Changelog - August 2024 - .Net 8, AuthJS 5 and NextJS 14

Section 2

Lesson 8 - Creating the first micro service

- Callout added to advise using the -controllers switch when creating the webapi project, otherwise it creates the project as minimal API (@7:00)

Lesson 10 - Adding the entity classes

- Callout added to advise that the C# dev kit now automatically uses the correct namespace based on the folder (@1:46)
- Callout added to advise that “Intellicode” is no longer available as it has been discontinued by Microsoft. (@3:58)

Lesson 11 - Adding the Database context class

- Small update to advise where to find the nuget gallery which was updated to v1 (@1:58)
- Callout added to advise that AutoMapper 13 includes the Dependency injection extensions in the main package (@4:00)

Section 3

Lesson 23 - Creating the second micro services

- Callout added to advise using the -controllers switch when creating the webapi project, otherwise it creates the project as minimal API (@3:32)
- Callout added to advise that AutoMapper 13 includes the Dependency injection extensions in the main package (@8:06)

Lesson 24 - Adding the MongoDb server

- Callout added to correct the path of the mongo db data volume to mongodata:/data/db (@5:03)
- Callout added to advise that ‘docker compose down -v’ will remove named volumes along with container (@0:57)

Section 4

Lesson 42 - Adding a message outbox

- Callout added to point out the importance of using the same version of MassTransit.EntityFrameworkCore as MassTransit.RabbitMQ (@3:57)
- Callout added to point out all relational DBs are now supported for the Transactional outbox (@5:34)

Section 5

Lesson 51 - Reviewing and configuring our new project

- Callout added advising that Identity Server 7 uses .Net 8 so there is no need to upgrade any of the packages (@1:23)

Section 7

Lesson 77 - Dockerizing the Identity Service

- Update to the lesson added to advise students on Mac computers that the identity service port may need to change from 5000 to 5001 as Apple are using port 5000 for an AirPlay service (@3:51)

Lesson 78 - Debugging a .Net service in a docker container

- Lesson re-recorded as there are a few differences in setting up the debugger and in the code for the Identity Server diagnostics file.

Lesson 80 - Testing our docker containers

- Identity configuration updated for the options as it is a bit fussier in ID7 than previously to the following (@4:08)

```
// HostingExtensions.cs

if (builder.Environment.IsEnvironment("Docker"))
{
    options.IssuerUri = "http://localhost:5000";
}
```

Section 8

Lesson 83 - Creating the NextJS project

- Callout added advising minimum version of node.js required at time of update is 18.17 (@0:22)

Lesson 84 - Reviewing and simplifying the NextJS Project

- Callout added to advise logs may be slightly different in Next14 (@0:24)
- Callout added to advise the next.config.js is now next.config.mjs in NextJS 14+ (@7:44)

Lesson 85 - Creating a nav bar

- Callout & screenshot added advising of a setting in VS Code that has the auto closing tag functionality (@2:20)

Lesson 86 - Fetching data from the API

- Update provided to the lesson to demo how to turn on full fetch logging in the **next.config.mjs** to provide similar logging output to NextJS 13 (@4:23)

```
/** @type {import('next').NextConfig} */
const nextConfig = {
    logging: {
        fetches: {
            fullUrl: true
        }
    }
};

export default nextConfig;
```

Lesson 88 - Styling the auction cards

- Updated from @5:58 onwards.
- Can use the ‘relative’ for ‘aspect-video’ rather than installing @tailwindcss/aspect-ratio
- Update config in the next.config.mjs as the previous approach using ‘domains’ is deprecated.

```
// AuctionCard.tsx
import React from 'react'
import Image from "next/image";

type Props = {
    auction: any;
}

export default function AuctionCard({ auction }: Props) {
    return (
        <a href="#">
            <div className='relative w-full bg-gray-200 aspect-video rounded-lg overflow-
                <Image
                    src={auction.imageUrl}
```

```
                    fill
                    // example of meaningul alt tag
                    alt={`Image of ${auction.make} ${auction.model} in ${auction.color}`}
                    priority
                    className='object-cover'
                    sizes='(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 25vw'
                />
            </div>
            <div className='flex justify-between items-center mt-4'>
                <h3 className='text-gray-700'>{auction.make} {auction.model}</h3>
                <p className='font-semibold text-sm'>{auction.year}</p>
            </div>
        </a>
    )
}
```

```
// next.config.mjs

/** @type {import('next').NextConfig} */
const nextConfig = {
    logging: {
        fetches: {
            fullUrl: true
        }
    },
    images: {
        remotePatterns: [
            {protocol: 'https', hostname: 'cdn.pixabay.com'}
        ]
    }
};

export default nextConfig;
```

