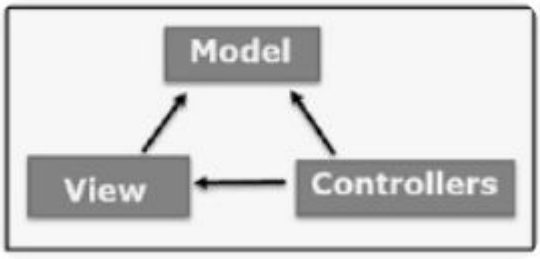
Изпитна тема № 13: Интернет програмиране

MVC модел. Бази данни и ORM технологии. Реализиране на CRUD операции. Създаване на шаблонни изгледи. Упълномощаване (authentication) и удостоверяване (authorization). Управление на сесии и бисквитки. Често срещани уязвимости в сигурността на уеб приложенията (SQL инжек-ция, XSS атака, CRSF, LFI и RFI, DDoS атака, MITM атака). Принципи на REST API. Работа с REST API в JSON/XML формат. Използване на AJAX в REST API.

Дидактически материали: *Компютър с подходяща версия на интегрирана среда за разработка от изучаваните, уеб браузър и сървърен софтуер. Наличие на подходяща версия на СУБД от изучаваните. Задачи и фрагменти от код на програмен език от изучаваните.*

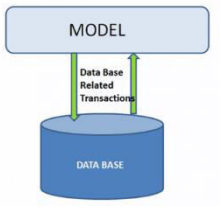
Критерии за оценяване на изпитна тема № 13	Максимален брой точки
1. Обяснява и представя графично MVC модела.	12
2.Описва същността и демонстрира употребата на ORM технологиите.	8
3. Реализира CRUD операции.	24
4. Дефинира и създава шаблонен изглед.	8
5. Различава автентикацията и авторизацията.	8
6. Диференцира сесиите от бисквитките.	8
7. Обяснява и дава пример за различните уязвимости в сигурността на уеб при-ложенията. Прави изводи за предотвратяването им.	16
8. Обяснява основните принципи на REST API.	4
9. Възпроизвежда и обяснява код за работа с REST API в JSON/XML формат.	6
10. Възпроизвежда и обяснява код за използване на AJAX в REST API.	6
Общ брой точки	100

1. Обяснява и представя графично MVC модела – 12т.



The Model-View-Controller (MVC) е архитектурен модел, който разделя приложение на три основни логически компонента: модел, изглед и контролер. Всеки от тези компоненти е създаден, за да обработва специфични аспекти на разработка на приложение.

Моделът (*Model*) е централен компонент в шаблона. Това е динамичната структура от данни на приложението, независима от потребителския интерфейс. Моделът управлява данните, логиката и правилата на приложението.



Изгледът (*View*) е изходящият поток от информация (*това, което приложението изпраща като отговор до дисплея, респективно – до потребителя, в следствие на неговата заявка*). Възможни са няколко различни изгледа на една и съща информация, като например различни диаграми за мениджмънт на даден ресурс или различни таблици.

Контролерът (*Controller*) -приема потребителския вход (т.е. данните, които потребителя въвежда, неговите заявки и т.н.) и ги преобразува в команди към модела или изгледа.

MVC стъпки:

Входяща заявка се пренасочва към контролер (За уеб: HTTP Заявка)

Контролерът обработва заявка и създава презентационен модел (избира и подходящ резултат (например: View)

Моделът се предава на изгледа. Изгледът преобразува Модела в подходящ изходен формат (HTML).Отговорът е предоставен (HTTP отговор)

2. Описва същността и демонстрира употребата на ORM технологиите-8т.

ORM (Object-Relatoinal Persistence Framework) се използва за автоматично създаване, схема на бази данни.

Object–relational mapping (ORM, O/RM, and O/R mapping tool) в компютърните науки е техника за програмиране за преобразуване на данни между системи от типове, използвайки обектно-ориентирани езици за програмиране. Това на практика създава "база виртуални обекти", която може да се използва в рамките на езика за програмиране.

