



UNIVERSITY OF
LEICESTER

**Department of Informatics
University of Leicester**

**Environmental Energy Quotes
Application**

by

Akhil Nuthakki

Project Supervisor: Prof. Dr Reiko Heckel

Project Supervisor: Dr Neslihan Suzen

environmentalenergies

Contents

1. Introduction	3
1.1 Aims and Objectives.....	3
1.2 Challenges.....	3
2. Requirements	4
2.1 Essential Requirements	4
2.2 Recommended Requirements	4
2.3 Optional Requirements	4
2.4 Use Case Diagram	5
2.5 Activity Diagram.....	6
2.6 Sequence Diagram.....	7
3. Background Research.....	8
3.1 Energy Domain and Quotation process	8
3.2 ASP.NET Core MVC Framework.....	11
3.3 Entity Framework Core	12
3.4 Word Solutions	16
3.5 Conclusion	17
4. Technical Specification	18
5. System Design	18
5.1 Application Architecture	18
5.2 Model Design	19
5.3 GUI Design	27
6. Implementation	33
7. Future Work	47
8. References	48

1. Introduction

1.1 Aims and Objectives

Environmental Energies deliver environmental energy solutions targeting residential homeowners, commercial companies, manufacturing, distribution, and agricultural Industries. The company processes for quoting and contracting for new work is time consuming due to the variety and number of bespoke quotes they need to provide to each homeowner. Their productivity would greatly be enhanced with process innovation with the introduction of a more automated Quoting/Contracting package. The existing process involves manually searching and updating the required mapping data in a excel spreadsheet along with other input parameters to create cost models using excel-formulas. Once the customers are happy with the generated cost models, employees are manually updating the required fields in a formatted word template and converts it into a pdf before sending an email to the customer attached. The current process takes up most of the effort in the repeated tasks of generating quote every time they have a new customer, and they didn't have any proper data structure and storage for the quotes to re-use the data for analytics and improvement of business.

So, the objective of this project is to develop a web-based application that speed up and make the process more cost effective for company to enable their small team to quote for work quickly and more effectively.

This web-based application must be developed for the employees of environmental energies to calculate cost models and generate quotes based on the given input parameters and necessary mapping data that are currently existing in spreadsheets. An algorithm must be developed to calculate both financed and un-financed cost models using provided input parameters and existing mapping data. After the calculation of cost model's user is provided with the options to download quote and email quote to customer which is formatted in word document based on calculations and other input parameters which is then converted to a pdf before attaching the file to email to the customer.

1.2 Challenges

The two main challenges for this development of the application are understanding of the energy domain-oriented terms and working with the documents to generate quote. The first challenge is to understand the current existing process and energy domain-oriented terms to design the scalable, normalized data structure and relations between the data tables based on domain-oriented terms. It also reflects in the implementation of relationships using the entity framework core. The second challenge is working on word solutions. The factors that need to be considered while working with the word solutions are file handling, automation time, the amount of space each document takes up in the machine as the current format of the document holds at approximately 6,043 KB, the IT resources required for the concurrent word automations. So, while generating the word document we need to consider all the above factors to provide a seamless experience to the application user and need to keep an eye on the cost of the required infrastructure to pull the word automation and storage of the generated quote.

2. Requirements

2.1 Essential Requirements

1. User Authentication with an email and password.
2. Consolidate the existing spreadsheets and define scenarios of data entry, calculation, and quote generation.
3. Design the normalized database including the mapping tables (Postcode-Region & Irradiance-Datasets), Quotes, customer details reducing redundancy.
4. Calculation of un-financed and financed cost models based on the input investment parameters.
5. Generation of quotes in a document of pdf format addressing the customer based on the relevant standards and legislation.
6. An Automatic formatted email attached with the generated energy quote sent to the customer email address from the organization group email box.

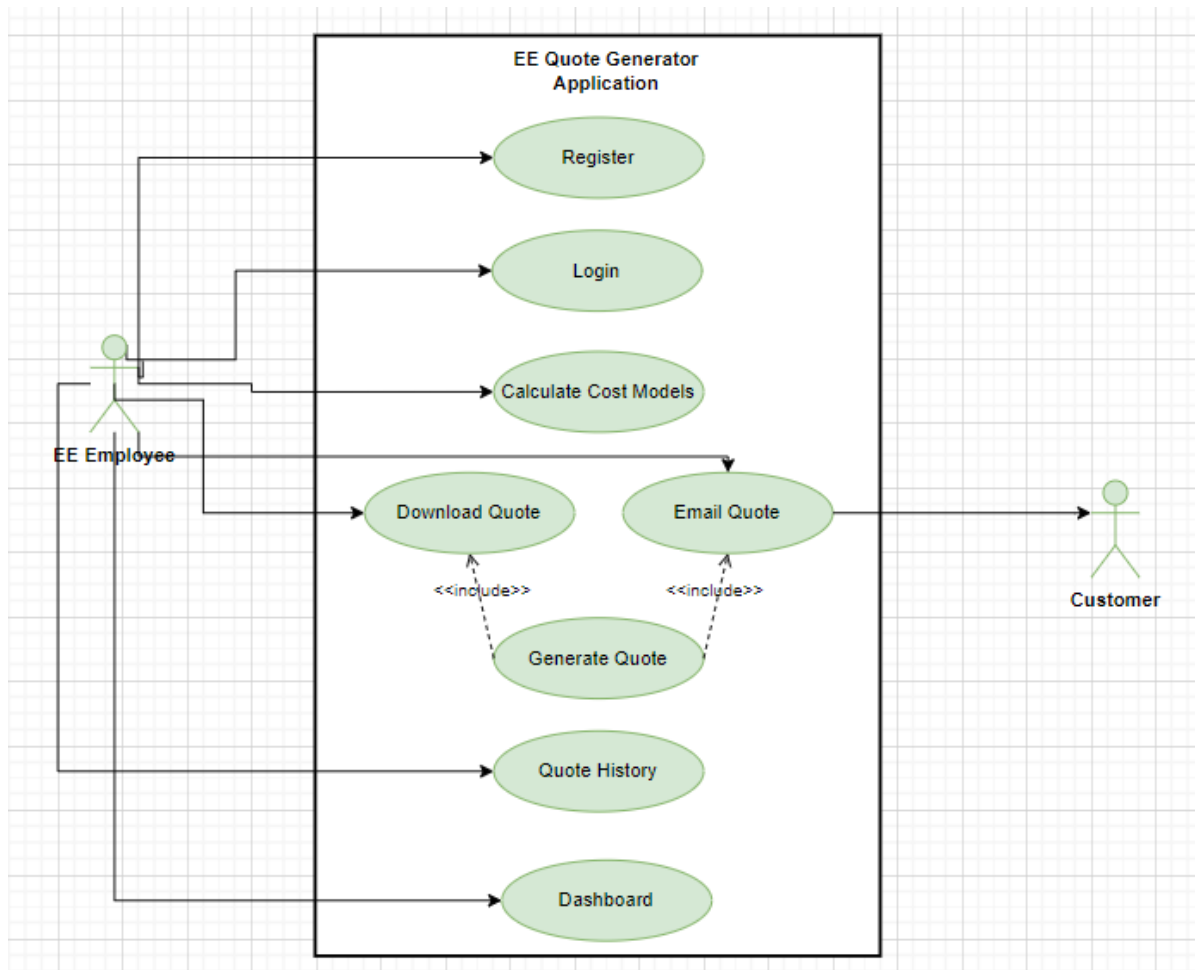
2.2 Recommended Requirements

1. Responsive web pages that can be accessible across all the devices such as mobile, tablet and a desktop.
2. Recent History of quotes to look-up to the past generated quote details and to retrieve customer details to make them a new offer.
3. Download the generated quote in the original format.

2.3 Optional Requirements

1. Dashboard to provide the overview of the quotes generated such as count by the region, by the cost, and various other factors.
2. Admin access to control the user management and to upload the existing mapping data.
3. Comparison of the multiple input parameters for a site and their cost models to make an effective decision.

2.4 Use Case Diagram



Above is the Use Case diagram that shows all the use cases of the EE Quote Generator Application. Please find the detailed description of the Use cases and Actors of the application below:

Actors:

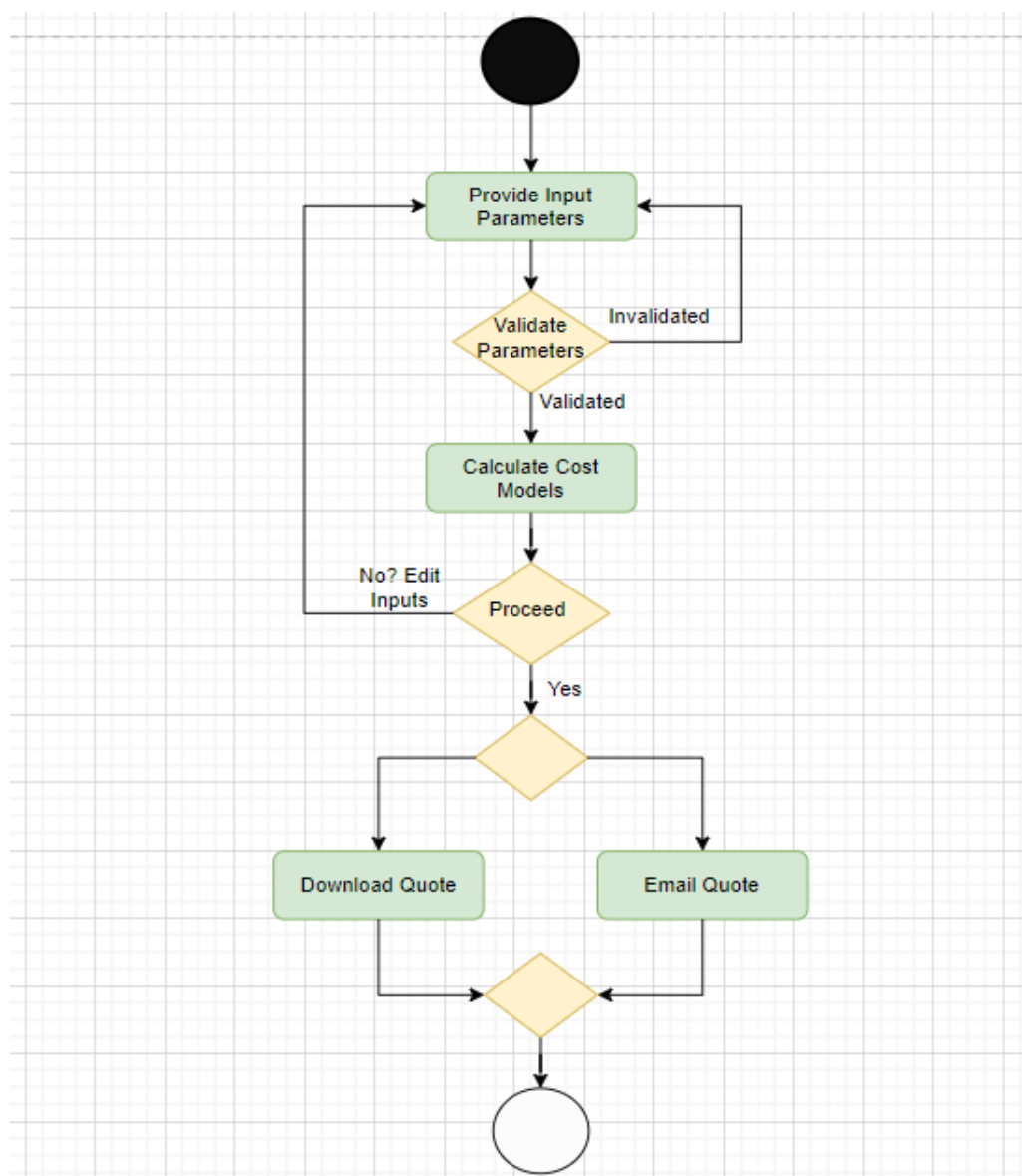
1. *EE Employee* – Employee of Environmental Energies acts a primary actor also classified as active actor for this application calls on the application to deliver its quote generation services.
2. *Customer* – Customer is recipient of the services of this application acts a secondary actor also called as passive actor for this application.

Use Cases:

1. *Login* – User (EE Employee) interacts with this use case. User provides their login credentials such as username and password to complete user authentication that verifies the identity of the user attempting to access the application.
2. *Register* – User register for the access of this application by providing their details and setting up their username and password.
3. *Calculate Cost Model* – User provides the input parameters that are required to calculate the cost model and after successful validation of the inputs the application will calculate cost model for the given inputs.