Lesson 89 - Adding a countdown timer to the auction card

- Updated to make it consistent with previous lesson update
- Changed the aspect ratio to make it consistent with what was previously used for the car images:

```
// Auction.Card.tsx
export default function AuctionCard({ auction }: Props) {
    return (
        <a href="#">
            <div className='relative w-full bg-gray-200 aspect-[16/10] rounded-lg overflo
                // rest of code omitted
```

Lesson 90 - Adding loading to the images

- Updated as ‘onLoadingComplete’ is deprecated and made consistent with previous lesson updates. Code change:

```
export default function CarImage({imageUrl}: Props) {
    const [isLoading, setLoading] = useState(true);

    return (
        <Image
            // omitted
            sizes='(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 25vw'
            onLoad={() => setLoading(false)}
        />
```

```
    )
}
```

Lesson 92 - Adding pagination to our list

- Update from @1:08 to show the updated flowbite docs and config for flowbite. Updated tailwind.config.ts:

```
import type { Config } from "tailwindcss";
import flowbite from 'flowbite-react/tailwind';

const config: Config = {
  content: [
    "./pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./components/**/*.{js,ts,jsx,tsx,mdx}",
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
    flowbite.content(),
  ],
  theme: {
    extend: {
      backgroundImage: {
        "gradient-radial": "radial-gradient(var(--tw-gradient-stops))",
        "gradient-conic":
          "conic-gradient(from 180deg at 50% 50%, var(--tw-gradient-stops))",
      },
    },
  },
  plugins: [
    flowbite.plugin()
  ],
};
export default config;
```

Lesson 93 - Using server functions in client components

- Removed demo of setting experimental server actions as they are no longer experimental.

Lesson 96 - Refactoring our code to use the zustand state

- Shallow is now depracted in Zustand. Updated lesson to demonstrate ‘useShallow’ in place of this (@2:36)

```
// Listings.tsx

import { useShallow } from 'zustand/react/shallow';

export default function Listings() {
    const [data, setData] = useState<PagedResult<Auction>>();
    const params = useParamsStore(useShallow(state => ({
        pageNumber: state.pageNumber,
        pageSize: state.pageSize,
        searchTerm: state.searchTerm
    })));
```

Section 9

Lessons 106-113

- All these lessons were re-recorded to accommodate changes required for using next-auth v5. To install use:

```
npm install next-auth@beta
```

