**1 ) הגדרת תחום ו־MVP**

מיני־בנק: **לקוחות, חשבונות, ותנועות** (הפקדה/משיכה/העברה). תפקידי מערכת: **Public**, **User**, **Admin**.  
MVP כולל:

* Login: Form in angular an Basic Authentication in the API
* דשבורד משתמש: יתרות + היסטוריית תנועות (עם סינון/מיון בסיסי).
* יצירת תנועה: הפקדה/משיכה/העברה בין חשבונות.
* מסך Admin : רשימת לקוחות, חסימה/הפעלה.
* Public : דף נחיתה פשוט.

**2 ) סיפורי משתמש עם Acceptance Criteria**

**Public**

1. **כמבקר** אני רוצה לראות דף נחיתה כדי להבין מה האפליקציה עושה.  
   **AC** : דף /home פתוח כפתור “Login”.

**User**

1. **כמשתמש** אני רוצה להתחבר כדי לראות את החשבונות שלי.  
   **AC**: POST /auth/login מחזיר User+Role; ב־Angular נשמר jwt/token\* או סימולציה (לצורך המבחן אפשר לשמור “session marker” בלוקל־סטורג’).
2. **כמשתמש** אני רוצה לראות יתרה והיסטוריית תנועות לכל חשבון.  
   **AC**: GET /accounts/my מחזיר חשבונות; GET /transactions?accountId=... מחזיר הרשומות עם פאג’ינציה.
3. **כמשתמש** אני רוצה לבצע הפקדה/משיכה.  
   **AC**: POST /transactions/deposit|withdraw עם סכום>0, עדכון יתרה אטומי.
4. **כמשתמש** אני רוצה להעביר כסף בין חשבונות (שלי או ללקוח אחר).  
   **AC**: POST /transactions/transfer טרנזקציה אחת עם שתי רשומות (Debit/Credit); כשל→רולבאק.
5. **כמשתמש** אני רוצה סינון/מיון בסיסי להיסטוריה.  
   **AC**: פרמטרים type, dateFrom, dateTo, sortBy, page.

**Admin**

1. **כאדמין** אני רוצה לראות רשימת לקוחות ולשנות סטטוס (Active/Blocked).  
   **AC**: GET /admin/customers + PUT /admin/customers/{id}/status.
2. **כאדמין** אני רוצה לפתוח חשבון ללקוח.  
   **AC**: POST /admin/accounts (CustomerId, Currency).

**3 ) מודל נתונים (DB)**

* **Customers**: Id, FullName, Email, IsActive, CreatedAt
* **Users**: Id, Username, PasswordHash, Role (Admin/User), CustomerId?
* **Accounts**: Id, CustomerId (FK), IBAN/Number, Currency, Balance (decimal(18,2)), IsActive, CreatedAt
* **Transactions**: Id, AccountId, Type (Deposit/Withdraw/TransferIn/TransferOut), Amount, CreatedAt, ReferenceId (לשרשור בין in/out), Description
* **AuditLogs**: Id, CorrelationId, Path, Method, StatusCode, DurationMs, UserName, CreatedAt

יחס: Customer 1..\* Accounts; Account 1..\* Transactions.  
אינדקסים מומלצים: Transactions(AccountId, CreatedAt DESC), Users(Username UNIQUE).

**4 ) ארכיטקטורת ה־API (.NET 8)**

**Solution structure (Clean-ish):**

* Bank.Domain – ישויות, Enums, ולוגיקת דומיין.
* Bank.Application – Interfaces (Services/Repositories), DTOs, Validators.
* Bank.Infrastructure – EF Core (Repositories), Migrations, DataSeed.
* Bank.Api – Controllers, Filters, Middlewares, DI.

