

מטרת סיום קורס - פיתוח צד שרת

מטרת המטלה

מטרת שלב זה בפרויקט הנגמר היא להקים את תשתית השירות, לישם עבודה נכונה בארכיטקטורת שלוש שכבות (Tier Architecture-3), ולבצע תקשורת מלאה בין בסיס הנתונים, השירות והלקוח (Client). עלייכם לישם את הלוגיקה העסקית של הפרויקט הקבוצתי שלכם.

הנחיות כלליות

- המטלה מתבצעת **בקבוצות הפרויקט**.
- יש לישם את הקוד בשפת C#.
- הגישה לנגנים תתבצע אך ורק באמצעות Entity ADO.NET (אין להשתמש ב-Framework).
- יש להקפיד על עקרונות כתיבת קוד נקי וחילוקה ברורה לשכבות.

חלק 1: מסד הנתונים

עליכם לתוכנן ולהקם את מסד הנתונים עבור הפרויקט שלכם ב-SQL Server.

1. צרו את הטבלאות הרלוונטיות לשויות בפרויקט.
2. הקפידו על קשרי גומלין (Relationships) ומפתחות (PK, FK).
3. הקפידו על כתיבת פרוצדורות.

חלק 2: בניית שכבות השירות (BL & DAL)

יש לבנות את הלוגיקה של המערכת בחלוקת לשכבות:

- **שכבה (DAL - Data Access Layer):** מחלקות האחראיות על התחברות ל-DB, הרצת SqlConnection (Select, Insert, Update, Delete) באמצעות SqlCommand-SqlDataReader-
SqlCommand.
- **שכבה (BL - Business Logic):** מחלקות המכילות את הלוגיקה העסקית, בדיקות תקינות (Validations) ועיבוד נתונים לפניה שליחתם ל-DB או חזרתם לקונטROLLER. השכבה תקרא לפונקציות מה-DAL.

חלק 3: יצירת API

- יש ליצור לפחות 4 קונטROLLERים (Controllers) שונים המיצגים 4 שיטות מרכזיות במערכת שלכם (לדוגמה: משתמשים, מוצרים, הזמינות, הודעות).
- כל קונטROLLER צריך להשך נקודות קצה (End Points) המשמשות בפועל HTTP מתאימים (GET, POST, PUT, DELETE).
- הקונטROLLER יפנה אך ורק לשכבה ה-BL לקבלת מידע.

חלק 4: ממשק משתמש

- צרו דפי HTML מתאימים ל- API שיוצרתם בחלק הקודם.
- כתבו קוד JavaScript המשתמש בפקודת fetch כדי לפנות לכל אחד מ-4 הקונטROLLERים שיצרתם. **זכור לעבד כפי שנלמד בכיתה:**
 - קובץ [index.js](#)ראשי אליו מייבאים את הפונקציות השונות. בקובץ זה יהיו אירועים.
 - קובץ [index.css](#) מותאים לכל ישות עם פונקציות רלוונטיות עברורה.
- הציגו את הנתונים המתקיים מהשרת באופן נגיש לעין ומעוצב באמצעות CSS.

חלק 5: העלאת ותצוגת קבצים

עליכם לאפשר העלאת קובץ (תמונת או מסמך) עבור אחת מהশכבות שבחרתם (לדוגמה: הוספה לתמונה למשתמש או ל מוצר).

1. **שמירה:** הקובץ הפיזי ישמר בתיקייה ייעודית בשרת (למשל `images` בתוך `wwwroot`), ואילו בבסיס הנתונים (DB) ישמר רק נתיב הקובץ (Path/Filename) כשדה מסווג מחרוזת (`NVARCHAR`).
2. **API:** יש למשוך פונקציה בקונטROLLER המקבלת קובץ (`FormFile`), שומרת אותו בשרת, ומעדכנת את ה-DB בנתיב החדש.
3. **Client:** בטופס ה-HTML, יש להוסיף שדה `input type="file"` ולהציג את התמונה / קובץ שהועלה לאחר שלילת הנתונים.

הוראות הנשה

1. יש להניש את המטלה **שבוע הבא**.
2. יש להעלות קובץ ZIP לטיית ההנשה במודול המכיל את כל הקוד וכן קישור לניטהאב.