4. *Generate Quote* – This service is implemented when the user accesses the Download Quote and Email Quote functionalities.
5. *Download Quote* – User could be able to download the generated quote in the original word document format.
6. *Email Quote* – User could be able to directly send the generated quote document attached in an email to the customer.
7. *Quote History* – User access the history of quotes generated and can re-quote the work for the customer with some new offer.
8. *Dashboard* – Dashboard provides the analytical insights of the history of quotes generated so far. It will provide the summary of total quotes generated.

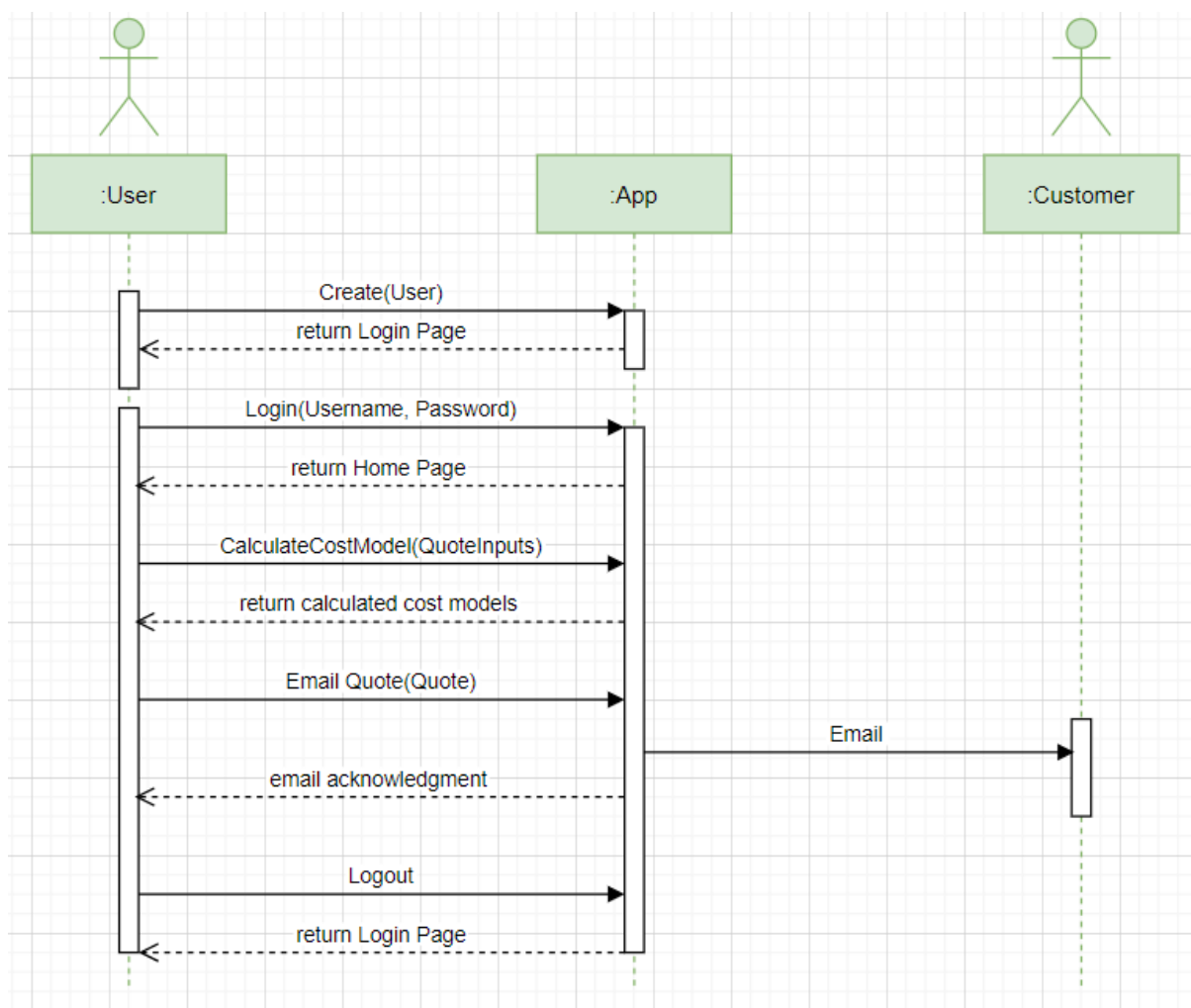
2.5 Activity Diagram



Activity Diagram

Activity Diagram shown above describes the flow of control of generate quote process in the EE Quote Generator Application. Initially, User provides the investment parameters and site details to calculate the cost models. The inputs will be validated before proceeding with the calculations. If the inputs are validated, the cost projections and quote summary will be calculated and presented back to the user. If the inputs are not valid, application will throw a validation error asking user to provide the valid inputs. Once, the cost projections are presented, user can proceed with either of the action to download the quote or email the quote to the customer.

2.6 Sequence Diagram



Sequence diagram shows the sequence of steps that happens during a quote generation process. User will create an account to access the application. After creation, user logs into the application using registered username and password. Once Logged In, user provides the quote inputs such as site details, investment parameters and other metrics to calculate the cost model and email the quote to customer with the help of calculated cost models.

3. Background Research

3.1 Energy Domain and Quotation process

Basic understanding of the energy domain-oriented terms is a must require criteria to improve the process of automation by designing complete data structure and developing a web application. Environmental Energies Ltd has provided the existing excel spreadsheets of the mapping data such as Postcode Region Mapping, Irradiance Mapping, and most importantly the Investment appraisal form to calculate the cost models. Firstly, we must understand each one of the investment input parameters of the appraisal form to calculate both financed and un-financed cost models which helps us to design the data structure and the relations of the data tables to reduce redundancy.

Solar Electricity (PV) Investment Appraisal

Site details:

18 Bushmead AvenueMK40 3QL

Assumptions

Size of system3.96

Cost of System£3,400

Annual Generation (MCS Standard)880

Azimuth Of0°

Roof Pitch °35

MakeTBC

Watts330

20 Year Profit

Yield

IRR

20 year Co2 savings

UN-FINANCED

£24,950

26.6%

36

Tonnes

FINANCED

£24,851

25.89%

30.0%

Year 1 Co2 savings

2

Tonnes

PV COST

Additional Costs

Additional Costs

TOTAL COST

£ 3,400

£ -

£ 3,400

UN-FINANCED MODEL

Year

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Current Cost per kWh

0.260

0.272

0.285

0.297

0.308

0.319

0.330

0.341

0.352

0.363

0.374

0.385

0.396

0.407

0.418

0.429

0.440

0.451

0.462

0.473

0.484

Output kWh

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

3,467

Capital Outlay

£3,400

Insurance

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

Maintenance

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

Monitoring only

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

TOTAL ANNUAL COST

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

Annual Revenue

£908

£947

£989

£1,033

£1,079

£1,129

£1,179

£1,231

£1,286

£1,344

£1,404

£1,467

£1,532

£1,601

£1,672

£1,747

£1,825

£1,907

£1,992

£2,080

£2,171

Revenue from Export

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

£0

Total Revenue

£908

£947

£989

£1,033

£1,079

£1,129

£1,179

£1,231

£1,286

£1,344

£1,404

£1,467

£1,532

£1,601

£1,672

£1,747

£1,825

£1,907

£1,992

£2,080

£2,171

Cash Flow (£)

-£3,400

£908

£947

£989

£1,033

£1,079

£1,129

£1,179

£1,231

£1,344

£1,404

£1,467

£1,532

£1,601

£1,672

£1,747

£1,825

£1,907

£1,992

£2,080

£2,171

Balance, End of Period

£568

£1,515

£2,504

£3,537

£4,616

£5,741

£6,912

£8,129

£9,393

£10,704

£12,064

£13,473

£14,932

£16,441

£18,000

£19,609

£21,268

£22,977

£24,736

£26,545

environmentalenergies

சூரிய மின்சாரம்

சூரிய மின்சாரம்

The above is the screenshot of the investment appraisal excel spreadsheet to calculate both un-financed and financed **cost models** and the **summary** (highlighted with **border** in the above screen shot). Employee of EE provides the **site information** and other **investment & input parameters** on the top left division of the spreadsheet.

The Investment parameters includes No of Panels, Panel Watts, Panel Make and Cost of the system is the information of investment in the energy system by the customer. The Input parameters includes details such as Exportation of Electricity, Current cost of Electricity, Retail Price Inflation as it effects all the calculations over the 20-year cost model, Finance details, and Annual maintenance package details.

Here comes the most important details of the Investment appraisal that are required to carry out the calculation of the cost models. The **site details** provided in the form includes the Address of the site, Inclination of the site, Orientation of the site and Annual Generation Value (MCS Standard) which is the Irradiance value of the site based on zone location, Inclination and Orientation after taking shade factor into account. Shade factor is the parameter that suggest the amount of shading that occurs to the system in the site due to the external factors.

Annual Generation Value (MCS Standard) is calculated manually by the user referring to the Postcode Region Mapping and Irradiance Mapping spreadsheets which are shown below:

Postcode	Region	Postcode	Region	Postcode	Region	Postcode	Region
AB	16	G	14	N	1	SK	7E
AL	1	GL	5E	NE	9E	SK13	6
B	6	GU	1	NG	11	SK17	6
BA	5E	GU11-12	3	NN	6	SK22-23	6
BB	7E	GU14	3	NP	5W	SL	1
BD	11	GU28-29	2	NPS	13	SM	1
BD23-24	10	GU30-35	3	NR	12	SN	5E
BH	3	GU46	3	NW	1	SN7	1
BL	7E	GU51-52	3	OL	7E	SO	3
BN	2	HA	1	OX	1	SP	5E
BR	2	HD	11	PA	14	SP6-11	3
BS	5E	HP	10	PE	12	SR	9E
BT	21	HG	1	PE9-12	11	SR7-8	10
CA	8E	HR	6	PE20-25	11	SS	12
CB	12	HS	18	PH	15	ST	6
CF	5W	HU	11	PH19-25	17	SW	1
CH	7E	HX	11	PH26	16	SY	6
CH5-8	7W	IG	12	PH30-44	17	SY14	7E
CM	12	IP	12	PH49	14	SY15-25	13
CM21-23	1	IV	17	PH50	14	TA	5E
CO	12	IV30-32	16	PL	4	TD	9S
CR	1	IV36	16	PO	3	TD12	9E
CT	2	KA	14	PO18-22	2	TD15	9E
CV	6	KT	1	PR	7E	TF	6
CW	7E	KW	17	RG	1	TN	2
DA	2	KW15-17	19	RG21-29	3	TQ	4
DD	15	KY	15	RH	1	TR	4
DE	6	L	7E	RH1-20	2	TS	10
DG	8S	LA	7E	RH77	2	TW	1
DH	10	LA7-23	8E	RM	12	UB	1
DH4-5	9E	LD	13	S	11	W	1
DL	10	LE	6	S18	6	WA	7E
DN	11	LL	7W	S32-33	6	WC	1
DT	3	LL23-27	13	S40-45	6	WD	1
DY	6	LL30-78	13	S49	6	WF	11
E	1	LN	11	SA	5W	WN	7E
EC	1	LS	11	SA14-20	13	WR	6
EH	15	LS24	10	SA31-48	13	WS	6
EH43-46	9S	LU	1	SA61-73	13	WV	6
EN	1	M	7E	SE	1	YO	10
EN9	12	ME	2	SG	1	YO15-16	11
EX	4	MK	1			YO25	11
FK	14	ML	14			ZE	20
FY	7E						

Above Diagram shows post code and region mapping.