- To update your project to use this please update/create the following files

```
// web-app/app/api/auth/[...nextauth]/route.ts

import { handlers } from "@/auth"
export const { GET, POST } = handlers
```

```
// web-app/auth.ts

import NextAuth, { Profile } from "next-auth"
import { OIDCConfig } from 'next-auth/providers'
import DuendeIDS6Provider from "next-auth/providers/duende-identity-server6"

export const { handlers, signIn, signOut, auth } = NextAuth({
    session: {
        strategy: 'jwt'
    },
    providers: [
        DuendeIDS6Provider({
          id: 'id-server',
          clientId: "nextApp",
          clientSecret: "secret",
          issuer: "http://localhost:5001",
          authorization: {params: {scope: 'openid profile auctionApp'}},
          idToken: true
        } as OIDCConfig<Omit<Profile, 'username'>>),
    ],
    callbacks: {
        async authorized({auth}) {
            return !!auth
        },
        async jwt({token, profile, account}) {
            if (account && account.access_token) {
                token.accessToken = account.access_token
            }
            if (profile) {
                token.username = profile.username
            }
            return token;
        },
        async session({session, token}) {
            if (token) {
                session.user.username = token.username;
                session.accessToken = token.accessToken;
            }
            return session;
        }
    }
})
```

```
// web-app/types/next-auth.d.ts

import NextAuth, { type DefaultSession } from "next-auth"
import {JWT} from 'next-auth/jwt';

declare module "next-auth" {
  interface Session {
    user: {
      username: string
    } & DefaultSession["user"]
```

```
    accessToken: string
  }

  interface Profile {
    username: string;
  }
}

declare module 'next-auth/jwt' {
  interface JWT {
    username: string
    accessToken: string
  }
}
```

```
// .env.local

AUTH_SECRET="7vgUxWjehgeKTOFH2dZu0zSeKP61o9gl0b1vuHCqeMo="
```

```
// web-app/app/actions/authActions.ts

'use server'

import { auth } from '@/auth'

export async function getCurrentUser() {
    try {
        const session = await auth();

        if (!session) return null;

        return session.user;
    } catch (error) {
        return null;
    }
}

// The getTokenWorkaround can be removed. We are populating the access_token in the sessi
// easily get from the auth() function
```

```
// web-app/app/actions/auctionActions.ts

'use server';

import { auth } from '@/auth';
import { Auction, PagedResult } from '@/types';

// getData omitted

export async function updateAuctionTest() {
    const data = {
        mileage: Math.floor(Math.random() * 10000) + 1
    }

    const session = await auth();

    const res = await fetch('http://localhost:6001/auctions/afbee524-5972-4075-8800-7d1f9
        method: 'PUT',
```

```
        headers: {
            'Content-type': 'application/json',
            'Authorization': 'Bearer ' + session?.accessToken
        },
        body: JSON.stringify(data)
    });

    if (!res.ok) return {status: res.status, message: res.statusText}

    return res.statusText;
}
```

```
// web-app/app/session/page.tsx

import { auth } from '@/auth'
import React from 'react'
import Heading from '../conponents/Heading';
import AuthTest from './AuthTest';

export default async function Session() {
    const session = await auth();
    return (
        <div>
            <Heading title='Session dashboard' />

            <div className='bg-blue-200 border-2 border-blue-500'>
                <h3 className='text-lg'>Session data</h3>
                <pre className='whitespace-pre-wrap break-all'>{JSON.stringify(session, n
            </div>

            <div className='mt-4'>
                <AuthTest />
            </div>
        </div>
    )
}
```

Section 10

Lesson 117 - Getting the auctions won

- Just a small update to make it consistent with previous lesson updates (demoing the next-auth.d.ts) @3:36
- Dropdown bug where clicking out didn’t remove the dropdown has been fixed so removed the element of the lesson mentioning this.

Lesson 118 - Creating an Auction form

- React datepicker now comes with a type definition file so removed the part of the demo that installed the types.

Lesson 122 - Creating a reusable date input

- Lesson re-recorded to use the slightly different approach required for using the React datepicker types:

```
import {useController, UseControllerProps} from "react-hook-form";
import {Label} from "flowbite-react";
import 'react-datepicker/dist/react-datepicker.css'
import DatePicker, { DatePickerProps } from 'react-datepicker';

type Props = {
    label: string;
    type?: string;
```

```
    showLabel?: boolean;
} & UseControllerProps & DatePickerProps

export default function DateInput(props: Props) {
    const {fieldState, field} = useController({...props, defaultValue: ''})
    return (
        <div className='mb-3'>
            {props.showLabel && (
                <div className='mb-2 block'>
                    <Label htmlFor={field.name} value={props.label} />
                </div>
            )}
            <DatePicker
                {...props}
                {...field}
                placeholderText={props.label}
                selected={field.value}
                className={`
                    rounded-lg
                    w-[100%]
                    flex flex-col
                    ${fieldState.error
                        ? 'bg-red-50 border-red-500 text-red-900'
                        : (!fieldState.invalid && fieldState.isDirty)
                        ? 'bg-green-50 border-green-500 text-green-900' : ''
                    }
                    `}
            />
            {fieldState.error && (
                <div className='text-red-500 text-sm'>{fieldState.error.message}</div>
            )}
        </div>
    );
}
```

