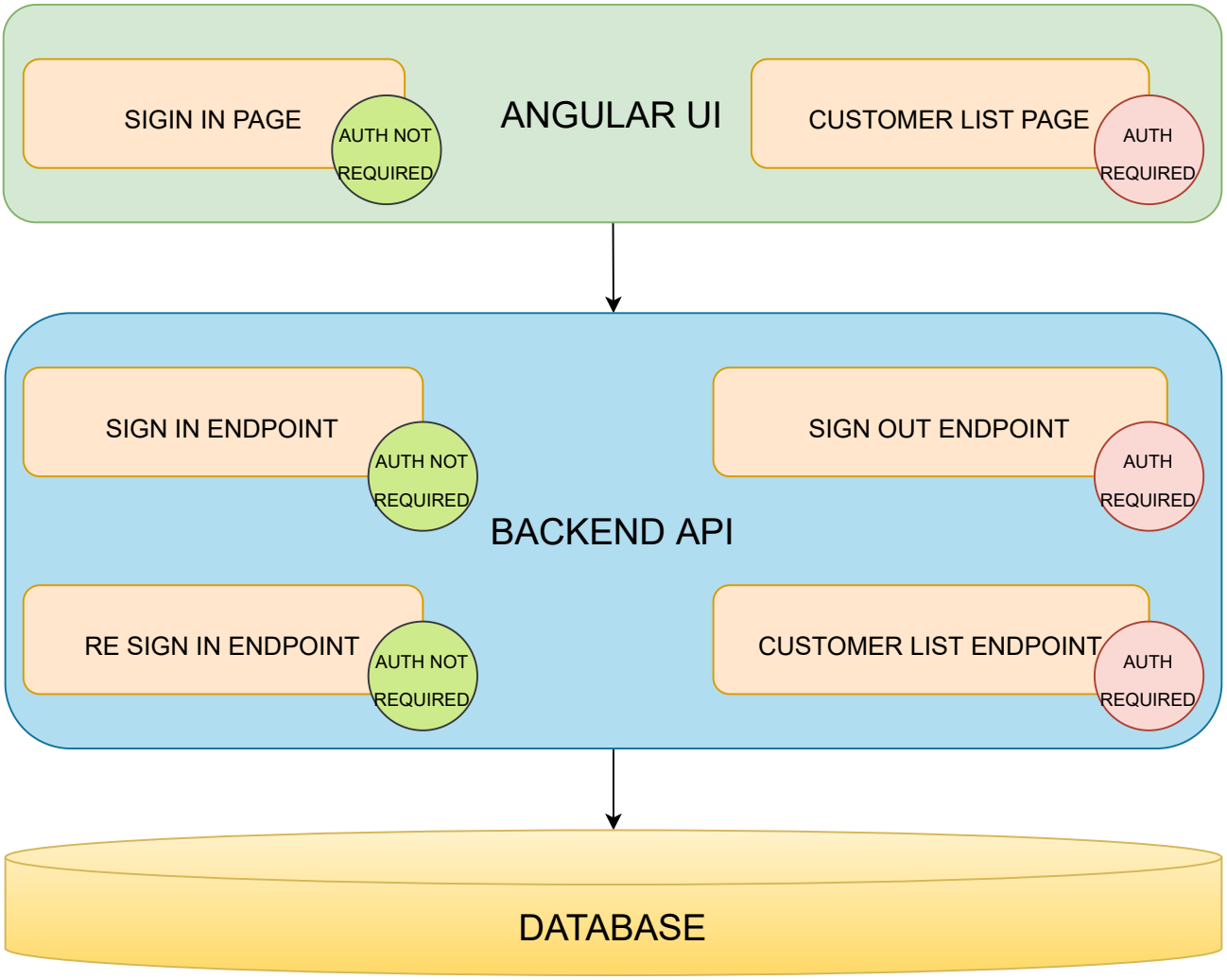
