den Boxen ist mit Flexbox horizontal und vertikal zentriert.
- Der Abstand zwischen dem Rand vom Container und den Boxen beträgt 10px.
- Zwischen der Bildschirmbreite von 601px bis 1024px soll der Container Hintergrund nicht gray, sondern rot sein. Implementieren Sie hier die Farben Rot als HEX-Wert. (Syntax reicht)
- Wie können Sie ausschließlich mit Flexbox und ohne Media Queries das Verhalten realisieren welches man im Desktop/Handy Vergleich sieht. (Abbildung 3 und 4)

B)

- Das Bild aus Abbildung 5 soll mithilfe von HTML und CSS responsiv dargestellt werden. Abhängig von der Bildschirmbreite soll jeweils eine Bilddatei mit geringerer Dateigröße geladen werden, wobei das Bild seine Proportionen beibehält. Tipp: Welches HTML-Tag ist hierfür geeignet?

Abbildung 3: Aufgabe A - CSS-Layout - Desktop

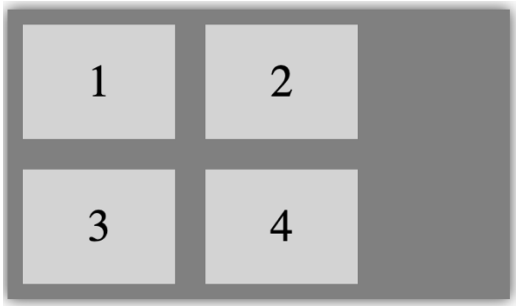

Abbildung 4: Aufgabe A - CSS-Layout - Handy

Abbildung 5: Aufgabe B - Responsives Bild

A) HTML & CSS

B) HTML & CSS

Aufgabe 5: Fehlersuche und Codeverständnis (10 Punkte)

A) Im folgenden JavaScript-Code werden eine normale Funktion und eine Arrow Funktion innerhalb eines Objekts verwendet. Welche Ausgaben erscheinen in der Konsole und warum? **(5 Punkte)**

```
this.name = "Max";

const user = {
  name: "Lina",
  sayHiArrow: () => {
    console.log("Hi " + this.name);
  },
  sayHi: function () {
    console.log("Hi " + this.name);
  }
};

user.sayHi();      // Konsolenausgabe, HIER HINSCHREIBEN →
user.sayHiArrow(); // Konsolenausgabe, HIER HINSCHREIBEN →
```

Warum?

B) Finden Sie alle syntaktischen sowie logischen Fehler im folgenden JavaScript-Code und korrigieren Sie diese. **(5 Punkte)**

```
const input = document.getElementMyId("username");

const submit = document.querySelector("#submitBtn");

submit.addEventListener("click", sayHello());

function sayHello() {

  const name = input.value;

  if (name.length = 0) {
    name = "Kein Namen";

    console.log("Bitte geben Sie einen Namen ein");

  } else {

    console.log("Hallo " + Name);

  }

}
```

Aufgabe 6: Wissensfragen (20 Punkte)

Frage 1: Nennen Sie drei HTTP-Methoden und erläutern Sie, wofür sie im REST-Kontext genutzt werden. **(6 Punkte)**

Frage 2: Erklären Sie SQL-Injection und Cross-Site Scripting (XSS) und nennen Sie jeweils eine Maßnahme die uns PHP zur Vermeidung bietet. **(6 Punkte)**

Frage 3: Welche Aussagen zur MVC-Architektur sind richtig? Markieren Sie die richtigen Aussagen.
Punkte gibt es nur bei vollständiger und fehlerfreier Auswahl. (4 Punkte)

A) Das Model ist für die Daten und die Geschäftslogik zuständig.

B) Die View ist für die Darstellung der Daten verantwortlich.

C) Der Controller verarbeitet Benutzereingaben und koordiniert Model und View.

D) Die View greift direkt auf die Datenbank zu.

Frage 4: Wofür stehen die Buchstaben CRUD und welche grundlegenden Operationen beschreiben sie in der Webentwicklung? Markieren Sie die richtigen Aussagen.
Punkte gibt es nur bei vollständiger und fehlerfreier Auswahl. (4 Punkte)

A) C steht für Create und bezeichnet das Erstellen neuer Datensätze.

B) R steht für Remove und beschreibt das Löschen von Daten.

C) U steht für Update und bedeutet das Verändern bestehender Daten.

D) D steht für Deploy und beschreibt das Bereitstellen der Daten.

Anhang 1: Alle Controller erben von **BaseController** und die Models von **BaseModel**

BaseController.php

```php
<?php
declare(strict_types=1);

abstract class BaseController {
    protected function renderHtml(string $viewFile, array $viewData = []): void {
        header('Content-Type: text/html; charset=UTF-8');
        extract($viewData);
        require $viewFile;
    }

    protected function renderJson(mixed $data): void {
        header('Content-Type: application/json');
        echo json_encode($data, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
    }
}
```

BaseModel.php

```php
<?php
declare(strict_types=1);

abstract class BaseModel {
    protected readonly MySQLi $db;

    public function __construct() {
        $host = 'localhost';
        $user = 'public';
        $password = 'public';
        $database = 'hda_weather';

        $this->db = new MySQLi($host, $user, $password, $database);

        if ($this->db->connect_error) {
            throw new Exception('Error: ' . $this->db->connect_error);
        }

        if (!$this->db->set_charset('utf8mb4')) {
            throw new Exception('Error setting charset: ' . $this->db->error);
        }
    }

    public function __destruct() {
        if (isset($this->db)) {
            $this->db->close();
        }
    }
}
```

Zusatzpapier (Falls Sie zu einer Aufgabe etwas ergänzend erwähnen wollen)