Lesson 123 - Creating a fetch wrapper

- Re-recorded as now using the session access token rather than the workaround in previous version. The only change is in the getHeaders() function:

```
async function getHeaders() {
    const session = await auth();
    const headers = {
        'Content-type': 'application/json'
    } as any;
    if (session?.accessToken) {
        headers.Authorization = 'Bearer ' + session.accessToken
    }
    return headers;
}
```

Lesson 124 - Adding the create auction server action

- Updated to make the lesson consistent with previous lesson changes

Lesson 126 - Adding the auction details page content

- Added callout advising to use aspect-[16/10] rather then the aspect tailwind extension:

```
// web-app/app/auctions/details/[id]/page.tsx

<div className='w-full bg-gray-200 relative aspect-[16/10] rounded-lg overflow-hidden'>
```

```
    <CarImage imageUrl={data.imageUrl} />
</div>
```

Section 11

Lesson 131 - Creating the Bid Service

- Callout added to advise to use the -controllers switch when using .Net 8 (@0:41)

Lesson 137 - Adding the DTOs and Automapper

- Callout added advising that we can just use AutoMapper 13 as it comes with DI and we do not need the Dependency injection version (@1:21)

Section 13

Lesson 157 - Refactoring the auctions into a zustand store

- Small update to lesson to demo using the 'useShallow' rather than 'shallow' when getting the data from the aucton store @8:19

```
  // Listings.tsx

    const data = useAuctionStore(useShallow(state => ({
        auctions: state.auctions,
        totalCount: state.totalCount,
        pageCount: state.pageCount
    })))
```

Lesson 165 - Adding SignalR to the client app

- Re-recorded to better use SignalR using a ref instead of useState to store the connection. This prevents the connection from being renewed due to a route change or anything else that causes the useEffect to re-execute. This is the updated code for this lesson:

```
// web-app/app/providers/SignalRProvider.tsx
'use client'

import { useAuctionStore } from '@/hooks/useAuctionStore'
import { useBidStore } from '@/hooks/useBidStore'
import { Bid } from '@/types'
import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr'
import { useParams } from 'next/navigation'
import React, { ReactNode, useEffect, useRef } from 'react'

type Props = {
    children: ReactNode
}

export default function SignalRProvider({ children }: Props) {
    const connection = useRef<HubConnection | null>(null);
    const setCurrentPrice = useAuctionStore(state => state.setCurrentPrice);
    const addBid = useBidStore(state => state.addBid);
    const params = useParams<{id: string}>();

    useEffect(() => {
        if (!connection.current) {
            connection.current = new HubConnectionBuilder()
                .withUrl('http://localhost:6001/notifications')
                .withAutomaticReconnect()
                .build();

            connection.current.start()
```

```
                .then(() => 'Connected to notification hub')
                .catch(err => console.log(err));

            connection.current.on('BidPlaced', (bid: Bid) => {
                setCurrentPrice(bid.auctionId, bid.amount);
            })
        }
    }, [setCurrentPrice])

    return (
        children
    )
}
```

Lesson 166 - Adding the new bid to SignalR

- Re-recorded to fix a bug and also to use the useCallback function for the listener handler for adding a bid. Updated code:

```
// web-app/app/providers/SignalRProvider.tsx

'use client'

import { useAuctionStore } from '@/hooks/useAuctionStore'
import { useBidStore } from '@/hooks/useBidStore'
import { Bid } from '@/types'
import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr'
import { useParams } from 'next/navigation'
import React, { ReactNode, useCallback, useEffect, useRef } from 'react'

type Props = {
    children: ReactNode
}

export default function SignalRProvider({ children }: Props) {
    const connection = useRef<HubConnection | null>(null);
    const setCurrentPrice = useAuctionStore(state => state.setCurrentPrice);
    const addBid = useBidStore(state => state.addBid);
    const params = useParams<{id: string}>();

    const handleBidPlaced = useCallback((bid: Bid) => {
        if (bid.bidStatus.includes('Accepted')) {
            setCurrentPrice(bid.auctionId, bid.amount);
        }

        if (params.id === bid.auctionId) {
            addBid(bid);
        }
    }, [setCurrentPrice, addBid, params.id])

    useEffect(() => {
        if (!connection.current) {
            connection.current = new HubConnectionBuilder()
                .withUrl('http://localhost:6001/notifications')
                .withAutomaticReconnect()
                .build();

            connection.current.start()
                .then(() => 'Connected to notification hub')
                .catch(err => console.log(err));
```

```
        }

        connection.current.on('BidPlaced', handleBidPlaced);

        return () => {
            connection.current?.off('BidPlaced', handleBidPlaced)
        }

    }, [setCurrentPrice, handleBidPlaced])

    return (
        children
    )
}
```