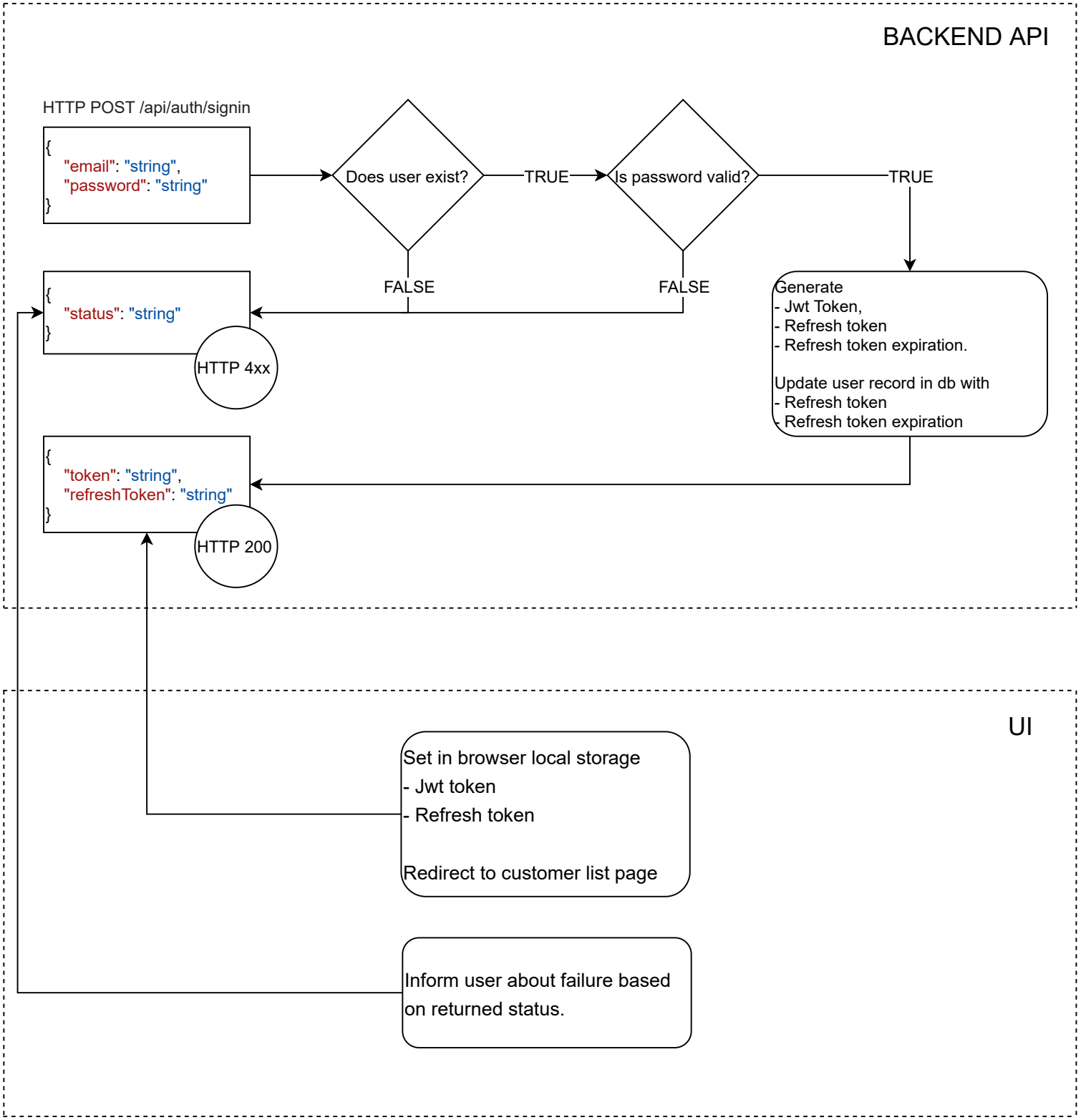
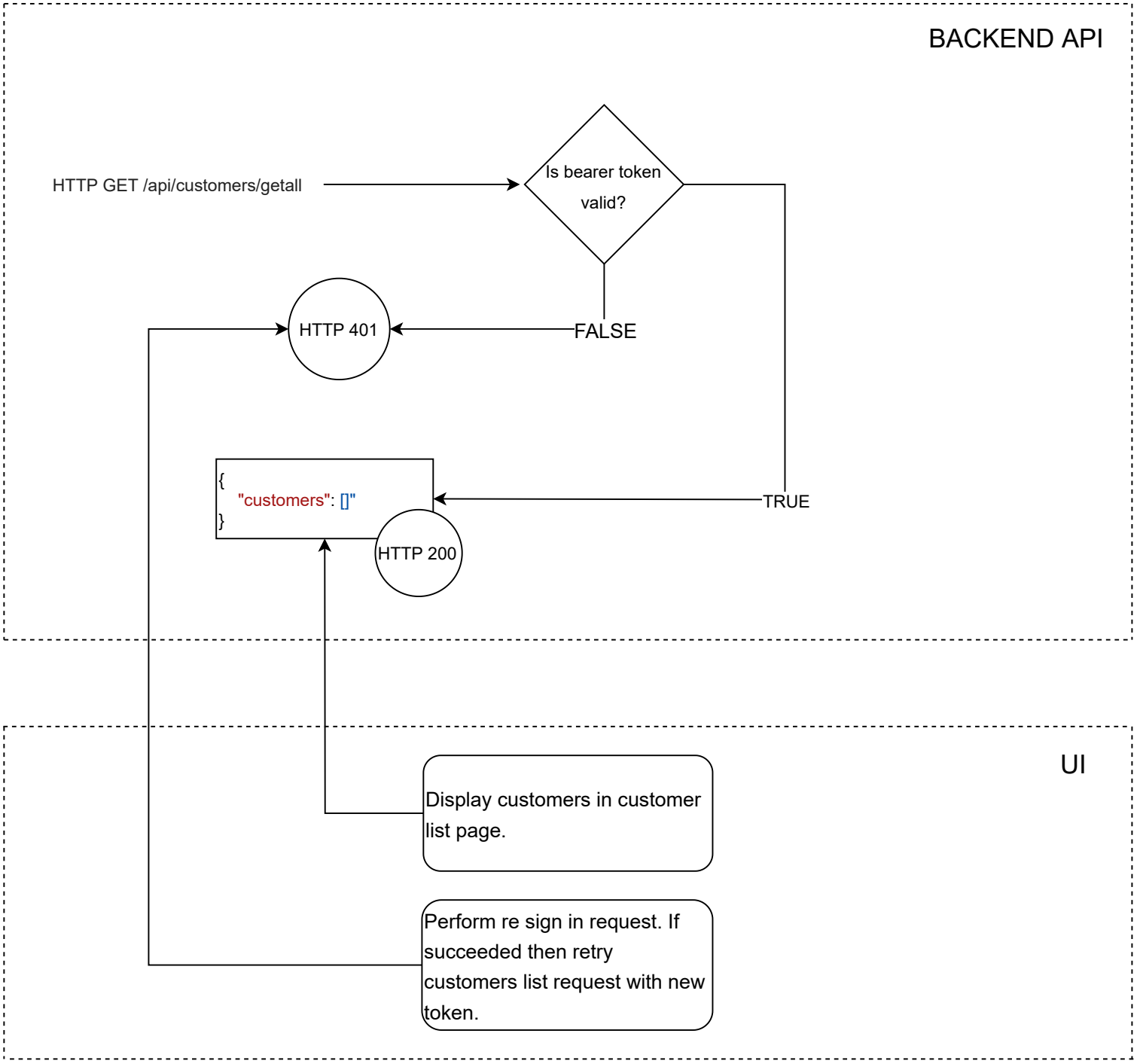
Angular JWT based Authentication