Orientation (variation from south)																																						
Slope	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175		
0	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307	307
1	316	316	316	316	315	315	314	314	314	313	312	311	310	309	308	307	307	306	305	304	303	302	302	301	300	300	300	300	300	300	300	300	300	300	300	300	300	300
2	324	324	324	324	323	323	322	322	321	320	319	318	317	316	315	314	313	312	311	310	309	308	307	306	305	304	303	302	301	300	300	300	300	300	300	300	300	300
3	333	333	333	332	331	330	329	328	327	325	323	322	320	318	315	311	309	306	304	302	899	897	895	893	891	889	887	885	884	883	882	881	880	879	878	877	876	875
4	341	341	341	340	339	338	337	335	333	331	329	328	324	321	318	315	312	309	306	303	300	897	894	891	888	885	883	880	878	876	874	873	872	871	870	869	868	867
5	349	349	349	348	347	345	344	342	339	337	334	331	328	324	321	318	315	312	309	306	303	300	897	894	891	888	885	883	880	878	876	874	873	872	871	870	869	868
6	357	357	357	356	354	353	350	348	345	342	339	335	331	327	323	319	314	310	305	300	896	891	887	882	878	874	870	866	863	860	857	855	853	852	851	850	849	
7	365	365	364	362	360	357	354	351	348	344	340	336	332	328	325	321	316	311	306	300	895	890	885	879	873	866	863	859	855	852	849	846	844	842	841	840	839	
8	373	372	372	370	369	366	364	361	357	353	349	344	339	333	328	322	316	310	304	897	891	885	879	873	867	862	857	852	848	844	840	837	835	833	831	830	829	
9	380	380	379	377	376	373	370	368	362	358	353	348	342	336	330	323	317	310	303	899	892	885	879	873	867	862	856	850	845	840	835	831	829	825	823	821	820	
10	387	387	386	384	382	379	376	372	368	363	357	352	346	339	332	326	319	312	305	894	887	879	871	864	856	848	839	830	821	812	803	794	785	776	767	758	750	
11	394	394	393	391	389	386	382	378	373	368	362	356	349	341	334	326	318	310	301	893	884	876	868	859	852	844	837	830	824	818	814	809	806	803	801	800	799	
12	1001	1000	999	998	995	992	989	985	979	972	966	959	952	944	936	928	919	909	899	889	880	870	860	850	841	832	824	816	808	802	796	791	787	784	781	780	779	
13	1007	1007	1006	1004	1001	998	993	989	983	977	970	962	955	946	938	929	919	909	899	889	880	870	860	850	841	832	824	816	808	802	796	791	787	784	781	780	779	
14	1014	1014	1013	1012	1010	1007	1003	999	991	974	958	937	919	897	874	851	828	805	782	759	736	713	690	667	644	621	600	578	555	532	509	486	463	440	417	394	371	
15	1020	1019	1018	1016	1013	1009	1004	998	992	985	977	969	960	951	941	930	919	908	896	884	872	860	848	836	825	814	803	793	783	773	763	753	743	733	723	713	703	
16	1025	1025	1024	1021	1018	1014	1009	1003	996	989	981	972	963	953	942	930	918	906	894	882	870	857	844	832	819	806	793	779	765	751	737	723	709	695	681	667	653	
17	1031	1031	1029	1027	1023	1019	1014	1007	1001	993	984	975	965	955	943	931	919	907	895	883	871	858	845	832	819	806	793	779	765	751	737	723	709	695	681	667	653	639
18	1036	1036	1034	1032	1028	1024	1019	1012	1004	996	988	979	967	956	945	933	920	907	894	880	867	853	840	827	814	801	789	769	750	732	715	700	686	672	658	644	630	
19	1041	1041	1039	1037	1033	1028	1023	1016	1008	1000	990	980	969	958	946	933	920	906	893	879	864	850	836	822	809	795	781	767	753	739	725	711	697	683	669	655	641	
20	1046	1046	1044	1041	1038	1033	1027	1020	1012	1003	993	983	971	959	947	934	921	907	893	878	862	847	832	816	803	789	777	764	753	743	735	726	717	714	712	710	708	
21	1051	1050	1049	1046	1042	1037	1031	1023	1015	1006	996	985	973	961	949	936	922	908	894	878	862	846	830	813	798	784	770	757	745	735	726	717	713	709	705	701		
22	1055	1055	1053	1050	1046	1041	1034	1027	1019	1009	998	987	975	962	949	936	922	908	894	878	862	846	830	813	798	784	770	757	745	735	726	717	713	709	705	701	697	
23	1060	1059	1057	1054	1050	1044	1038	1030	1021	1012	1001	989	976	963	949	936	922	908	894	878	862	846	830	813	798	784	770	757	745	735	726	717	713	709	705	701	697	
24	1064	1063	1061	1058	1054	1048	1041	1032	1024	1014	1003	991	978	964	949	936	922	908	894	878	862	846	830	813	798	784	770	757	745	735	726	717	713	709	705	701	697	
25	1067	1067	1065	1062	1057	1051	1044	1036	1027	1018	1005	992	979	965	949	934	918	901	883	866	848	830	812	794	777	760	744	729	715	703	692	683	676	670	666	664		
26	1071	1070	1068	1065	1060	1054	1047	1039	1029	1019	1007	994	980	965	950	934	917	900	882	864	845	827	808	790	772	754	738	722	707	695	684	675	667	661	657	654	651	
27	1074	1073	1071	1068	1063	1057	1050	1041	1031	1020	1008	995	981	965	950	934	917	900	880	861	842	823	804	785	767	749	731	715	700	687	676	666	658	652	647	645	643	
28	1077	1076	1074	1071	1066	1060	1052	1043	1033	1022	1009	996	981	966	950	933	915	897	878	859	840	820	800	781	762	743	725	708	693	679	667	657	649	643	638	635	633	
29	1080	1079	1077	1073	1068	1062	1054	1045	1032	1023	1011	997	982	968	949	931	913	894	874	854	833	813	792	772	751	731	713	694	678	663	650	640	631	624	619	616		
30	1082	1081	1079	1076	1071	1064	1056	1047	1036	1025	1011	997	982	968	949	931	913	894	874	854	833	813	792	772	751	731	713	694	678	663	650	640	631	624	619	616		
31	1084	1084	1081	1078	1073	1066	1058	1048	1036	1025	1012	998	982	966	948	929	912	892	872	851	831	809	788	767	746	726	706	688	671	655	642	631	622	616	609	606		
32	1087	1086	1083	1079	1074	1067	1059	1049	1037	1025	1013	999	983	966	948	929	912	892	872	851	831	809	788	767	746	726	706	688	671	655	642	631	622	616	609	606		
33	1090	1089	1086	1082	1077	1070	1062	1052	1040	1028	1016	1002	986	968	949	930	912	892	872	851	831	809	788	767	746	726	706	688	671	655	642	631	622	616	609	606		
34	1093	1092	1089	1085	1079	1072	1064	1054	1042	1030	1018	1004	988	969	950	931	912	892	872	851	831	809	788	767	746	726	706	688	671	655	642	631	622	616	609	606		
35	1096	1095	1092	1088	1082	1075	1067	1056	1044	1032	1020	1006	990	971	952	933	914	894	873	852	832	810	789	768	747	727	707	689	672	656	643	632	623	617	611	605		
36	1100	1099	1096	1092	1086	1079	1071	1060	1048	1036	1024	1010	994	975	956	937	918	898	877	856	835	814	793	772	752	732	712	694	677	661	648	637	628	622	616	610		
37	1103	1103	1100	1096	1090	1083	1075	1064	1052	1040	1028	1014	1000	984	965	946	927	908	887	866	845	824	803	782	762	742	722	702	684	667	651	638	627	620	614	608		
38	1106	1106	1103	1100	1095	1089	1082	1074	1063	1051	1039	1026	1012	996	977	958	939	919	898	877	856	835	814															

User then use the calculated information in the Investment Appraisal Excel sheet to generate the quote of the formatted document.

2022/([Job Number])/S/IC

Environmental Energies Limited
Harborough Innovation Centre
Airfield Business Park, Market Harborough,
Leics, LE16 7WB
Tel: 01858 525 407
Email: sales@environmentalenergies.co.uk
www.environmentalenergies.co.uk



Combined quote-contract V8 2021

(Today's Date)

Customer Name: (MASTER NAME)
Quote Reference: 2022/([MASTER JOB NUMBER])/([MASTER PROJECT TYPE])/([MASTER PROJECT LEADER INITIALS])
Site Details: (SITE DETAILS)
Install Size: (INSTALL SIZE)
MPAN: (MASTER MPAN)

Dear (NAME),

Many Thanks for your enquiry into the potential install of solar PV (Site Details) please find below our quotation/contract following the site visit.

We are highly accredited in recognition for our track record of designing and delivering high quality renewable solutions and are a certified **Which? Trusted Trader**, as well as being approved by the **Carbon Trust**, and **NFU Farm Energy**, all which command extremely high standards of both workmanship and ethical practices. We are also an **MCS** approved installer, **CHAS & Construction Line Gold** accredited as well as being members of **RECC** and **TrustMark** registered, ensuring best practice to all our clients. **TrustMark** is a **Government** endorsed quality scheme which covers work for consumers in and around their home, offering you peace of mind that Environmental Energies has been thoroughly vetted to meet required standards and is committed to good customer service, technical competence and trading practices.

Our approach is very straightforward, we manage the complete installation process, from design supply and installation to commissioning of the system, ensuring we deliver a fully comprehensive turn-key solar PV solution

1. QUOTATION SUMMARY

Further to our discussions, I can confirm that by utilising the suitable space available and working in line with your requirements and the roof space available we would look to install a (INSTALL SIZE) Roof Mounted Solar PV array made up of (NO. PANELS) X (PANELS WATTS)_w and Solar Panels.

I have assumed the property will use 100% of the power generated given the conversation we have had and predicted energy storage. The figures shown below are based on this information,

Costs & Returns

Cost of PV System (exc VAT)	YIELD	EST. 20 YEAR PROFIT
£(COST OF SYSTEM)	(YIELD)%	£(20-YEAR PROFIT)
kWp Installed	Panels Required	Year 1 kWh Generated
(SYSTEM SIZE)	(NO. PANELS)	(OUTPUT KWH)
20 YEAR Co2 SAVINGS (Tonnes)	CO ² per Year (Tonnes)	Payback Years
18t	1t	(PAYBACK)

* VAT EXC. at 20% (if you are over 60 VAT will be charged at 5%)

Above Diagram shows quote format.

Along with the calculated cost models and summary and other investment input parameters user updates the additional components that are required as part of the quotation /contract package.

2. INSTALL DETAILS

Site: (SITE DETAILS)
System: (SYSTEM SIZE)
Components:

Item	Quantity	
(Wattage Of Panel) Solar PV	(NO. PANELS)	Yes
Inverter	1	Yes
Roof Mounting System	1	Yes
Optimisers – Solar Edge		No
Lifting Equipment, Scaffolding and Netting as required		Yes
Waste Removal		Yes
DC and AC Isolators		Yes
DC and AC Wiring		Yes
Connection to the distribution board		Yes
Approved electrical generation meter	1	Yes
Batteries Fox ESS 5.2Kw		No
EV Chargers – 7Kw Zappi		No
Immersion Controller – Solic 200		No
Pigeon proofing		No
System Installation, commission and warranties		Yes
MCS registration		Yes
Structural Report		No
Monitoring Equipment – Customer Use Only – Via Inverter – Comms Cable by others		Yes
DNO Documents	TBC	Yes
Proposed Commencement Date	TBC	

Once they have completed the updating the calculations and information in the quote format, they will manually mail the quote template to the customer.

3.2 ASP.NET Core MVC Framework

ASP.NET Core MVC Framework is a lightweight, open-source framework that integrates easily with the core aspects of the ASP.NET.

Architectural Pattern:

ASP.NET Core MVC Framework uses Model View Controller architectural pattern which categorizes the components into three major categories such as Models, Views, and Controllers. The Controller invokes the Model to retrieve data and executes the necessary business logic and then passes the Model data to the View which then returns to the requestor client.

Model – It represents a database table or a state of the application.

View – It is a visual representation of the model in the client browser

Controller – It is a communication between the Model and View.

Benefits of MVC Architectural pattern are:

1. The division of components into meaningful categories helps developers to easily manage large-scale applications.

2. Architectural pattern allows us to develop application faster than usual as division of work section-by-section allows large number of people to work on the development simultaneously.
3. It simplifies the Test-Driven Development and makes it easier to test the individual components as the developers go by unit testing after development of each functionality among the model, view, and controller.

Routing:

Conventional Routing is one of the routing techniques used in the ASP.NET Core MVC. Conventions will be configured to match the incoming URL endpoints as given below. Controller actions will be these endpoints in ASP.NET MVC.

```

}
}
}
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=User}/{action=Login}/{id?}");
});

```

Model Binding:

The data that comes from HTTP Requests is used among Controllers and Razor pages. The model binding system retrieves data from form fields, route data and query string and provides the data to controllers to razor pages by converting string data to .NET types and updates the properties of complex types.