Предимства и недостатъци на ORM

С **ORM** значително се опростява необходимия изходен код. Картографирането ще бъде автоматично. Това го прави лесен и бърз за използване, освен това той осигурява сигурност на слоя за достъп до данни срещу атаки.

Недостатъци-В силно натоварени среди може да се намали производителността, тъй като се добавя допълнителен слой към системата.

ORM за езици за програмиране

- **Java**: Hibernate, MyBatis, iBatis, Ebean и др.
- **. NET**: Entity Framework, nHibernate, MyBatis.Net и др.
- **PHP**: Doctrine, Propel, Rocks, Torpor и др.
- **Pyton:** Peewee, SQLAlchemy, PonyORM, Elixir и др.

3. Реализира CRUD операции – 24т.

CRUD	SQL
Create	INSERT
Read	SELECT
Update	UPDATE
Delete	DELETE

Преглед на CRUD операциите

База данни, в която софтуерът създава, чете, актуализира и изтрива данните чрез Заявки. Компютърният софтуер може да отговори на потребителските изисквания по-бързо и ефективно чрез правилно проектирана база данни и заявки. Това предполага, че тестването и проверката на базата данни е много важен фактор.

CRUD означава:

- Създаване (CREATE) –въвеждане, запис в базата данни
- четене(READ) избиране от БД и преглед;
- актуализиране(UPDATE) – актуализиране на данните, напълно или частично и
- изтриване(DELETE) – изтриване и унищожаване на записа.

Това са четирите основни и основни функции на постоянното съхранение. Те често се правят в софтуерни приложения чрез формуляри.

В зависимост от софтуерните изисквания, CRUD циклите могат да варират.

Например: понякога продавачът създава акаунт и потребителят го разглежда. Потребителят може да няма привилегията да го редактира или изтрие. От друга страна, изискването може да бъде: потребителят създава своя акаунт и продавачът го потвърждава и одобрява. За тези функционалности в базата данни се изпълнява съответна заявка.

Пример: Да се създаде таблица „Groceries“с три колони -„ID“(първичен ключ)„Name“ и „Quantity“(от билет 6)

4. Дефинира и създава шаблонен изглед – 8т.

TemplateView е общ изглед, базиран на класове, който помага на разработчиците да създадат изглед за конкретен шаблон.

MVC шаблони:

Когато стартираме проект има възможност за *избор на шаблони* инициали, които отговарят на различни сценарии и изисквания. Когато се избере един от тези шаблони, проектът ще наследи някои структури и файлове, които служат като основа.

За да се избере шаблон, първо трябва да се създаде нов проект във VS(Visual Studio), *нов проект* и ние избира се **C #,** след това вида на проекта в този случай *ASP.NET MVC 4*.

Налични са 6 вида шаблони:

Empty: Този шаблон създава само основната структура, за да има скелета на приложението. Той се използва най -вече, когато вече се знае какъв ще бъде проекта и от какви компоненти се нуждае. Въпреки че е основен този шаблон *включва някои библиотеки на Javascript*.

Basic: Подобно на предишния подход, той създава само структурата на папките, необходима за изграждане на приложение *ASP.NET MVC.* Тъй като този шаблон е толкова основен, той изисква да се създаде всичко, което ще се използваме.

Internet Application: Той започва там, където е бил празният шаблон,

съдържа всичко необходимо, допълнително създава и **AccountController**, тоест контролер, който да обработва всичко, свързано с регистрацията на потребителя и управлението на сесията.

Intranet Application: Работи подобно на **Internet Application**, само че се различава по това, че е конфигуриран да използва удостоверяване *Базиран на Windows*. Това са софтуерни приложения за данни, които се използват предимно във вътрешната мрежа на организация.

Mobile Application: Това е друг вариант на интернет приложението, но този шаблон ни носи *Jquery Mobile Javascript рамка*, която е фокусирана върху мобилни платформи като мобилни телефони и таблети, допълнително изгледите са оптимизирани за показване на мобилни устройства.