SIGIN IN PAGE
AUTH NOT REQUIRED
ANGULAR UI
CUSTOMER LIST PAGE
AUTH REQUIRED
SIGN IN ENDPOINT
AUTH NOT REQUIRED
SIGN OUT ENDPOINT
AUTH REQUIRED
BACKEND API
RE SIGN IN ENDPOINT
AUTH NOT REQUIRED
CUSTOMER LIST ENDPOINT
AUTH REQUIRED
DATABASE

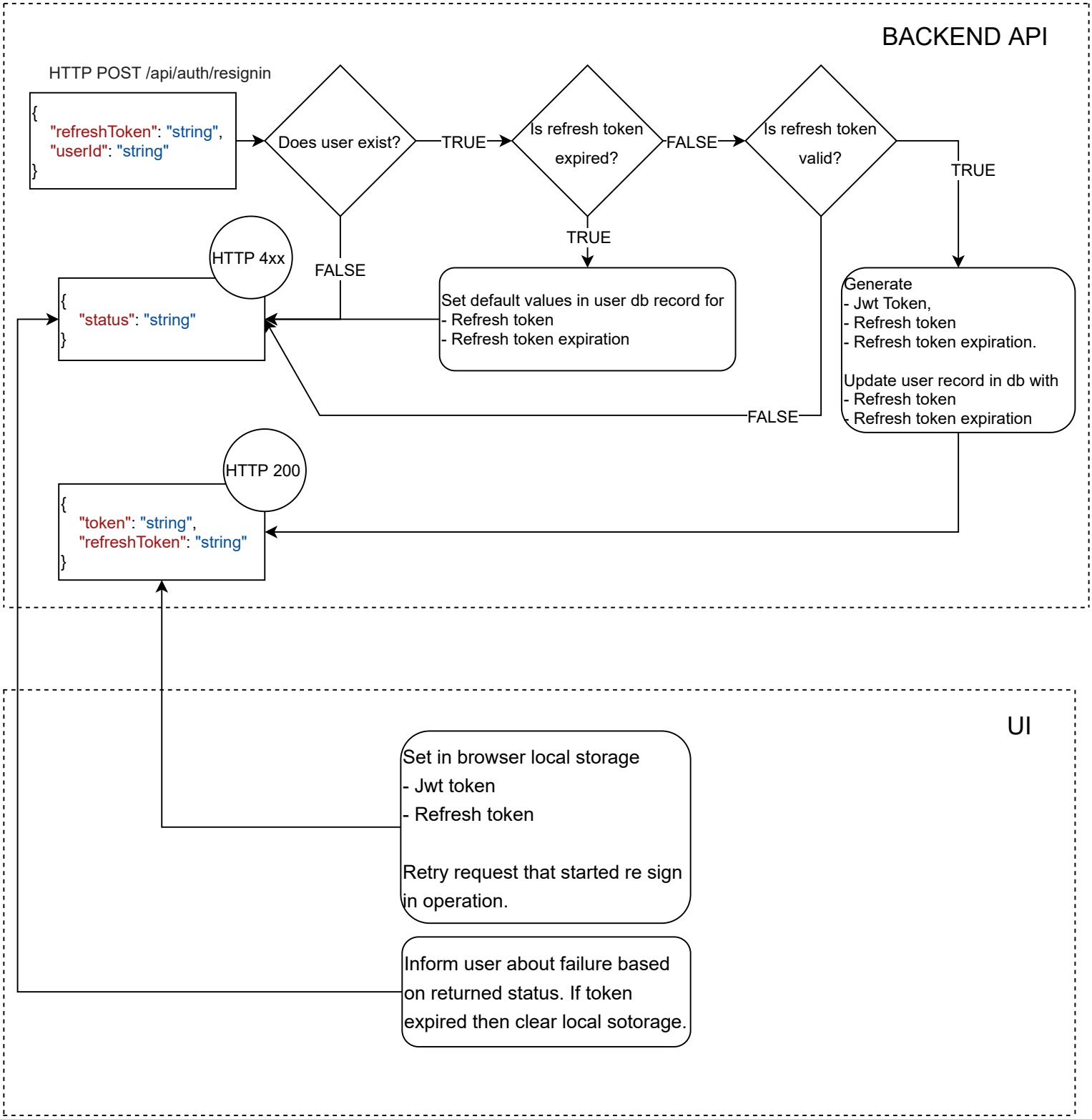
SIGN IN


BACKEND API
HTTP POST /api/auth/signin
{
"email": "string",
"password": "string"
}
Does user exist?
TRUE
Is password valid?
TRUE
FALSE
FALSE
{
"status": "string"
}
HTTP 4xx
Generate
- Jwt Token,
- Refresh token
- Refresh token expiration.
Update user record in db with
- Refresh token
- Refresh token expiration
{
"token": "string",
"refreshToken": "string"
}
HTTP 200
UI
Set in browser local storage
- Jwt token
- Refresh token
Redirect to customer list page
Inform user about failure based on returned status.