Razor View Engine:

Razor View Engine is a server-side markup syntax which helps us to write server-side code and HTML in web pages using C#. The syntax must be enclosed with @{...} and ending statements with “;”. These Razor files will have extension of .cshtml.

```

@{
    string message = "foreignObject example with Scalable Vector Graphics (SVG)";
}

<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
    <rect x="0" y="0" rx="10" ry="10" width="200" height="200" stroke="black"
        fill="none" />
    <foreignObject x="20" y="20" width="160" height="160">
        <p>@message</p>
    </foreignObject>
</svg>

```

3.3 Entity Framework Core

Entity Framework Core is a light weight, open-source version of Entity Framework data access technology. Entity Framework eliminates the need of the data access code and enables developers to work with a database using .NET Objects. It is used for Object Relational Mapping (ORM). ORM is a programming technique for converting data between relational databases and .NET programming language. It sits between the .NET Objects and database server.

Creation and Configuration of a Model:

EF Supports different model development techniques as given below:

1. Generating a model from an existing database.
2. Matching a model with an existing database.
3. Creating a database from a model by EF Code first Migrations.

This project uses the EF Code first Migrations technique to create a database from a model and follow any changes made through the model.

Configuring a model can be achieved using Data Annotations and Fluent API.

For most of the properties, data annotations are used to configure a model.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.DataAnnotations.Annotations;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
}

[Table("Blogs")]
public class Blog
{
    public int BlogId { get; set; }

    [Required]
    public string Url { get; set; }
}
```

Complex properties of a model can be configured using Fluent API by overriding *OnModelCreating* method in the derived context from the *DbContext*.

```
using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.FluentAPI.Required;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    #region Required
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
    #endregion
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Query Data:

Language Integrated Query (LINQ) is used to query data from the database in Entity Framework Core. LINQ provides the way of writing typed queries using any .NET Language.

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Save Data:

Instances of the Entities are responsible for the data being created, updated, and deleted in the database. Each instance of the entity class that relates to the DbContext will have *State* and *ChangeTracker* to track the changes made to the entity class or to determine if it is a new instance. These changes are recorded in the ChangeTracker and used to INSERT, UPDATE, DELETE while using *SaveChanges* on to the database.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Relationships:

A relationship between two entities is represented by a foreign key constraint in the Entity Framework Core. The relationships are mainly classified into four types:

1. One-to-One Relationship
2. One-to-Many Relationship
3. Many-to-One Relationship
4. Many-to-Many Relationship

The relationships that are used in this project so far based on the data requirements are One-to-One and One-to-Many.

One-to-One Relationship:

Foreign key is used to describe the relationship by storing the primary key values of the related entity in the dependent entity. It will have a reference navigation property on both sides of the relationship.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

One-to-Many Relationship:

One-to-Many Relationship entities follows the same convention as the one-to-one relationship, but additionally the dependent class will have its own primary key and the relatable class with appropriate data structure to inject the dependent class into the relatable class.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

Code First to a New Database:

Once the model is created, configured, or even updated later, we need to update these to the database schema. Code First Migrations feature will help us to create or update the database schema as per the creation or configuration of model in .NET project.

The Code First Migrations uses the following steps to perform the migrations to the database schema:

1. Go To Tools -> Library Package Manager -> Package Manager Console.
2. Run the Enable-Migrations command to enable the Code First Migrations feature in the project.
3. Run the Add-Migration {AddUrl} command in the Package Manager Console. It will create the Migrations folder in the project folder path if none exists. It will also create files in the Migrations folder named *DbContextModelSnapshot.cs* and *Timestamp_AddUrl.cs*.
4. *DbContextModelSnapshot.cs* contains the current snapshot of the database schema in the entities context.

5. *Timestamp_AddUrl.cs* contains *Up* and *Down* methods that is used to update the database schema based on the Model Builder.

```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

6. Run the Update-Database command to migrate the models to database schema. Update-Database schema uses the *DbContextModelSnapshot.cs* and recent *Timestamp_Addurl.cs* -> *Up* to create or update the schema.
7. Remove-Migration {AddUrl} command is used for the removal of the last migration changes which uses *Timestamp_Addurl.cs* -> *Down* to roll back the changes in the last migration and updates the *DbContextModelSnapshot.cs*. to the current snapshot of the schema.

3.4 Word Solutions

VSTO Add-ins for Microsoft Office Word are available in the Visual Studio project templates to create document-level customizations. Automation of word, Extension of word features, and Customization of word user interface can be achieved by using these solutions.

Automation of word:

Word Object Model provides many types such as classes and interfaces for the automation of word. Automation of word includes programmatical creation of tables, formatting documents, and update the text with use of Ranges and Paragraphs.

Word object model:

Development of word solutions in visual studio requires interaction with Word object model. Word object model consists of classes and interfaces which indeed are available in the Microsoft Interop Word assembly and are defined in the Microsoft.Office.Interop.Word namespace.

Following tasks are used to perform specific tasks on word using word object model:

Work with documents:

Document Object available inside the *Application* Object consists of *Open* method to open the existing Microsoft word documents. The method requires the fully specified file path of

the document. The method returns the document which is a presentation of the opened document.

```
this.Application.Documents.Open(@"C:\Test\NewDocument.docx");
```

Work with text in documents:

Find Object which is part of both *Selection* Object and *Range* Object can be used to search for the text in the word and replace the text with the specified text. *Execute* method of the find object with parameters that sets the rules of the execution can be used to update the text in the documents.

```
public bool Execute (ref object FindText, ref object MatchCase, ref object
MatchWholeWord, ref object MatchWildcards, ref object MatchSoundsLike, ref object
MatchAllWordForms, ref object Forward, ref object Wrap, ref object Format, ref object
ReplaceWith, ref object Replace, ref object MatchKashida, ref object MatchDiacritics, ref
object MatchAlefHamza, ref object MatchControl);

private void SearchReplace()
{
    Word.Find findObject = Application.Selection.Find;
    findObject.ClearFormatting();
    findObject.Text = "find me";
    findObject.Replacement.ClearFormatting();
    findObject.Replacement.Text = "Found";

    object replaceAll = Word.WdReplace.wdReplaceAll;
    findObject.Execute(ref missing, ref missing, ref missing, ref missing, ref missing
        ref missing, ref missing, ref missing, ref missing, ref missing,
        ref replaceAll, ref missing, ref missing, ref missing, ref missing);
}
```

Work with tables:

Each table consists of cells. Each *Cell* Object represents one cell in the table and can be referred by the row location and column location of the cell. Once *Cell* Object is referred to a cell by its location and add text to the cell.

```
Word.Cell cell = this.Tables[1].Cell(1, 1);

cell.Range.Text = "Name";
cell.Range.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphRight
```

3.5 Conclusion

.NET Framework provides the dynamic link libraries for the Microsoft word solutions as generating quote is one of the key requirements of the project. It also provides the Entity framework to map the objects to the data tables as relational database and helps the developers to work with a database using .NET Objects. So, it's highly reliable to use ASP.NET MVC Core framework with Entity Framework to develop this application.

4. Technical Specification

Programming Languages:

1. C# - 9.0
2. JavaScript - ES6
3. HTML - 5
4. CSS - 3
5. SQL

Frameworks:

1. ASP.NET Core MVC Framework
2. Entity Framework

Dependencies:

1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.SqlServer
3. Microsoft.EntityFrameworkCore.Tools
4. Microsoft.Office.Interop.Word

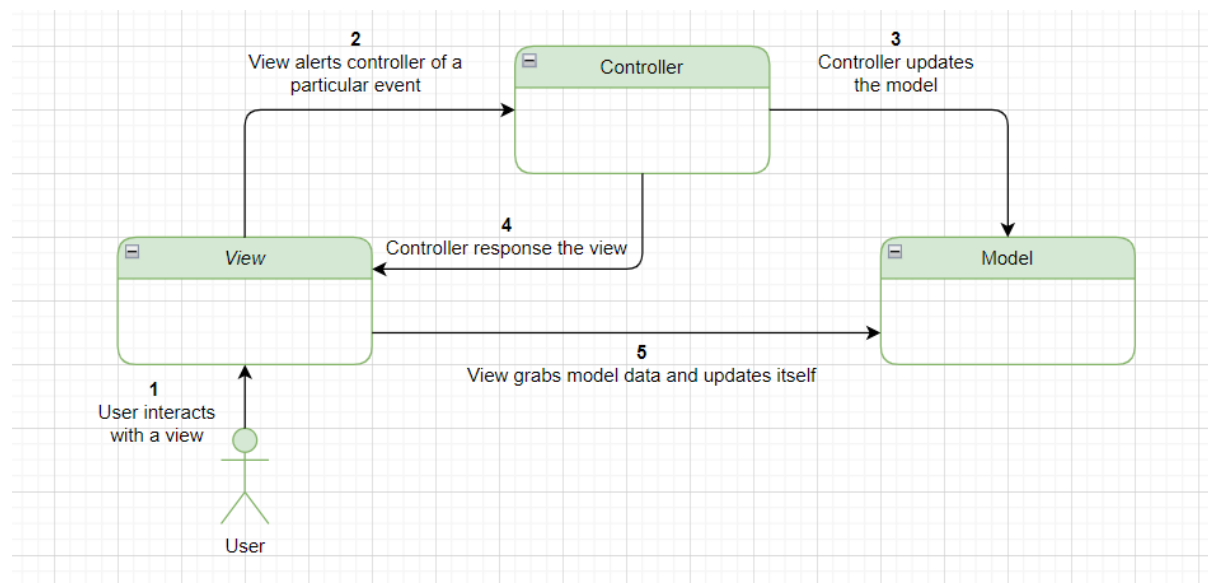
IDE:

1. Microsoft Visual Studio 2019
2. Microsoft SQL Server Management Studio 18

5. System Design

5.1 Application Architecture

EE Quote Generator Application uses MVC (Model View Controller) Architectural pattern. The below given figure illustrates the flow of requests from the user in MVC Architecture.



Models, Views, and Controllers are designed as per the requirements of the application. Detailed view of three major components is given below:

Controllers:

1. *User Controller* – It handles the client requests of user authentication and user registration to access the application.
2. *Generate Quote Controller* – It handles the client requests of calculation of cost models, download quotes, email quotes to customer, and view of quote details.
3. *History Controller* – It handles the requests of viewing the list of past generated quotes and dashboard of data.

The Implementation of each of the above given controller is discussed in detail in the below Implementation section.

Models:

1. *User* – It represents the data of user.
2. *Quote* – It represents the primary data belongs to quote entity.
3. *Customer* – It represents the data of quote customer.
4. *InvestmentParameters* – It represents the data of investment input parameters provided by the user.
5. *Components* – It represents components data of quoted work.
6. *CostSummary* – It represents cost projections summary of the quotation.
7. *CostProjections* – It represents the calculated cost projections of the provided input parameters.
8. *Irradiance* – It represents Irradiance mapping data.
9. *RegionMap* – It represents Region Postcode mapping data.

The creation of models, attributes of models, and relationships between models are discussed in detail in the below Model Design section.

Views:

Each controller has views for their respective actions and action results. Here is list of views that belongs to each controller.

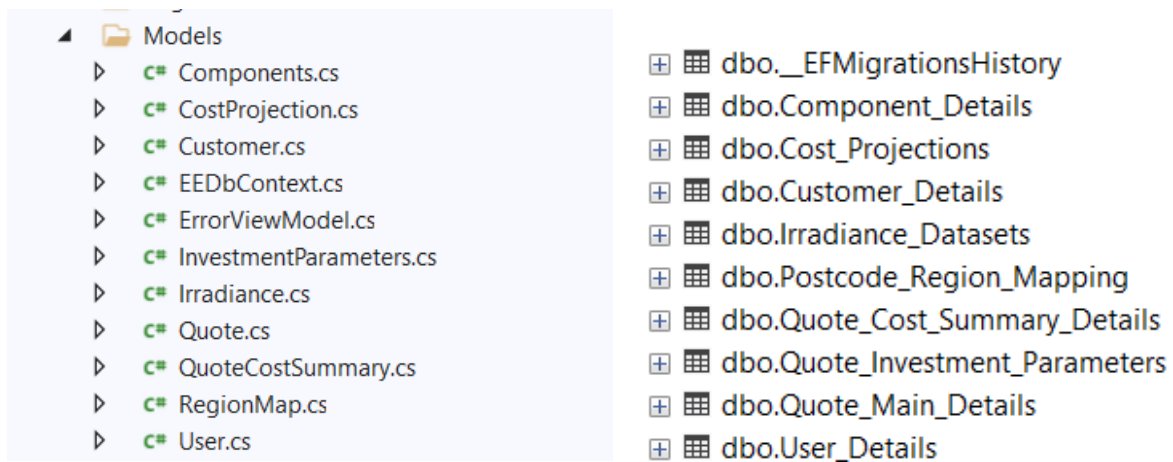
1. ***User:***
 - a. *Login* – View for the user to provide their login credentials
 - b. *Create* – View for the user to register for application access.
2. ***Generate Quote:***
 - a. *QuoteInputs* – View to provide investment and input parameters to calculate cost models
 - b. *CalculateCostModel* – View to show the calculated cost summary and cost projections models.
 - c. *QuoteDetails* – View that shows the entire quote model which includes all the details of the quote.
3. ***History:***
 - a. *Index* – View that shows the list of past generated quotes.
 - b. *Dashboard* – Analytical view of the generated quotes.

5.2 Model Design

One of the main challenges of this application is to design scalable and normalized data structure by defining various data scenarios and entry points. MVC architecture pattern defines data tables as models in the project solution. Entity framework core maps the data tables and columns to the models and its attributes with minimal code.

Based on various defined data scenarios and data entry points, all the required data is categorized and configured as various models and their corresponding table names are given below:

1. User (User_Details)
2. Quote (Quote_Main_Details)
3. Customer (Customer_Details)
4. InvestmentParameters (Quote_Investment_Parameters)
5. Components (Component_Details)
6. CostProjection (Cost_Projections)
7. QuoteCostSummary (Quote_Cost_Summary_Details)
8. RegionMap (Postcode_Region_Mapping)
9. Irradiance (Irradiance_Datasets)



User (User_Details):

User Model represents the data of the user of the application. It consists of attributes such as Id, UserEmail, FirstName, LastName, UserRole, Password, and UserId of the user. The Id attribute is primary key and identity generated value that starts from 1 with an increment of 1. The UserId is a computed column which is a concatenation of first character of FirstName, first character of LastName, and Id.