Lesson 167 - Adding a toast for an auction created

- Updated to use handler in signalR for the createdAuction toast and be consistent with prior updates. Updated code in the provider (@5:38):

```
'use client'

import { useAuctionStore } from '@/hooks/useAuctionStore'
import { useBidStore } from '@/hooks/useBidStore'
import { Auction, AuctionFinished, Bid } from '@/types'
import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr'
import { User } from 'next-auth'
import { useParams } from 'next/navigation'
import React, { ReactNode, useCallback, useEffect, useRef } from 'react'
import toast from 'react-hot-toast'
import AuctionCreatedToast from '../conponents/AuctionCreatedToast'

type Props = {
    children: ReactNode
    user: User | null
    notifyUrl: string
}

export default function SignalRProvider({ children, user, notifyUrl }: Props) {
        // omitted

    const handleAuctionCreated = useCallback((auction: Auction) => {
        if(user?.username !== auction.seller) {
            return toast(<AuctionCreatedToast auction={auction} />, {
                duration: 10000
            })
        }
    }, [user?.username])

        // omitted

        connection.current.on('BidPlaced', handleBidPlaced);
        connection.current.on('AuctionCreated', handleAuctionCreated);
        connection.current.on('AuctionFinished', handleAuctionFinished);

        return () => {
            connection.current?.off('BidPlaced', handleBidPlaced);
            connection.current?.off('AuctionCreated', handleAuctionCreated);
            connection.current?.off('AuctionFinished', handleAuctionFinished);
        }
```

```
    }, [setCurrentPrice, handleBidPlaced, handleAuctionCreated, handleAuctionFinished, no

    return (
        children
    )
}
```

Lesson 168 - Adding a toast for an auction finished event

- Updated to be consistent with previous updates and using a callback handler in the Provider (@4:14).
  Updated SignalRProvider code:

```
'use client'

// imports omitted
import { getDetailedViewData } from '../actions/auctionActions'
import AuctionFinishedToast from '../conponents/AuctionFinishedToast'

type Props = {
    children: ReactNode
    user: User | null
    notifyUrl: string
}

export default function SignalRProvider({ children, user, notifyUrl }: Props) {
    const connection = useRef<HubConnection | null>(null);
    const setCurrentPrice = useAuctionStore(state => state.setCurrentPrice);
    const addBid = useBidStore(state => state.addBid);
    const params = useParams<{id: string}>();

    const handleAuctionFinished = useCallback((finishedAuction: AuctionFinished) => {
        const auction = getDetailedViewData(finishedAuction.auctionId);
        return toast.promise(auction, {
            loading: 'Loading',
            success: (auction) =>
                <AuctionFinishedToast
                    auction={auction}
                    finishedAuction={finishedAuction}
                />,
            error: (err) => 'Auction finished'
        }, {success: {duration: 10000, icon: null}})
    }, [])

    // omitted bidplaced and auction finished handlers

    useEffect(() => {
        if (!connection.current) {
            connection.current = new HubConnectionBuilder()
                .withUrl(notifyUrl)
                .withAutomaticReconnect()
                .build();

            connection.current.start()
                .then(() => 'Connected to notification hub')
                .catch(err => console.log(err));
        }

        connection.current.on('BidPlaced', handleBidPlaced);
        connection.current.on('AuctionCreated', handleAuctionCreated);
```

```
        connection.current.on('AuctionFinished', handleAuctionFinished);

        return () => {
            connection.current?.off('BidPlaced', handleBidPlaced);
            connection.current?.off('AuctionCreated', handleAuctionCreated);
            connection.current?.off('AuctionFinished', handleAuctionFinished);
        }

    }, [setCurrentPrice, handleBidPlaced, handleAuctionCreated, handleAuctionFinished, no

    return (
        children
    )
}
```

Lesson 169 - Disabling the auction finished form when the auction finishes

- Updated to maintain consistency with the previous lesson updates. No code changes.