Web API: Както показва името му, този шаблон е специализиран за създаване **RESTful API.** Може бързо да се генерират уеб услуги, за да се захранват други приложения, необходими за комуникация с конкретната структура.

5. Различава автентикацията и авторизацията- 8т.

- **Автентикация** - процес на проверка на самоличността на потребител или компютър. Въпроси: Кой си ти? Как го доказваш? Поверителните данни могат да бъдат парола, смарт карта, външен маркер и т.н.

Комбинацията от потребителско име и парола е най-популярният механизъм за удостоверяване и е известна още като удостоверяване с парола. В web приложенията се създава форма с полета за въвеждане на потребителско име и парола. След това се сравняват данните, въведени от потребителя, със стойностите, съхранявани преди това БД. При валидна комбинация на идентификационните данни на потребителят се дава достъп до акаунт.

Удостоверяването може да бъде еднофакторно и многофакторно. Има три вида фактори за удостоверяване, както в следните категории:

- Нещо, което се знае - парола, PIN;
- Нещо, което имаме – смартфон, e-mail;

- Нещо, което см - биометрична автентификация (пръстов отпечатък, глас, ирис);

▪ Авторизация - процес на определяне на това, което на потребителя е разрешено да прави на компютър или мрежа. Въпроси: Какво можете да правите? Можете ли да видите тази страница?

В web приложенията могат да се контролират нивата на привилегия за процесите, които изпълнява приложението от гледна точка на комуникацията с БД. Авторизацията е процес, чрез който сървърът определя дали клиентът има разрешение за използване на ресурс или достъп до файл. Това обикновено е съчетано с автентикация, така че сървърът да има някаква представа кой е клиентът, който иска достъп.

6. Диференцира сесиите от бисквитките – 8т.

- **Cookies** - Малък файл с обикновен текст без изпълним код. Изпраща се от сървъра до браузъра на клиента. Съхранява се от браузъра на устройството на клиента (компютър, таблет и т.н.) . Съхранява малка част данни за конкретен клиент и уеб сайт . Използват се за Управление на сесиите - вход, колички за пазаруване, резултати от игри или нещо друго, което сървърът трябва да запомни; Персонализация; Потребителски предпочитания, теми и персонализирани настройки; Проследяване; Записване и анализ на поведението на потребителя.
- **Структура** - бисквитката се състои от име, стойност и атрибути (незадължително). Атрибутите: Двойки ключ-стойност с допълнителна информация. Не са включват в заявките. Използват се от клиента за контрол на бисквитките.
- **Обхват** - определя се от атрибутите Domain и Path. Domain – определя уебсайта, към който принадлежи бисквитката. Path – Указва URL път, който трябва да съществува в
искания ресурс, преди да бъде изпратена.
- **Живот** - Определя се от атрибутите Expires и Max-Age. Expires – определя датата, на която браузърът трябва да изтрие бисквитката. По подразбиране бисквитките се изтриват след края на сесията. Max-Age – интервал от секунди преди бисквитката да бъде изтрита.

Видове cookies :