```

[Table("User_Details")]
public class User
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required(ErrorMessage = "User Email is required")]
    [Column(TypeName = "nvarchar(250)")]
    public string UserEmail { get; set; }

    [Column(TypeName = "varchar(100)")]
    [Required(ErrorMessage = "First Name is required")]
    public string FirstName { get; set; }

    [Column(TypeName = "varchar(100)")]
    [Required(ErrorMessage = "Last Name is required")]
    public string LastName { get; set; }

    [Column(TypeName = "varchar(100)")]
    [Required(ErrorMessage = "Role is required")]
    public string UserRole { get; set; }

    [Column(TypeName = "nvarchar(max)")]
    [Required(ErrorMessage = "Password is required")]
    public string Password { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(10)")]
    public string UserId { get; set; }
}

```

dbo.User_Details
Columns
Id (PK, int, not null)
UserEmail (nvarchar(250), not null)
FirstName (varchar(100), not null)
LastName (varchar(100), not null)
UserRole (varchar(100), not null)
Password (nvarchar(max), not null)
UserId (Computed, varchar(18), null)

Quote (Quote_Main_Details):

Quote Model represents the main data of the quote such as QuoteId, QuoteReference, MasterProjectType, MasterMPAN, QuoteGeneratedBy, QuoteGeneratedDate. The QuoteId attribute is primary key and identity generated value that starts from 1 with an increment of 1. The QuoteReference is a computed column which is a concatenation of Current Year, an underscore, QuoteId, an underscore, MasterProjectType, an underscore, and first two characters of QuoteGeneratedBy.

The other quote details that are required to perform the calculation and quote generation is segregated into various models and glued on to the main quote class through one-to-one and one-to-many relationships for a better scalable and normalized data structure. Here is the list of models that are categorized based on the inputs and outputs of the quote and its relationships with the *Quote* Model.

1. Customer (Customer_Details) – This model holds the customer details of quote. It has a one-to-one relationship with Quote Model.
2. InvestmentParameters (Quote_Investment_Parameters) – This model holds all the system details that customer is agreed to invest on, electricity cost details, annual maintenance cost details, and finance details. It has a one-to-one relationship with Quote Model.
3. Components (Component_Details) – This model consists of data of all the additional components required for the quoting the work. It has a one-to-one relationship with Quote Model.
4. QuoteCostSummary (Quote_Cost_Summary_Details) – This model consists of summary of the calculation of cost models of the given investment and input parameters of the quote. It has a one-to-one relationship with Quote Model.
5. CostProjection (Cost_Projections) – This model consists of calculated cost models and quote model has a one-to-many relationship with this model.

```

[Table("Quote_Main_Details")]
public class Quote
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    public int QuoteId { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(50)")]
    public string QuoteReference { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(15)")]
    [Display(Name = "Master Project Type")]
    public string MasterProjectType { get; set; }

    [Column(TypeName = "nvarchar(max)")]
    [Display(Name = "Master MPAN")]
    public string MasterMPAN { get; set; }

    [Required]
    public DateTime QuoteGeneratedDate { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(15)")]
    public string QuotedGeneratedBy { get; set; }

    public virtual Customer Customer { get; set; }

    public virtual InvestmentParameters InvestmentParam { get; set; }

    public virtual QuoteCostSummary CostSummary { get; set; }

    public virtual List<CostProjection> CostProjections { get; set; }

    public virtual Components Components { get; set; }
}

```

dbo.Quote_Main_Details
Columns
QuoteId (PK, int, not null)
QuoteReference (Computed, nvarchar(52), null)
MasterProjectType (nvarchar(15), not null)
MasterMPAN (nvarchar(max), null)
QuoteGeneratedDate (datetime2(7), not null)
QuotedGeneratedBy (nvarchar(15), not null)

Customer (Customer_Details):

Customer model represents the data of the customer that consists of FirstName, LastName, EmailId, PhoneNumber, Address, Postcode. The QuoteId acts as both foreign key and primary key of the Customer model in the database table as it has a one-to-one relationship with the *Quote* model.

```

[Table("Customer_Details")]
public class Customer
{
    [Key, ForeignKey("QuoteId")]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int QuoteId { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(15)")]
    [Display(Name = "First Name")]
    public string FirstName { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(15)")]
    [Display(Name = "Last Name")]
    public string LastName { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(100)")]
    [Display(Name = "Email ")]
    public string EmailId { get; set; }

    [Required]
    [Display(Name = "Phone Number")]
    public long PhoneNumber { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(max)")]
    [Display(Name = "Address")]
    public string Address { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(15)")]
    [Display(Name = "Postcode")]
    public string PostCode { get; set; }

    public virtual Quote quote { get; set; }
}

```

dbo.Customer_Details
Columns
QuoteId (PK, FK, int, not null)
FirstName (nvarchar(15), not null)
LastName (nvarchar(15), not null)
EmailId (nvarchar(100), not null)
PhoneNumber (bigint, not null)
Address (nvarchar(max), not null)
PostCode (nvarchar(15), not null)

Above diagram shows customer model

InvestmentParameters (Quote_Investment_Parameters):

InvestmentParameters model represents the various categories of the quote inputs. It consists of System details of the energy solution that customer is willing to invest on such as Cost of System, Panel Details. It also consists details of Current cost of electricity, Annual Maintenance packages, Price Inflation. The QuoteId of this model acts as both primary key and foreign key in relation with the *Quote* model in the database table.

```
[Table("Quote_Investment_Parameters")]
public class InvestmentParameters
{
    [Key, ForeignKey("QuoteId")]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int QuoteId { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(10)")]
    public string Zone { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(30)")]
    [Display(Name = "Panel Make")]
    public string PanelMake { get; set; }

    [Required]
    [Display(Name = "Panel Watts")]
    public int PanelWatts { get; set; }

    [Required]
    [Display(Name = "No of Panels")]
    public int NoOfPanels { get; set; }

    [Required]
    public int Orientation { get; set; }
```

Components (Component_Details):

Components model contains the data of required additional components such as Inverter, Panels, Roof Mounting System, Optimizers, Lifting equipment, DC/AC Isolators, DC/AC Wiring, Waster Removal, Electrical Distribution board connection, Electrical Generation Meter, Batteries, Immersion Controller, EV Chargers, and Pigeon Proofing for the quote. It has a one-to-one relationship with *Quote* model. QuoteId attribute act as both primary key and foreign key of the model in the database table.

```

[Table("Component_Details")]
public class Components
{
    [Key, ForeignKey("QuoteId")]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int QuoteId { get; set; }

    public int Inverter { get; set; }

    [Display(Name = "Roof Mounting System")]
    public bool RoofMountingSystem { get; set; }

    [Display(Name = "Optimizers")]
    public bool Optimizers { get; set; }

    [Display(Name = "Lifting Equipemnt")]
    public bool LiftingEquipment { get; set; }

    [Display(Name = "Waste Removal")]
    public bool WasteRemoval { get; set; }

    [Display(Name = "DC & AC Insulators")]
    public bool DcAcIsolators { get; set; }

    [Display(Name = "DC & AC Wiring")]
    public bool DcAcWiring { get; set; }

    [Display(Name = "Dist. Board Connection")]
    public bool DistributionBoardConnection { get; set; }
}

```

dbo.Component_Details
Columns
Quoteld (PK, FK, int, not null)
Inverter (int, not null)
RoofMountingSystem (bit, not null)
Optimizers (bit, not null)
LiftingEquipment (bit, not null)
WasteRemoval (bit, not null)
DcAcIsolators (bit, not null)
DcAcWiring (bit, not null)
DistributionBoardConnection (bit, not null)
ElectricalGenerationMeter (bit, not null)
BatteriesESS5kw (bit, not null)
EvChargers (bit, not null)
ImmersionController (bit, not null)
PegionProofing (bit, not null)
SystemInstallationWarranties (bit, not null)
MCSRegistration (bit, not null)
StructuralReport (bit, not null)

QuoteCostSummary (Quote_Cost_Summary_Details):

QuoteCostSummary represents the summary of the calculations of the cost models which comprises of the Estimated 20 Year profit, Capital Outlay, Yield, IRR, Assumed Annual Electricity Generated, CO2 Savings per 20 years, CO2 Savings per year, Pack back years and Finance type. QuoteId acts as both primary and foreign key for the Quote_Cost_Summary_Details table which is the corresponding table for this model. It has one-to-one relationship with *Quote*.


```

[Table("Quote_Cost_Summary_Details")]
public class QuoteCostSummary
{
    [Key, ForeignKey("QuoteId")]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int QuoteId { get; set; }

    [Required]
    public int CapitalOutlay { get; set; }

    [Required]
    [Column(TypeName = "decimal(10,2)")]
    public double EST20YearProfit { get; set; }

    [Required]
    [Column(TypeName = "decimal(10,2)")]
    public double AssumedAnnualElectricityGenerated { get; set; }

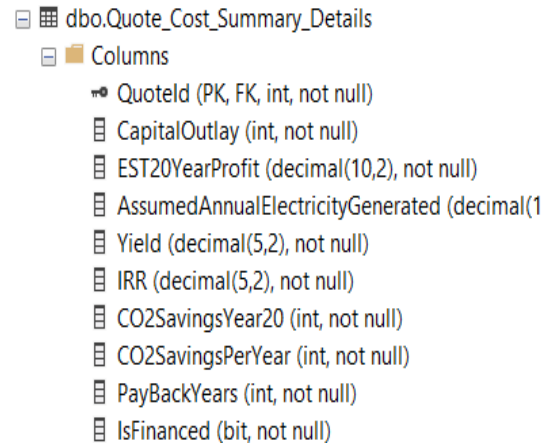
    [Required]
    [Column(TypeName = "decimal(5,2)")]
    public double Yield { get; set; }

    [Required]
    [Column(TypeName = "decimal(5,2)")]
    public double IRR { get; set; }

    [Required]
    public int CO2SavingsYear20 { get; set; }

    [Required]
    public int CO2SavingsPerYear { get; set; }
}

```



Above diagram shows Quote summary model

CostProjection (Cost_Projections):

CostProjection Model consists of 20-year projections of calculated estimated profits of the invested amount in the system installation. It consists of attributes such as Year, Current Cost per KW, Electricity Generated Kwh, Insurance Cost, Maintenance Cost, Revenue from Saving, Revenue from Exporting, Cash Flow, Finance cost details and Final Balance.

`QuoteId` acts as a foreign key and combination of `QuoteId`, and `Year` of cost projection acts as primary key for the model. Quote Model has a one-to-many relationship with the *CostProjection* as each quote will consists of 20-year cost projections.