**תלויות:**  
Controllers → Application.Services (Interfaces) → Infrastructure.Repositories (Interfaces→Impl) → EF Core.

**Interfaces חשובים:**

* IAuthService (Validate, Hash, GetUser)
* IAccountService (GetMyAccounts, GetByCustomer, OpenAccount)
* ITransactionService (Deposit, Withdraw, Transfer, GetPaged)
* ICustomerService (List, SetStatus)
* IUnitOfWork (Begin/Commit/Rollback)
* IAuditLogger כתיבת Audit

**Middleware מותאם:**

* CorrelationIdMiddleware – יוצר/מעביר X-Correlation-Id.
* RequestResponseLoggingMiddleware – לוג מובנה Body עד N תווים , משך, סטטוס.
* ExceptionHandlingMiddleware – מיפוי חריגות ל־ProblemDetails.

**Filters מותאמים:**

* ValidateModelFilter – עוצר 400 עם פרטי השגיאות.
* PerformanceActionFilter – מודד משך פעולה ומוסיף Header X-Elapsed-Ms.

**Authentication = Basic Auth**

* Header Authorization: Basic base64(username:password)
* BasicAuthenticationHandler מותאם שמוודא מול IAuthService.
* Authorize(Roles="Admin") / Authorize(Roles="User").

**מדיניות שגיאות אחידה (ProblemDetails):**

* 400 – ולידציה.
* 401/403 – הרשאות.
* 404 – לא נמצא.
* 409 – התנגשות (יתרה לא מספיקה).
* 500 – שגיאה כללית.

**5 ) חוזי API דוגמאות תמציתיות**

**Auth**

* POST /api/auth/login  
  Request: { "userName": "rami", "password": "1234" }  
  Response: { "userName":"rami","role":"Admin","customerId":1,"token":"(אופציונלי/דמה)" }

**Accounts**

* GET /api/accounts/my → של המשתמש המחובר.  
  Response: [{ "id":1,"number":"IL12...","currency":"ILS","balance":1200.50 }]
* POST /api/admin/accounts (Admin)  
  Request: { "customerId": 5, "currency": "USD" }

**Transactions**

* GET /api/transactions?accountId=1&type=&dateFrom=&dateTo=&sortBy=createdAt\_desc&page=1&pageSize=20  
  Response: { "items":[...], "total": 125 }
* POST /api/transactions/deposit  
  { "accountId":1,"amount":100.00,"description":"Cash" }
* POST /api/transactions/withdraw  
  { "accountId":1,"amount":50.00,"description":"ATM" }
* POST /api/transactions/transfer  
  { "fromAccountId":1,"toAccountId":7,"amount":25.00,"description":"Rent" }

**Customers (Admin)**

* GET /api/admin/customers?query=&page=1&pageSize=20
* PUT /api/admin/customers/{id}/status  
  { "isActive": true }

**6 ) אפליקציית Angular – תכנון**

**מבנה פרויקט**

src/

app/

core/ (services, interceptors, guards, models)

shared/ (components, directives, pipes)

features/

public/ (lazy) -> /home

auth/ (lazy) -> /login

user/ (lazy) -> /dashboard, /accounts/:id

admin/ (lazy) -> /admin

store/ (NgRx: reducers, actions, effects, selectors)

**Routing & Guards**

* /home – Public.
* /login – Public (Redirect אם Logged).
* /dashboard – Guard AuthGuard (Role: User|Admin).
* /admin – Guard AdminGuard.
* Fallback \*\* → NotFound.

**State (NgRx)**

* auth: { userName, role, customerId, isAuthenticated }
  + Actions: login, loginSuccess, loginFailure, logout
  + Effects: login$ → קורא ל־AuthService
  + Persistence: Hydration מ־localStorage
* accounts: { list, loading }
  + Actions: loadMyAccounts, success/failure
  + Effects: loadMyAccounts$
* transactions: { byAccountId: { [id]: PagedResult }, loading }
  + Actions: loadTxPage, deposit, withdraw, transfer (+success/failure)
  + Effects: קריאות API ועדכון אחרי פעולה
* customers (Admin): טעינת לקוחות + שינוי סטטוס

**Change Detection**

* כל הקומפוננטות ב־OnPush.
* שימוש ב־async pipe ו־trackBy.