- **Session Cookie** (временна бисквитка) - Когато дадена бисквитка няма дата или период на валидност, тогава тя е сесийна/временна. Браузърът разпознава дадена бисквитка като сесийна, ако към нея липсват атрибутите Expires и Max-Age. След като посетителят затвори браузъра си, сесийната бисквитка се изтрива. Сайтът може да премахва от браузъра сесийни бисквитки, например когато потребителят излезе от профила си в сайта или пък остане неактивен за определен период от време.
- **Persistent/Tracking Cookie** (постоянна/проследяваща бисквитка) - Когато бисквитката има дата и/или период на валидност, тогава тя може да се определи като „постоянна“. Това означава, че при затваряне на браузъра бисквитката не се изтрива.Постоянните бисквитки също може да служат на сайтовете за запомняне и различаване на посетителите си. Най-често с цел персонализиране, събиране на статистики за достъпа, подобряване на използваемостта на сайта и потребителското изживяване. Както и за други цели, които не изискват идентифицирането конкретно на потребителите чрез тяхна лична или удостоверяваща информация (потребител, парола).
- **Third-party Cookie** (бисквитка, получена от външен сайт/трета страна) - Разновидност на бисквитките са тези от „трета страна“. По-агресивните бисквитки от трета страна са тези на рекламните компании, които ги използват за събиране на информация и проследяване на поведението и активностите на посетителите, с цел показване на подходящи реклами и предложения.Когато на дадена уеб страница се изтеглят и зареждат ресурси от външни сайтове, докато браузърът зарежда тази страница, на заден фон той прави отделни заявки и към тези външни сайтове.Някои от тези сайтове може да си задават собствени бисквитки и когато браузърът им изпрати заявка, в получения отговор от тях може да има и бисквитки. Тъй като бисквитките не са от самия сайт, който реално се зарежда от посетителя, тези бисквитки се наричат от „трета-страна“.

- **Сесии** - Начин за съхраняване на информация за потребител, която да се използва на много страници. Сесията представлява HTTP диалога клиент-сървър, съдържащ поредицата от HTTP заявки и HTTP отговори, асоциирани с един и същ посетител. Сесията, използвана от уеб сайта, най-често е имплементирана програмно, чрез програмния език, на който е написан самия сайт. Тази сесия е „програмистки похват“, чрез който към

HTTP протокола се добавя липсващия механизъм за запазване на състоянието на HTTP диалога клиент-сървър. Сесията може да се определи като времето, през което един посетител е в статус „разпознат“ за сайта. По време на сесията се запазват резултати, данни или статус от взаимодействието на потребителя със съдържанието в сайта. Чрез сесията сайтът може да запомня състоянието на HTTP диалога клиент-сървър. За сесии, създадени например с PHP, се използват сесийни бисквитки, които пренасят идентификатора на сесията между сайта и посетителя.

Браузърът знае едно нещо със сигурност, свързано с понятието за сесия – когато види сесийна бисквитка, тоест такава без дата и период, да я означи като сесийна и да я премахне от паметта си, когато процесът му бъде прекратен (затваряне на браузъра). Браузърът не се интересува каква информация съдържа и за какво се използва самата бисквитка. За него това е просто поредната HTTP бисквитка, която ще обработи според наличните към нея атрибути и зададените от потребителя настройки (в сигурност и контрол върху записването и изпращането на бисквитки).

7. Обяснява и дава пример за различните уязвимости в сигурността на уеб приложенията. Прави изводи за предотвратяването им. (8 т.)

Уеб приложенията за уязвими от:
· SQL Injection
· XSS
· CSRF
· Parameter Tampering

SQL Injections - SQL Injector е техника за инжектиране на код който се използва за атака. Тя позволява да вземеш информацията на една база данни. Първите признаци за уязвимост се появяват когато потребителят който използва SQL Injection въведе знаци които не могат да се филтрират правилно.

XSS - Cross-site scripting (XSS) е често срещана уязвимост в уеб приложенията. Уеб приложенията показват JavaScript код, който се зпълнява в браузъра на клиента. Хакерите могат да поемат контрол над сесиите, бисквитките, паролите и т.н.

CSRF - Това е атака на уеб сигурност над HTTP протокола. Позволява изпълнението на неоторизирани команди от името на някой потребител. Потребителят има валидни разрешения за изпълнение на заявената команда.

Нападателят използва тези разрешения злонамерено, непознато за потребителя.

Parameter Tampering - манипулиране на параметри, обменяни между клиент и сървър. Променени низове за запитвания, тяло на заявка, бисквитки. Пропуснати валидации на данните, инжектирани допълнителни параметри.

8. Обяснява основните принципи на REST API (4т.)