```

[Table("Cost_Projections")]
public class CostProjection
{
    [ForeignKey("QuoteId")]
    public int QuoteId { get; set; }

    [Required]
    public int Year { get; set; }

    [Required]
    [Column(TypeName = "decimal(5,3)")]
    public double CurrentCostPerkWp { get; set; }

    [Required]
    [Column(TypeName = "decimal(20,2)")]
    public double OutputkWh { get; set; }

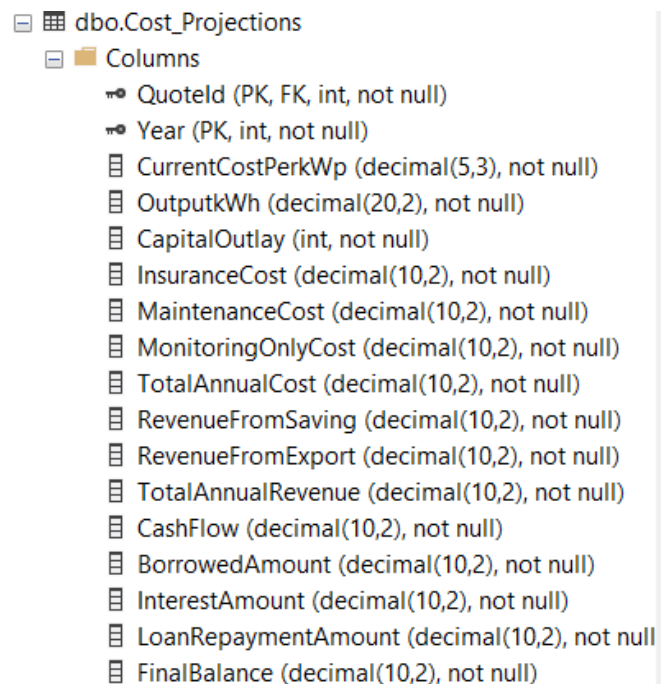
    [Required]
    public int CapitalOutlay { get; set; }

    [Required]
    [Column(TypeName = "decimal(10,2)")]
    public double InsuranceCost { get; set; }

    [Required]
    [Column(TypeName = "decimal(10,2)")]
    public double MaintenanceCost { get; set; }

    [Required]
    [Column(TypeName = "decimal(10,2)")]
    public double MonitoringOnlyCost { get; set; }
}

```



RegionMap (Postcode_Region_Mapping):

RegionMap Model represents the data of Regional Postcode Mapping in the application. The corresponding table of this model consists of two columns named Postcode and Region. Unlike the other models, there is no primary key specified for this model.

```
[Table("Postcode_Region_Mapping")]
[Keyless]
public class RegionMap
{
    [Required]
    [Column(TypeName = "nvarchar(30)")]
    public string Postcode { get; set; }

    [Required]
    [Column(TypeName = "nvarchar(20)")]
    public string Region { get; set; }
}
```

dbo.Postcode_Region_Mapping

Columns

Postcode (nvarchar(30), not null)

Region (nvarchar(20), not null)

Irradiance (Irradiance_Datasets):

Irradiance Model represents the data of Irradiance datasets in the application. The data consists of Irradiance value in a zone based on Orientation and Inclination of the location. It consists of attributes such as Zone, Orientation, Inclination and Annual Generation (MCS Standard). Unlike the other models, there is no primary key specified for this model.

```
[Table("Irradiance_Datasets")]
[Keyless]
public class Irradiance
{
    [Required]
    [Column(TypeName = "nvarchar(20)")]
    public string Zone { get; set; }