מחוון הבדיקה

הפקחת ניקוד (דוגמאות)	קriterיונים לניקוד מלא (100%)	משקל	קטgorיה
<p>(20- נק') חסר ידע על הקוד.</p> <p>(10- נק') חסר הבנה למה השתמשו בפרוצדורות.</p>	שליטה מלאה בקוד, הבנת הזרימה, ומענה על שאלות תיאורתיות (כולל הסבר על יתרונות SP).	20%	הגנה על הפרויקט (ביקורת בע"נ)
<p>(15- נק') כתיבת שאילתות SQL "ריגילות" בקוד במקום SP.</p> <p>(5- נק') לוגיקה עסקית מורכבת בתוך ה-SP במקום ב-BL.</p>	חובה: כל הפניות ל-DB מבוצעות דרך Procedures בלבד. אין כתיבת שאילתות SQL (SELECT/INSERT .#C-#) בתוך קוד שימוש בפרמטרים לפרוצדורה.	15%	בסיס נתונים ופ्रוצדורות Stored) (Procedures
<p>(10- נק') השארת חיבורים פתוחים.</p> <p>(5- נק') שימוש תחביר בקריאה לפרוצדורה.</p>	הגדרת CommandType.StoredProcedure. שימוש ב-giving וסגירת חיבורים. קראית נתונים תקינה (Reader).	15%	ימוש ADO.NET ב-C#
(10- נק') ערבות שכבות.	הפרדה ל-API, BL, DAL. הكونטROLLER נקי מלוגיקה.	15%	ארQUITטורה (N-Tier)
(5- נק') חסר בקונטROLLERים.	4 קונטROLLERים, מימוש מלא של CRUD	15%	היקף פונקציונליות
(5- נק') שימרת קובץ ב-DB.	העלאת קובץ, שמירה בתיקייה, שימרת נתיב ב-DB.	10%	טיפול בקובציים (File Upload)
(5- נק') ממשק לא עובד.	קוד SJ תקין (Fetch), תצוגה ויזואלית.	10%	אינטרגרציית צד לקוח (Client)

בנק שאלות הכנה להגנה על הפרויקט

חלק א': ארכיטקטורה ומבנה (Architecture)

1. הסבירו את ארכיטקטורת N-Tier שבניתם. מה התפקיד של כל שכבה (DAL, BL, API)?
2. למה אנחנו לא כותבים את הקוד שניגש למסד הנתונים ישירות בתוך ה-Controller? מה הבעה עם זה?
3. תארו את זרימת המידע (Data Flow) מרגע של לקוח לוחץ על כפתור באתר ועד שהמידע נשמר בטבלה. אילו מחלקות ופונקציות מופעלות בדרך?
4. אם נרצה מחר להחליף את מסד הנתונים מ-SQL Server 7, Oracle, באילו שכבות נצטרך לגעת ובאיזה לא? מדוע?

חלק ב': מסד הנתונים ו-Stored Procedures

1. מדוע השתמשנו ב-Stored Procedures במקום לכתוב שאילתות SQL רגילות (inline SQL) בתוך הקוד? ציינו לפחות שני יתרונות.
2. איך הקוד ב-C# "ידע" שהוא מפעיל פרוצדורה ולא סתם פקודה טקסט? (רמז: CommandType).
3. היכן מוגדרת מחזורת החיבור (Connection String)? האם זה בטוח לשמר אותה בקוד עצמו? איפה נהוג לשמר אותה בפרויקט אמיתי?
4. מהם היתרונות של שימוש בפרמטרים (SqlParameter) מול שרשור מחזרות?

חלק ג':นำไปס טכני (#ADO.NET & C)

1. מה המשמעות של הפקודה `using` סביב יצירת ה-SqlConnection? מה יקרה אם תהיה שנייה (Exception) בתוך הבלוק ולא נשימוש ב-`using`?
2. מה ההבדל בין `ExecuteScalar`, `ExecuteNonQuery`, `ExecuteReader`? מתי נשימוש בכל אחד מהם?
3. כיצד אתם מתמודדים עם שגיאות? אם ה-DB למטה, מה המשתמש יקבל? (אם התוכנה תקורים או תחזיר הודעה מסוימת?)
4. הראו לי בקוד: כיצד מנעתם SQL Injection?

חלק ד': טיפול בקבצים (File Upload)

1. כה משתמש מעלה תמונה, מה בדיק נשמר בסיס הנתונים ומה נשמר בתיקיה בשרת?
למה לא לשמור את הקובץ עצמו בטור ה-DB?
2. כיצד צד הלקוח (Client) שולח את הקובץ? (הסבירו את השימוש ב-FormData).
3. איך דאגתם שלא יוצר מצב שבו שני משתמשים מUpload קובץ עם אותו שם (למשל image.jpg) והאחד דורס את השני?

חלק ה': שאלות "מה אם" (Scenarios)

(שאלות אלו בודקות הבנה عمוקה ויכולת דיבאנינג)

1. "אם אני מוחק עכשו שורה בטבלה ב-MySQL ידנית, האם האתר שלכם יקרוס כהונסה להציג אותה, או שידע להתמודד עם זה?"
2. "אם אני משנה את שם הפרטצורה ב-MySQL, מה עלי לשנות בקוד ה-C#?"
3. "הציבו על שורת קוד אחת שאתם נאים בה במיוחד, או שהיה לכם קשה במיוחד לכתוב אותה, והסבירו למה".

טיפים להגנה מוצלחת

- **דעו את הקוד:** אל תגידו "את זה יosi כתב אז אני לא יודע". כל חבר צוות אחריו על כל הפרויקט.
- **השתמשו במונחים מקצועיים:** במקום להגיד "הדבר הזה שמתחבר", אמרו "DAL Layer" . "Connection Object".
- **היו כנים:** אם אתם לא יודעים תשובה, עדיף לומר "אני לא בטוח כרגע, אבל אני משער ש..." מאשר לנסות להמציא תשובה לא הגיונית.

בהצלחה!!