Архитектурният стил на „REST“ прилага шест ограничителни условия, като същевременно дава свобода за дизайна и имплементацията на индивидуалните компоненти:

Клиент-сървър

Единният интерфейс разделя клиента и сървъра. Това означава, например, че клиента не се грижи за складирането на данни. Тази задача остава изцяло за сървъра, като по този начин се подобрява портативността на клиентския код (може да се използва в различни среди). Сървърът няма връзка с потребителския интерфейс и по този начин е по-семпъл и лесен за премащабиране. Клиентът и сървърът могат да бъдат заменяни или развивани независимо един от друг, стига това да не налага промяна на единния интерфейс помежду им.

Без статус на сесията

Следващото условие е на сървъра да не се запазват статуси на сесиите. Всяка заявка от клиента, съдържа в себе си нужната информация за нейната обработка, статуси на сесии се запазват единствено при клиента.

Кеширане

Клиентът има право да кешира (запазва) информация, получена в отговор от сървъра, за да я преизползва при последващи заявки. За тази цел сървърът трябва имплицитно или експлицитно да е посочил дали информацията в отговора може да се кешира, за да се избегнат случаи, в които клиентът получава грешна информация при бъдещи заявки. При правилно управление и използване на кеширането могат частично или напълно да се елиминират ненужни взаимодействия между клиента и сървъра, като по този начин се подобрява бързината и производителността.

Многослойна система

Обикновено клиентът не знае дали е свързан с крайния сървър или със сървър-посредник. Сървърите-посредници подобряват ефективността, като увеличават капацитета за обработване на заявки и предоставят споделени кешове. Също така те допринасят да подобряването на сигурността.

Код при поискване (незадължително)

Сървърът може временно да разшири функционалността, изпращайки код, който се изпълнява директно при клиента. Например клиентски скриптове, написани на JavaScript или компилирани компоненти като Java applets.

Единен интерфейс

Единният интерфейс между клиента и сървъра разделя и опростява архитектурата. По този начин всеки компонент може да се развива самостоятелно.

Единственото условие на REST архитектурата, което не е задължително е „Код по поискване“. Всяко приложение (услуга), изпълняващо на гореописаните условия, може да се нарече „RESTful“. Ако нарушава дори едно от условията, то не може да бъде считано за „RESTful“.

Всяка разпространена хипермедийна система, съответстваща на архитектурния стил на „REST“ притежава нужната производителност, мащабируемост, опростеност, гъвкавост, видимост, портативност и надеждност.

9. Възпроизвежда и обяснява код за работа с REST API в JSON/XML формат- 6т.

Обяснете как се използва XMLHttpRequest, за да получите данни от сървъра:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "json_demo.txt");
xmlhttp.send();
</script>

</body>
</html>
```

json_demo.txt:

```
{
    "name":"John",
    "age":31,
    "pets":[
        { "animal":"dog", "name":"Fido" },
        { "animal":"cat", "name":"Felix" },
        { "animal":"hamster", "name":"Lightning" }
    ]
}
```

За да изпратим заявка до сървър, използваме методите open() и send() на обекта XMLHttpRequest:

xmlhttp.open("GET", "json_demo.txt");

xmlhttp.send();

Отговорът на сървъра може да се зададе чрез следните свойства:

responseText	**получава данните за отговора като низ**
responseXML	**получава данните за отговора като XML данни**

10. Възпроизвежда и обяснява код за използване на AJAX в REST API- 6т.

Обяснете как уеб страница може да извлече информация от XML файл с AJAX:

```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Get my CD collection</button>
<br><br>
<table id="demo"></table>
<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
```

```
  xhttp.onload = function() {
    myFunction(this);
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("CD");
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Когато потребител щракне върху бутона " Get my CD collection" се изпълнява функцията loadDoc(). Тя създава обект XMLHttpRequest, добавя функцията, която да се изпълни, когато отговорът на сървъра е готов, и изпраща заявката до сървъра.

Когато отговорът на сървъра е готов, се създава HTML таблица, възлите (елементите) се извличат от XML файла и накрая се актуализира елемента "демо" с HTML таблицата, пълна с XML данни