    [Required]
    public int Inclination { get; set; }

    [Required]
    public int Orientation { get; set; }

    [Required]
    public int AnnualGenValue { get; set; }
}
```

dbo.Irradiance_Datasets

Columns

Zone (nvarchar(20), not null)

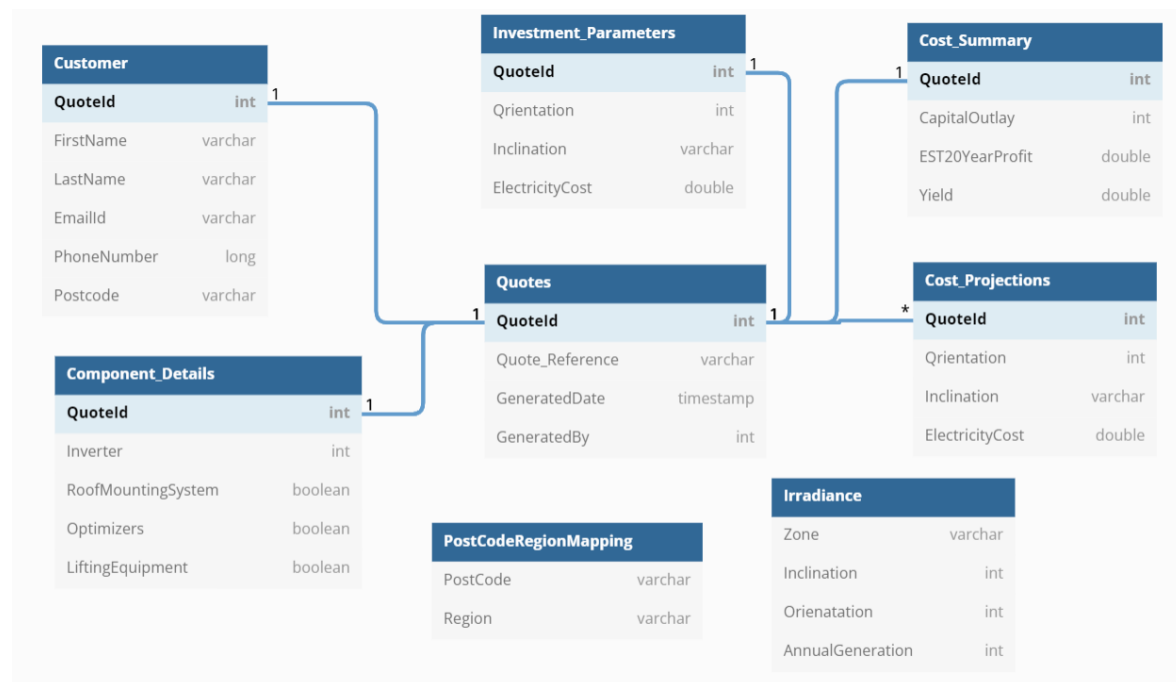
Inclination (int, not null)

Orientation (int, not null)

AnnualGenValue (int, not null)

Model Diagram:

The below diagram shows the models and their relation between them in this application.



5.3 GUI Design

An easier User Interface is what differentiates a great application from a good application. The Views in the MVC architecture renders the model data and sent back to client browser by the controller. Razor View engine is used to write the server code and html in the web pages using C# which must be enclosed with @{..} in the html code. Based on the requirements, we have developed multiple views for each controller. The User Interface developed in this application are responsive web pages. Bootstrap, a front-end web framework that includes HTML, CSS based templates and many other optional JavaScript plugins provides the ability to develop responsive web pages. We will discuss each view of the controller in detail and the shared views of the entire application.

1. Login (User Controller):

Login view is a responsive web page that contains the shared header and footer components. It mainly consists of the user form to provide username and password. The page is easy to understand and easy to navigate to the create account page if the user doesn't have an account. The below images show the page across both laptops, tablets, and mobile devices.

environmentalenergies

Login

Email

Type your email

Password

Type your password

Login

Doesn't have an account? [Create One](#)

© 2022 - Environmental Energies Ltd

environmentalenergies

Login

Email

Type your email

Password

Type your password

Login

Doesn't have an account? [Create One](#)

© 2022 - Environmental Energies Ltd

Above diagram shows login page

2. Create (User Controller):

Create view is a responsive web page that contains the shared header and footer components. It also contains the user form to provide all the necessary details to create an account to access this application. This also provides an easy understandable navigation to login screen once the account is created. This page is also developed to be responsive across multiple devices.

environmentalenergies

Create Account

First Name

Type your first name

Last Name

Type your last name

Email

Type your email

Password

Type your password

Role

--Select role--

Confirm Password

Repeat your password

Create

[Back to Login](#)

© 2022 - Environmental Energies Ltd

environmentalenergies

Create Account

First Name

Type your first name

Last Name

Type your last name

Email

Type your email

Role

--Select role--

Password

Type your password

Confirm Password

Repeat your password

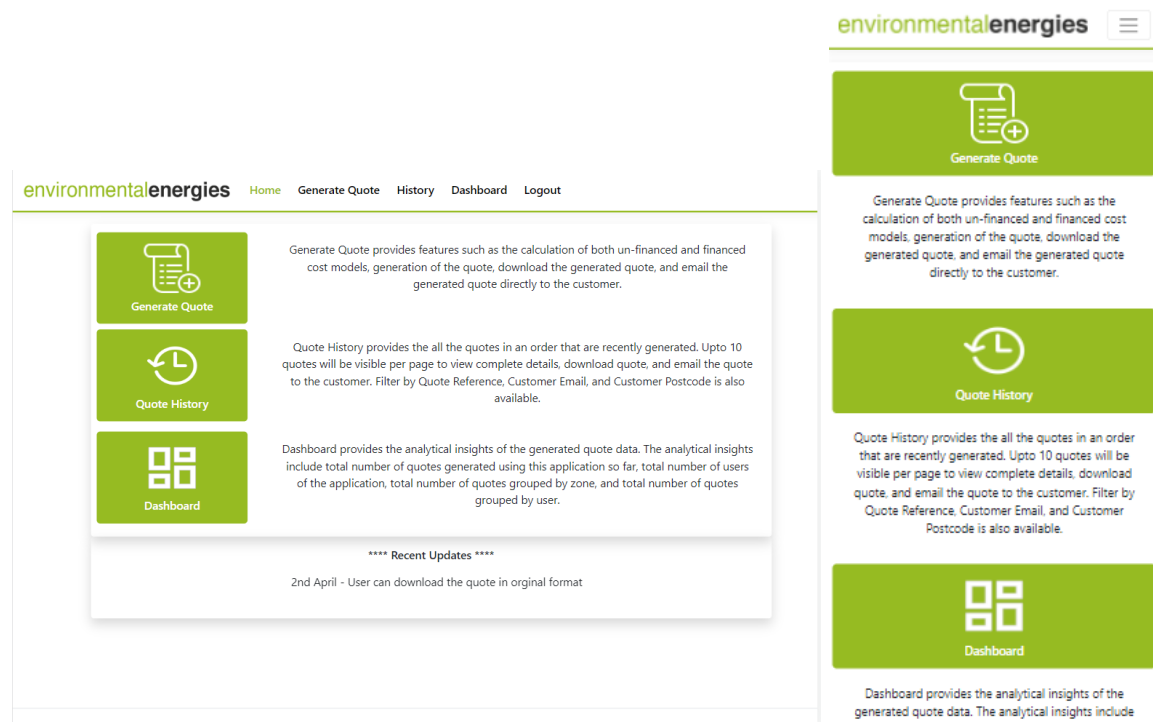
Create

[Back to Login](#)

© 2022 - Environmental Energies Ltd

3. Index (Home Controller):

The Index page of the home controller acts as the home screen for the application once user successfully logs in and easily indicated as such in the navigation bar of the page. This responsive web page consists of links to other functionalities of the application with a three-line description. The navigation bar (header) and footer in this page are shared components. These are shared across all other pages excluding the login and create pages. The three main functionalities of the tool such as Generate Quote, History, and Dashboard are provided as a clickable tile with an icon. This page also consists of a section that represents recent updates in the application for the employees of Environmental energies to keep up to date with the application.



4. Quote Inputs (Generate Quote Controller):

The Quote Inputs page of the Generate Quote Controller contains the navigation bar (header) and footer shared components. It consists of a responsive form to provide the input and investment parameters of the quote. The form inputs are divided into categories based on the data and made it easier for the user (site engineer) to provide the inputs in a convenient way and helps the user to skip the non-required inputs to concentrate more on required inputs based on the categories.

environmentalenergies

HomeGenerate QuoteHistoryDashboardLogout

Solar Electricity Investment Appraisal

Customer Details

First Name

Type first name

Last Name

Type last name

Email

Type email

Phone Number

Type phone number

Address

Type address

Postcode

Type postcode

System Details

Master MPAN

Type master MPAN

Master Project Type

Enter project type

Roof Mount Type

--Select Type--

Panel Make

TBC

Panel Watts

Enter panel watts

No of Panels

Enter no of panels

Orientation

Enter orientation

Inclination

Enter inclination

System Cost

Enter system cost

Shade Factor

Enter shade factor

Electricity and Cost Details

Current Electricity Cost

0.2400

Green Levy Taxes Cost

0.1000

Customer Details

First Name

Type first name

Last Name

Type last name

Email

Type email

Phone Number

Type phone number

Address

Type address

Postcode

Type postcode

System Details

Master MPAN

Type master MPAN

Master Project Type

Above diagram shows quote inputs page

5. Calculate Cost Model (Generate Quote Controller):

The calculate cost model page contains the shared components along with the calculations of cost projections and cost summary of the provided inputs. The cost summary and cost projections are presented in separate sections and as a responsive table. It also consists of two buttons that are situated on the top left and right side the page indicating user either to go back for updating the inputs or proceed with the results.

environmentalenergies

HomeGenerate QuoteHistoryDashboardLogout

Back

Proceed

Cost Summary

Model Type	Capital Outlay	Estimated 20 Year Profit	Yield	20 Year Co2 Savings
UN-FINANCED	£3400	£34600	36	36

Detailed Profit Estimation

Year	Current Cost Per kWh	Output kWh	Total Annual Cost	Total Revenue	Cash Flow	Final Balance
1	0.34	3564	£0	£1212	£1212	£-2188
2	0.357	3546	£0	£1266	£1266	£-922
3	0.375	3528	£0	£1323	£1323	£401
4	0.394	3511	£0	£1383	£1383	£1784
5	0.414	3493	£0	£1446	£1446	£3230
6	0.435	3476	£0	£1512	£1512	£4742

environmentalenergies

BackProceed

Cost Summary

Model Type	Capital Outlay	Estimated 20 Year Profit	Yield	20 Year Co2 Savings
UN-FINANCED	£3400	£34600	36	36

Detailed Profit Estimation

Current Cost per kWh	Output kWh	Total Annual Cost	Total Revenue	Cash Flow
0.34	3564	£0	£1212	£1212
0.357	3546	£0	£1266	£1266
0.375	3528	£0	£1323	£1323
0.394	3511	£0	£1383	£1383
0.414	3493	£0	£1446	£1446
0.435	3476	£0	£1512	£1512

6. Quote Details (Generate Quote Controller):

The Quote Details page contains the complete details of the quote generated in sections as the data is divided into categories to provide a clear understanding. This page also consists of buttons to download the quote, email the quote, and edit the quote details on the top right of the page and quote reference number on the top left of the page.

environmentalenergies

HomeGenerate QuoteHistoryDashboardLogout

Quote Details - 2022/5/S-B-EV/AN

EditDownloadEmail

Customer Details

First Name	Last Name	Email
Akhil	Nuthakki	akhil.nuthakki1@gmail.com
Phone Number	Address	Postcode
7438221589	5 DANBURY PLACE	LE5 0AZ

System Details

Master MPAN	Master Project Type	Roof Mount Type	
MPAN TEXT	S-B-EV	In-Roof	
Panel Make	Panel Watts	No of Panels	System Size
TBC	330	12	3.96
Orientation	Inclination	System Cost	Shade Factor
35	10	3400	10

Electricity and Cost Details

Current Electricity Cost	Green Levy Taxes Cost	Total Electricity Cost
0.24	0.1	0.34
Percentage of Electricity Exported	Assumed Export	Export Tariff
	0	0.02

Quote Details - 2022/5/S-B-EV/AN

EditDownloadEmail

Customer Details

First Name

Akhil

Last Name

Nuthakki

Email

akhil.nuthakki1@gmail.com

Phone Number

7438221589

Address

5 DANBURY PLACE

Postcode

LE5 0AZ

System Details

Master MPAN

MPAN TEXT

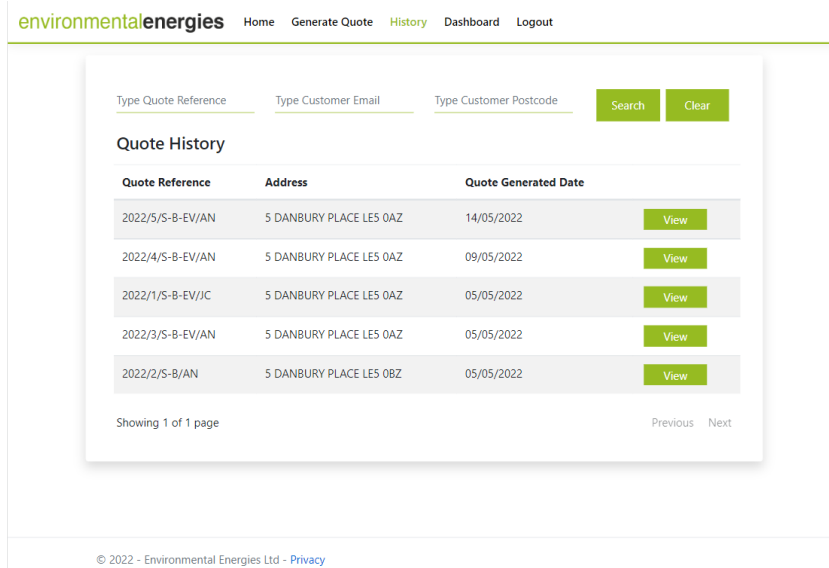
Master Project Type

S-B-EV

Roof Mount Type

7. Index (History Controller):

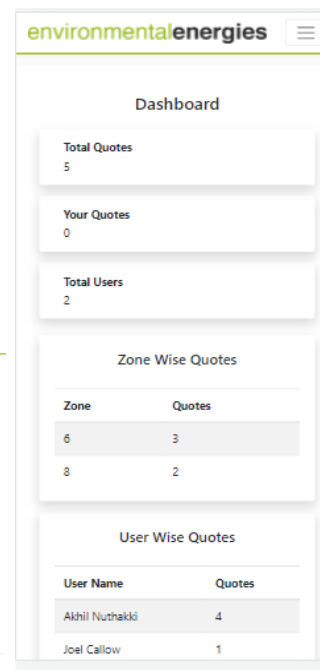
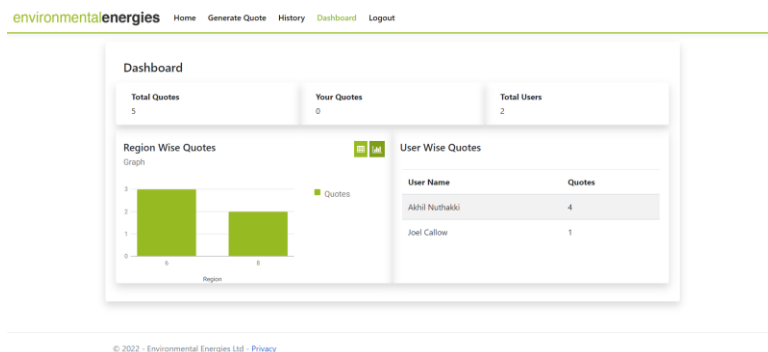
The Index page of the history controller consists of navigator bar (header) and footer shared components. It is a responsive web page that contains a paginated list of quotes in recent generated order. The total number of pages and current page showing is on the bottom left of the page and links to navigate to the next and previous pages. This list of quotes is presented 10 per page in a responsive table. This web page also consists of a form on the top to filter the data.



Above diagram shows history index page

8. Dashboard (History Controller):

The Dashboard web page is a responsive web page that provides the analytical insights of the quotes generated. The page is divided into sections to provide a clear view for the user while searching for the required analysed data. The data in some sections are presented in a responsive table to be able to access across multiple devices.



6. Implementation

DbContext:

This section explains the implementation of the application DbContext by extending the DbContext class of the Microsoft Entity Framework Core. EEDbContext is the application DbContext class that consists of DbSet type variables represents models that are used in the application and their corresponding mapping data tables in the database. DbSet class of the entity framework core is used to query and save instances of the entity. It is also used to translate the LINQ queries to conventional queries of the corresponding database.

Before proceeding further on this, let's see the declaration of these entities that maps to a data table when using migrations of the entity framework. The annotations used in the process of mapping entities to the data tables are *Table* (to represent model to a table), *Key* (represents primary key of table), *Required* (represents not null constraint), *Column* (represents column of the table), *NotMapped* (represents the variable not mapped as column).

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace EEQuoteGenerator.Models
{
    [Table("Customer_Details")]
    public class Customer
    {
        [Key, ForeignKey("QuoteId")]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int QuoteId { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(15)")]
        [Display(Name = "First Name")]
        public string FirstName { get; set; }
    }
}
```

After the declaration of all the models (entities) that needs to map into data tables in data base, all these entities are declared as type DbSet variables in the EEDbContext class. The additional constraints and composite primary key that cannot be declared via annotations needs to be implemented in the overridden method *OnModelCreating* of the DbContext class. The *OnModelCreating* method consists of creation of userEmail Unique Index, UserId Unique Index and UserId computed column of the User model. It also consists of creation of computed column of Quote Reference of Quote model and composite key of cost projections model.

```

public class EEDbContext:DbContext
{
    public EEDbContext(DbContextOptions<EEDbContext> options):base(options)
    {
        //this.ChangeTracker.LazyLoadingEnabled = false;
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>().HasIndex(user => user.UserEmail).IsUnique();
        modelBuilder.Entity<User>().HasIndex(user => user.UserId).IsUnique();
        modelBuilder.Entity<User>().Property(user => user.UserId).HasComputedColumnSql("SUBSTRING([FirstName], 1, 1) + SUBSTRING([LastNa

        modelBuilder.Entity<Quote>().Property(quote => quote.QuoteReference).HasComputedColumnSql("CAST(YEAR(GETDATE())) As varchar(16))

        modelBuilder.Entity<CostProjection>().HasKey(nameof(CostProjection.QuoteId), nameof(CostProjection.Year));
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Irradiance> IrradianceDatasets { get; set; }
    public DbSet<RegionMap> RegionMappings { get; set; }
    public DbSet<InvestmentParameters> QuoteInvestmentParameters { get; set; }
    public DbSet<Quote> Quotes { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<QuoteCostSummary> Summary { get; set; }
    public DbSet<CostProjection> Projections { get; set; }
    public DbSet<EEQuoteGenerator.Models.Components> Components { get; set; }
}

```

Above Diagram shows the application DbContext class.

After the implementation of EEDbContext class of the application, EF Code First to Database migrations is used to create database with specified models as tables using commands such as *Add-Migration*, *Remove-Migration*, and *Update-Database*.

User Controller:

User controller handles the requests of user authentication and user registration to access the application. It consists of five action methods. The detailed description of these action methods and URL's endpoints to execute these actions is given below:

1. Login (*HTTP Get*): This is the start-up page of the application. The URL endpoint for this action is <https://localhost/User/Login>. This method returns the Login View that contains a form to provide username and password for the authentication. Login View also contains a link to user registration page in case if user doesn't have an account and need to register to access the application.
2. Login (*HTTP Post*): This method is called when the user submits the login form with their username and password. This method validates the login credentials and returns the Index View of the Home Controller. The steps involved in the validation of the login credentials is as follows:
 1. It first checks if the given user email exists in the database.
 2. If user email Id exists, it retrieves the entire user object for the specific email. If it doesn't exist, the method will return login view with a message saying user email is not registered.
 3. Once the entire user object is retrieved, it is then checks if the password provided matches with password in the database. If it doesn't match, method will return login view with a message saying password did not match.
 4. If the given password matches with DB, it sets a HTTP Session variable with UserId value and returns the Index View of the Home controller.
 5. The password in the DB will be encrypted and needs to decrypt before matching with the given password. The decryption is done using decrypt method of encryption service. The detailed description of encryption service is given below.

```

public async Task<IActionResult> Login([Bind("UserEmail,Password")] User LoggedInUser)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.UserEmail == LoggedInUser.UserEmail);
    if (user == null)
    {
        ViewBag.ErrorMessage = "Email doesnot exists. Please create a new account.";
    }
    else
    {
        user.Password = EncryptionService.Decrypt(user.Password);
        if (user.Password != LoggedInUser.Password)
        {
            ViewBag.ErrorMessage = "Incorrect password.";
        }
        else
        {
            HttpContext.Session.SetString("UserId", user.UserId);
            return RedirectToAction("Index", "Home");
        }
    }

    return View(LoggedInUser);
}

```

Above Diagram shows implementation of Login post method.