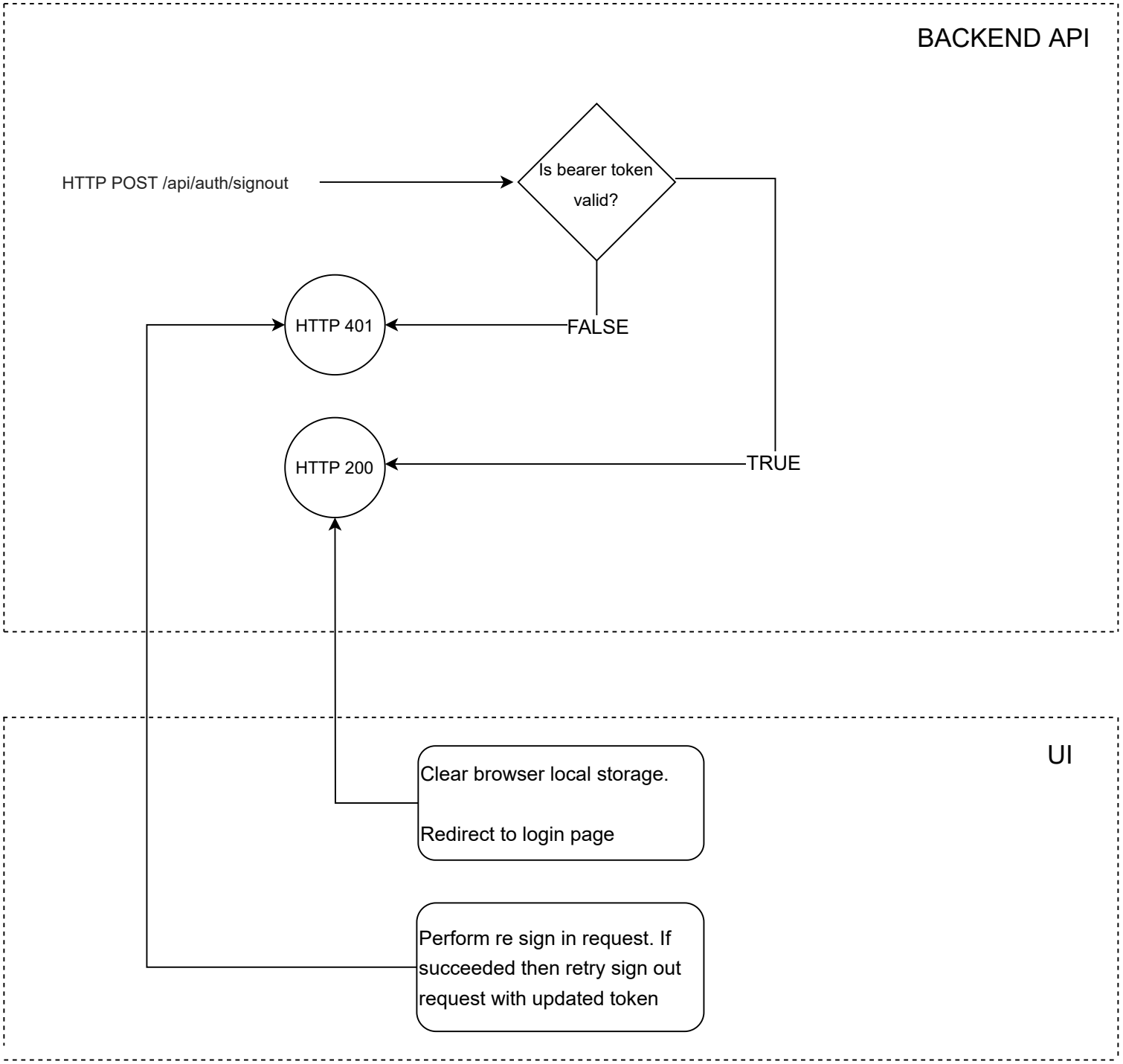
CUSTOMER LIST



RE SIGN IN

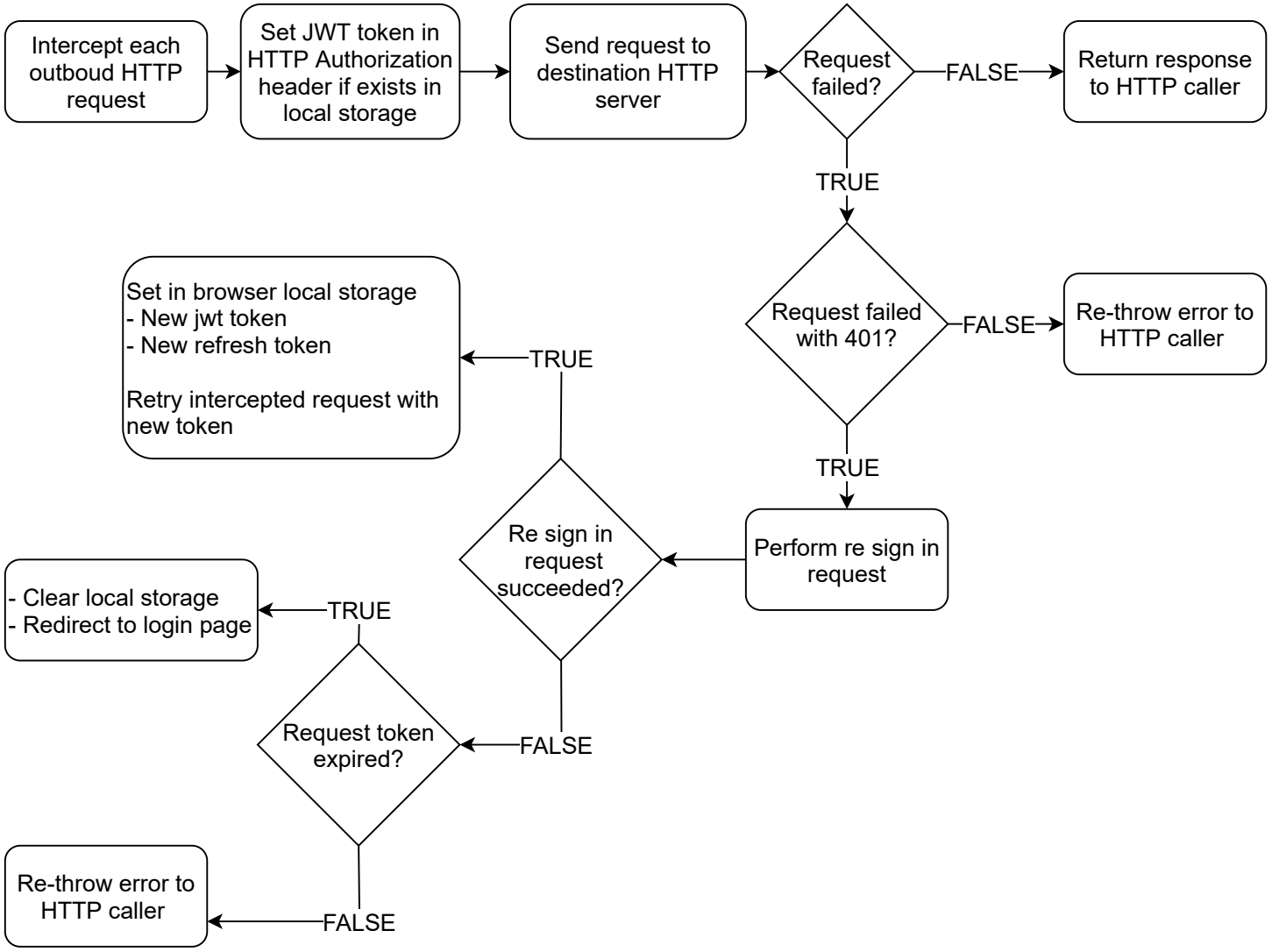

BACKEND API
HTTP POST /api/auth/resignin
{
"refreshToken": "string",
"userId": "string"
}
Does user exist?
TRUE
Is refresh token expired?
FALSE
Is refresh token valid?
TRUE
TRUE
FALSE
HTTP 4xx
{
"status": "string"
}
Set default values in user db record for
- Refresh token
- Refresh token expiration
Generate
- Jwt Token,
- Refresh token
- Refresh token expiration.
Update user record in db with
- Refresh token
- Refresh token expiration
FALSE
HTTP 200
{
"token": "string",
"refreshToken": "string"
}
UI
Set in browser local storage
- Jwt token
- Refresh token
Retry request that started re sign in operation.
Inform user about failure based on returned status. If token expired then clear local sotorage.

SIGN OUT


BACKEND API
HTTP POST /api/auth/signout
Is bearer token valid?
HTTP 401
FALSE
HTTP 200
TRUE
UI
Clear browser local storage.
Redirect to login page
Perform re sign in request. If succeeded then retry sign out request with updated token

ANGULAR HTTP INTERCEPTOR

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    CustomerListComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    HttpClientModule
  ],
  providers: [