Section 14 - Publishing locally to Docker Compose (original Section 14 moved into Legacy Section 14)

Lessons 173-178 were re-recorded. There is a new approach taken with dealing with Identity Server and instead of using extra_hosts and assigning a static IP to the identity server we use configuration in next-auth to be specific about where the ID Server is for internal comms from nextJS to IdentityServer and when it is for NextJS to Browser. The updated code in the auth.ts is:

```
import NextAuth, { Profile } from "next-auth"
import { OIDCConfig } from 'next-auth/providers'
import DuendeIDS6Provider from "next-auth/providers/duende-identity-server6"

export const { handlers, signIn, signOut, auth } = NextAuth({
    session: {
        strategy: 'jwt'
    },
    providers: [
        DuendeIDS6Provider({
            id: 'id-server',
            clientId: "nextApp",
            clientSecret: "secret",
            issuer: process.env.ID_URL,
            authorization: {
                params: { scope: 'openid profile auctionApp' },
                url: process.env.ID_URL + '/connect/authorize'
            },
            token: {
                url: `${process.env.ID_URL_INTERNAL}/connect/token`
            },
            userinfo: {
                url: `${process.env.ID_URL_INTERNAL}/connect/token`
            },
            idToken: true
        } as OIDCConfig<Omit<Profile, 'username'>>),
    ],
    callbacks: {
        async redirect({url, baseUrl}) {
            return url.startsWith(baseUrl) ? url : baseUrl
        },
        async authorized({ auth }) {
            return !!auth
        },
```

```
        async jwt({ token, profile, account }) {
            if (account && account.access_token) {
                token.accessToken = account.access_token
            }
            if (profile) {
                token.username = profile.username
            }
            return token;
        },
        async session({ session, token }) {
            if (token) {
                session.user.username = token.username;
                session.accessToken = token.accessToken;
            }
            return session;
        }
    }
})
```

And the docker-compose config for the web-app is:

```
    environment:
      - AUTH_SECRET="7vgUxWjehgeKTOFH2dZu0zSeKP61o9gl0b1vuHCqeMo="
      - AUTH_URL=https://app.carsties.local
      - AUTH_URL_INTERNAL=http://web-app:3000
      - API_URL=http://gateway-svc/
      - ID_URL=https://id.carsties.local
      - ID_URL_INTERNAL=http://identity-svc
      - NOTIFY_URL=https://api.carsties.local/notifications
      - VIRTUAL_HOST=app.carsties.local
      - VIRTUAL_PORT=3000
```

Another change in this section is the usage of the NOTIFY_URL instead of NEXT_PUBLIC_NOTIFY_URL. Instead of hardcoding this value we obtain it in a server component (layout.tsx) and then pass the property down to the SignalR Provider. This means we can easily update the value in config rather than needing to update a hardcoded value in the code which requires rebuilding the web app docker image.

```
// web-app/app/layout.tsx

export default async function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  const user = await getCurrentUser();
  const notifyUrl = process.env.NOTIFY_URL;

  return (
    <html lang="en">
```