**Lazy Loading**

* כל מודול פיצ’ר Lazy (loadChildren).

**Parent-Child Communication**

* דוגמא: AccountListComponent (Parent) → AccountCardComponent (Child) עם @Input() ו־@Output() (למשל “Select Account”).

**Services**

* auth.service.ts, account.service.ts, transaction.service.ts, customer.service.ts  
  (הקריאות עוברות דרך Interceptor שמוסיף Authorization Header אם צריך).

**Interceptor**

* auth.interceptor.ts: מוסיף Authorization: Basic ... (לצורך המבחן אפשר לבנות את ה־Base64 מראש מהטופס, או לשמור token דמה).
* error.interceptor.ts: טיפול גלובלי בשגיאות + הודעות טוסט.

**Forms**

* Login (Reactive).
* Transfer Form (Reactive) עם Validators (amount > 0, לא אותו חשבון, מספיק יתרה – בדיקת API/חישוב מקומי).

**Custom Directive**

* appAutofocus או appDebounceInput (שימושי בסינון היסטוריה).

**Pipes**

* currencySymbol (ממפה ILS/USD/EUR לסמל),
* transactionTypeText (תקצור Deposit/Withdraw/Transfer).

**Local Storage**

* שמירת auth slice (userName/role/customerId/isAuthenticated).
* אפשר גם cache קל ל־accounts (תלוי בזמן).

**7 ) איורים לארכיטקטורה מה נסביר בראיון**

* **Interfaces בכל שכבה** → מאפשר Mocking ו־DI אמיתי.
* **Middleware + Filters** → הפרדה בין Cross-Cutting Concerns ללוגיקת דומיין.
* **UnitOfWork** ב־Transactions → עקביות בטרנזקציות בנקאיות.
* **NgRx** → State שקוף, Effects לקריאות API, Immutable state, Time-travel (תאורטית).

**8) תכנית “4 שעות” לביצוע בפועל**

**שעה 0–0:30 – Bootstrap & שלדים**

* יצירת סולושן 4 שכבות + פרויקט Angular.
* התקנת EF Core + יצירת מודלים/DbContext.
* Routing בסיסי ב־Angular + מודולים Lazy.

**שעה 0:30–1:30 – API ליבה**

* Repositories + Services (Accounts, Transactions).
* פעולות Deposit/Withdraw/Transfer עם טרנזקציה.
* Middleware (Correlation/Exception) + Filter ולידציה.
* Seeding: משתמש Admin + User, לקוח, חשבון, כמה תנועות.
* Basic Auth Handler.

**שעה 1:30–2:30 – Angular UX**

* Auth module: טופס Login + NgRx auth.
* User module: Dashboard + טעינת חשבונות ועסקאות (NgRx).
* Transfer form (Reactive) + קריאות API (Effects) + הודעות הצלחה/שגיאה.

**שעה 2:30–3:15 – Admin + ליטושים**

* מסך Admin: רשימת לקוחות + שינוי סטטוס.
* Pipes/Directive קטנים.
* Guards (Auth/Admin).
* Interceptor (Authorization).

**שעה 3:15–4:00 – קשיחות/הברקות**

* Pagination + סינון תנועות.
* לוגים ו־Audit.
* Tests נקודתיים (Service אחד).
* README (הרצה, משתמשי דמו, ארכיטקטורה קצרה).
* Git commits מסודרים.

**9) Git & איכות קוד**

* ענף main + feature/api-core, feature/angular-auth, …
* Commits קטנים עם תגים: [api], [web], [infra].
* README עם תרשים קצר + users לדמו:
  + Admin: admin / P@ssw0rd!
  + User: user1 / P@ssw0rd!

**10) מה נחשב “אקסטרה” אם נשאר זמן**

* Rate limiting (Minimal).
* CORS מדויק.
* Health checks (/health).
* Dockerfile לשני הפרויקטים.
* Cache לשאילתות סטטיות (MemoryCache) או NgRx Entity.