    //Interceptor declaration
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



ANGULAR AUTH GUARDS

```
const routes: Routes = [
  {
    path: '',
    redirectTo: 'login',
    pathMatch: 'full'
  },
  {
    path: 'login',
    component: LoginComponent,
    canActivate: [AuthNotRequiredGuard]
  },
  {
    path: 'customer-list',
    component: CustomerListComponent,
    canActivate: [AuthRequiredGuard]
  }
];
```

```
export class AuthNotRequiredGuard implements CanActivate {

    constructor(
        private authStorage: AuthStorage,
        private router: Router) { }

    async canActivate(): Promise<boolean> {
        const token = this.authStorage.getAccessToken();
        if (token) {
            this.router.navigate(['customer-list']);
            return false;
        }
        else {
            return true;
        }
    }
}
```

```
export class AuthRequiredGuard implements CanActivate {

    constructor(
        private authStorage: AuthStorage,
        private router: Router) { }

    async canActivate(): Promise<boolean> {
        const token = this.authStorage.getAccessToken();
        if (token) {
            return true;
        }
        else {
            this.router.navigate(['login']);
            return false;
        }
    }
}
```

BACKEND AUTH CONFIGURATION

```
//Auth config start
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false;
    x.SaveToken = true;
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtSecretKey)),
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ClockSkew = TimeSpan.Zero
    };
});
//Auth config end
```

```
//Auth setup start
app.UseAuthentication();
app.UseAuthorization();
//Auth setup end
```

ConfigureServices()

Configure()

```
[Authorize]
[ApiController]
[Route("api/auth/")]
1 reference
public class AuthController : ControllerBase
{
    private readonly ICommandFactory _commandFactory;
    private readonly IProcessingInProgress _processingInProgress;

    0 references
    public AuthController(
        ICommandFactory commandFactory,
        IProcessingInProgress processingInProgress)
    {
        _commandFactory = commandFactory;
        _processingInProgress = processingInProgress;
    }

    [AllowAnonymous]
    [HttpPost("signin")]
    0 references
    public IActionResult SignIn(SignInRequestBody httpBody)
    {
        _commandFactory
            .CreateForSignIn(httpBody, _processingInProgress)
            .Handle();

        return _processingInProgress.ToActionResult();
    }
```