The final docker-compose file for this section is now:

```
services:
  postgres:
    image: postgres
    environment:
      - POSTGRES_PASSWORD=postgrespw
    ports:
      - 5432:5432
    volumes:
```

```
      - /var/lib/postgresql/data
  mongodb:
    image: mongo
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=mongopw
    ports:
      - 27017:27017
    volumes:
      - /data/db
  rabbitmq:
    image: rabbitmq:3-management-alpine
    container_name: rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
  auction-svc:
    image: trycatchlearn/auction-svc:latest
    build:
      context: .
      dockerfile: src/AuctionService/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      - ASPNETCORE_URLS=http://+:7777
      - RabbitMQ__Host=rabbitmq
      - ConnectionStrings__DefaultConnection=Server=postgres;User Id=postgres;Password=po
      - IdentityServiceUrl=http://identity-svc
      - Kestrel__Endpoints__Grpc__Protocols=Http2
      - Kestrel__Endpoints__Grpc__Url=http://+:7777
      - Kestrel__Endpoints__WebApi__Protocols=Http1
      - Kestrel__Endpoints__WebApi__Url=http://+:80
    ports:
      - 7001:80
      - 7777:7777
    depends_on:
      - postgres
      - rabbitmq
  search-svc:
    image: trycatchlearn/search-svc:latest
    build:
      context: .
      dockerfile: src/SearchService/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      - RabbitMQ__Host=rabbitmq
      - ConnectionStrings__MongoDbConnection=mongodb://root:mongopw@mongodb
      - AuctionServiceUrl=http://auction-svc
    ports:
      - 7002:80
    depends_on:
      - mongodb
      - rabbitmq
  identity-svc:
    image: trycatchlearn/identity-svc:latest
    build:
      context: .
      dockerfile: src/IdentityService/Dockerfile
    environment:
```

```
      - ASPNETCORE_ENVIRONMENT=Docker
      - ASPNETCORE_URLS=http://+:80
      - IssuerUri=https://id.carsties.local
      - ClientApp=https://app.carsties.local
      - ConnectionStrings__DefaultConnection=Server=postgres; User Id=postgres; Password=
      - VIRTUAL_HOST=id.carsties.local
    depends_on:
      - postgres
  gateway-svc:
    image: trycatchlearn/gateway-svc:latest
    build:
      context: .
      dockerfile: src/GatewayService/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Docker
      - ASPNETCORE_URLS=http://+:80
      - ClientApp=https://app.carsties.local
      - VIRTUAL_HOST=api.carsties.local
  bid-svc:
    image: trycatchlearn/bid-svc:latest
    build:
      context: .
      dockerfile: src/BiddingService/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      - RabbitMq__Host=rabbitmq
      - ConnectionStrings__BidDbConnection=mongodb://root:mongopw@mongodb
      - IdentityServiceUrl=http://identity-svc
      - GrpcAuction=http://auction-svc:7777
    ports:
      - 7003:80
    depends_on:
      - mongodb
      - rabbitmq
  notify-svc:
    image: trycatchlearn/notify-svc:latest
    build:
      context: .
      dockerfile: src/NotificationService/Dockerfile
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:80
      - RabbitMq__Host=rabbitmq
    ports:
      - 7004:80
    depends_on:
      - rabbitmq
  web-app:
    image: trycatchlearn/web-app
    build:
      context: .
      dockerfile: frontend/web-app/Dockerfile
    volumes:
      - /var/lib/web/data
    environment:
      - AUTH_SECRET="7vgUxWjehgeKTOFH2dZu0zSeKP61o9gl0b1vuHCqeMo="
      - AUTH_URL=https://app.carsties.local
      - AUTH_URL_INTERNAL=http://web-app:3000
      - API_URL=http://gateway-svc/
```

```
      - ID_URL=https://id.carsties.local
      - ID_URL_INTERNAL=http://identity-svc
      - NOTIFY_URL=https://api.carsties.local/notifications
      - VIRTUAL_HOST=app.carsties.local
      - VIRTUAL_PORT=3000
  nginx-proxy:
    image: nginxproxy/nginx-proxy
    container_name: nginx-proxy
    ports:
      - 80:80
      - 443:443
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
      - ./devcerts:/etc/nginx/certs
```

Section 15 - Publishing locally to Kubernetes (original Appendix B moved into Legacy Appendix B)

This section was re-recorded fully as there is no need to create an external Identity Server to get this to work with the changes implemented in Section 14. The gist of the deployment to kubernetes is the same but we create a kubernetes manifest for the identity-svc so we can run everything in the kubernetes Docker desktop cluster.

Section 16 - Publishing to a hosted Kubernetes cluster (original Appendix C moved into Appendix C)

As the identity-svc is part of the kubernetes cluster this section was also completely re-recorded to accommodate this. It makes for a significantly easier deployment.