3. Create (*HTTP Get*): This method is called when the user clicks on the link provided in the Login View to create an account if they don't have one. The URL endpoint for this action is <https://localhost:/User/Create>. This method returns the Create View that contains a form to provide user details to register for the access of this application.
4. Create (*HTTP Post*): This method is called when the user submits the create form with the user details. This method validates the given details and returns the Login View with a message saying to login with the registered details. The steps involved in the validation of given details is as follows:
 1. It first checks if the given user email exists in the database.
 2. If the user exists in the database, it will return the Create View with a message saying the user email is already exists in the database. Please try to login.
 3. If the user email doesn't exist in the database, it will save the details to the DB. The given password will be encrypted using encrypt method of encryption service before saving the data in database. The detailed description of encryption service is given below.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("UserEmail,FirstName,LastName,UserRole,Password")] User user)
{
    ViewData["RolesList"] = EEQuoteGenerator.Models.User.GetRolesList();
    var DbUser = await _context.Users.FirstOrDefaultAsync(u => u.UserEmail == user.UserEmail);
    if (DbUser == null)
    {
        user.Password = EncryptionService.Encrypt(user.Password);
        _context.Add(user);
        await _context.SaveChangesAsync();
        ViewBag.ErrorMessage = "Please login with your registered details";
        return View("Login");
    }
    else
    {
        ViewBag.ErrorMessage = "Email already exists. Please click below to login";
    }

    return View(user);
}

```

5. Logout (*HTTP Get*): This method is called when the user clicks on the logout link provided navigation bar of the application. The URL endpoint for this action is <https://localhost/User/Logout>. This method returns the Login View that contains a message saying that the user is successfully logged out. It also clears the UserId http session variable.

Encryption Service:

Encryption service provides the encrypt and decrypt methods which will return the encrypted and decrypted the string parameter using a 128-bit key respectively. The entire Encryption class is created from the existing Git code provided in the reference [8].

```
private static string AES256_KEY = "RgUkXp2s5v8y/B?E";

public static string Encrypt(string plainText)
{
    byte[] cipherData;
    Aes aes = Aes.Create();
    aes.Key = Encoding.UTF8.GetBytes(AES256_KEY);
    aes.GenerateIV();
    aes.Mode = CipherMode.CBC;
    ICryptoTransform cipher = aes.CreateEncryptor(aes.Key, aes.IV);

    using (MemoryStream ms = new MemoryStream())
    {
        using (CryptoStream cs = new CryptoStream(ms, cipher, CryptoStreamMode.Write))
        {
            using (StreamWriter sw = new StreamWriter(cs))
            {
                sw.Write(plainText);
            }
        }

        cipherData = ms.ToArray();
    }

    byte[] combinedData = new byte[aes.IV.Length + cipherData.Length];
    Array.Copy(aes.IV, 0, combinedData, 0, aes.IV.Length);
    Array.Copy(cipherData, 0, combinedData, aes.IV.Length, cipherData.Length);
    return Convert.ToBase64String(combinedData);
}
```

Generate Quote Controller:

Generate Quote Controller handles the requests of calculation of cost models for the given investment and input parameters, download the generated quote in the original format, email the generated quote to the customer, and view the complete details of a generated quote. The detailed description of action methods in this controller and the URL endpoints to execute these actions is given below:

1. QuoteInputs (*HTTP Get*): The method is called when the user clicks on the Generate Quote option. The URL endpoint for this action method is <https://localhost/GenerateQuote/QuoteInputs>. This method returns the Quote Inputs View of the controller that contains the form for the user to provide all the required quote details such as customer details, system details, electricity cost details, Annual package details, and finance details to calculate the cost projections of the invested system.

```
//GET: GenerateQuote/QuoteInputs
public IActionResult QuoteInputs()
{
    string UserId = HttpContext.Session.GetString("UserId");
    if (UserId == null)
    {
        return RedirectToAction("Login", "User");
    }
    ViewData["ProjectTypeList"] = EEQuoteGenerator.Models.Quote.GetProjectTypeList();
    return View("QuoteInputs");
}
```

Above Diagram shows implementation of Quote Inputs get method.

2. EditQuoteInputs (*HTTP Get*): This method is called when user clicks on the edit option of the quote and passes QuoteId as a parameter. The URL endpoint for this action method is <https://localhost:GenerateQuote/EditQuoteInputs/{id}>. The QuoteId is passed as a path variable to the action. The complete quote object including the customer object, investment parameters object, components object is retrieved from the DB and passed to QuoteInputs View for the user to edit the Quote inputs for the given QuoteId.

```
//GET: GenerateQuote/EditQuoteInputs/id
public async Task<IActionResult> EditQuoteInputs(int Id)
{
    string UserId = HttpContext.Session.GetString("UserId");
    if (UserId == null)
    {
        return RedirectToAction("Login", "User");
    }
    ViewData["ProjectTypeList"] = EEQuoteGenerator.Models.Quote.GetProjectTypeList();
    if (Id == 0)
    {
        return View("QuoteInputs");
    }
    var quote = await _context.Quotes
        .Include(quote => quote.Customer)
        .Include(quote => quote.InvestmentParam)
        .Include(quote => quote.CostSummary)
        .Include(quote => quote.CostProjections)
        .Include(quote => quote.Components)
        .Where(quote => quote.QuoteId == Id)
        .FirstOrDefaultAsync();
    return View("QuoteInputs", quote);
}
```

3. CalculateCostModel (*HTTP Post*): This method is called when the user submits the Investment Appraisal form and validation for the given inputs is successful. The Razor View Engine binds the form inputs to Quote model before it is called. The URL endpoint for this method is <https://localhost:GenerateQuote/CalculateCostModel>. The cost summary object and list of cost projections object will be updated to the Quote model in below given steps:
 - a. The region (zone) that the customer belongs to is retrieved from the Region Postcode Mapping table using the customer postcode.

```

public string GetZonefromPostCode(string postcode)
{
    String Zone = null;
    Regex CharCodeRGX = new Regex("[A-Z]{1,2}");
    Regex NumCodeRGX = new Regex("[0-9]{1,2}");
    string PCCharCode = "";
    int PCNumCode = 0;
    MatchCollection CharCodeMatches = CharCodeRGX.Matches(postcode);
    if (CharCodeMatches.Count > 0)
    {
        PCCharCode = CharCodeMatches.ElementAt(0).Value;
    }
    MatchCollection NumCodeMatches = NumCodeRGX.Matches(postcode);
    if (NumCodeMatches.Count > 0)
    {
        PCNumCode = Int32.Parse(NumCodeMatches.ElementAt(0).Value);
    }

    var RegionZoneMap = _context.RegionMappings.Where(r => r.Postcode.StartsWith(PCCharCode)).ToList();

    if (RegionZoneMap.Count == 0)
    {
        Zone = null;
    }
    else if (RegionZoneMap.Count == 1)
    {
        Zone = RegionZoneMap.ElementAt(0).Region;
    }
    else
    {
        string DefaultZone = null;
        foreach (RegionMap map in RegionZoneMap)
        {
            Regex rgx = new Regex("[0-9]{1,2}");
            MatchCollection bounds = rgx.Matches(map.Postcode);
            if (bounds.Count > 0)
            {
                if (bounds.Count == 1)
                {
                    int value = Int32.Parse(bounds.ElementAt(0).Value);
                    if (PCNumCode == value) { Zone = map.Region; }
                }
            }
        }
    }
}

```

Above Diagram shows implementation of algorithm to retrieve zone from given postcode.

- b. The Annual Generation (MCS Standard) value is retrieved from the Irradiance datasets using Zone, Orientation, and Inclination.
- c. The Final Annual Generation value of Quote object is updated after taking shade factor into account.
- d. The Quote Object of Updated Annual Generation and Zone fields is then passed to *CalculateandUpdateCostSummaryandProjections* method of *CalculateCostModel* Service to get updated with the cost projections and cost summary. The detailed description of *CalculateCostModel* Service and its methods is given below.
- e. Once updated, the method *CalculateandUpdateCostSummaryandProjections* returns the complete Quote Object.

The complete Quote Object is then saved into DB if the QuoteId of the Quote Object is 0. If the QuoteId of the Quote is not equal to 0, then it will be updated to the database. After save/update to the database, the method returns the view of the controller that contains cost summary and cost projections for the user to check out the results. User can either proceed with quote or edit the provided inputs.

```

{
    string UserId = HttpContext.Session.GetString("UserId");
    if(UserId == null)
    {
        return RedirectToAction("Login", "User");
    }
    string Zone = null;
    Quote.Customer.PostCode = Quote.Customer.PostCode.ToUpper();
    Zone = GetZonefromPostCode(Quote.Customer.PostCode);

    if(Zone == null)
    {
        ViewBag.ErrorMessage = "Couldn't able to find mapping Zone for the given postcode. Please contact System Administrator.";
        return View("QuoteInputs");
    }

    var IrradianceMap = await _context.IrradianceDatasets.FirstOrDefaultAsync(i => i.Zone == Zone && i.Inclination == Quote.InvestmentParam.Inclination && i.Orientation == Quote.InvestmentParam.Inclination && i.Orientation == Quote.InvestmentParam.Inclination);
    Quote.InvestmentParam.AnnualGeneration = IrradianceMap.AnnualGenValue - (IrradianceMap.AnnualGenValue * Quote.InvestmentParam.ShadeFactor)/100;
    Quote.InvestmentParam.Zone = Zone;

    Quote.QuoteGeneratedDate = DateTime.Now;
    Quote.QuoteGeneratedBy = UserId;

    Quote = CalculateCostModelService.CalculateandUpdateCostSummaryandProjections(Quote);

    try
    {
        if (Quote.QuoteId != 0)
        {
            Quote = UpdateAllPrimaryKeysOfEntites(Quote);
            _context.Quotes.Update(Quote);
            await _context.SaveChangesAsync();
        }
        else
        {
            var quote = _context.Quotes.Add(Quote);
            await _context.SaveChangesAsync();
            Quote.QuoteId = quote.Entity.QuoteId;
        }
    }
    catch(Exception ex)
    {
    }
}

```

4. **DownloadQuote (HTTP Get):** This method is called when the user clicks on the Download option of the Quote. The URL endpoint for this method is <https://localhost/GenerateQuote/DownloadQuote/{id}>. This method retrieves the complete quote object from the DB using the parameter Id passed as a path variable. The complete quote object is then passed to *CreateQuoteDocument* method of *GenerateQuote* Service. The *CreateQuoteDocument* method will create the quote for the specified quote and returns the server path of generated file to the action method. The file path is then used to memory stream the file content and return the file using File Stream Result of ms-word type. The detailed description of *GenerateQuote* Service and its methods is given below

```

public async Task<IActionResult> DownloadQuote(int Id){
    string UserId = HttpContext.Session.GetString("UserId");
    if (UserId == null)
    {
        return RedirectToAction("Login", "User");
    }
    var User = await _context.Users.FirstOrDefaultAsync(user => user.UserId == UserId);
    if (Id == 0)
    {
        return RedirectToAction(nameof(ProcessQuote), new { Id = 0 });
    }
    var Quote = await _context.Quotes
        .Include(quote => quote.Customer)
        .Include(quote => quote.InvestmentParam)
        .Include(quote => quote.CostSummary)
        .Include(quote => quote.CostProjections)
        .Include(quote => quote.Components)
        .Where(quote => quote.QuoteId == Id)
        .FirstOrDefaultAsync();

    string DestinationFile = GenerateQuoteService.CreateQuoteDocument(Quote, User, ".docx");

    //FileStream fs = new FileStream(DestinationFile, FileMode.Open);

    var memory = new MemoryStream();
    using (var stream = new FileStream(DestinationFile, FileMode.Open))
    {
        await stream.CopyToAsync(memory);
    }
    memory.Position = 0;

    GenerateQuoteService.DeleteQuote(DestinationFile);

    return File(memory, "application/vnd.ms-word", Path.GetFileName(DestinationFile));
}

```

5. **EmailQuote (HTTP Get):** This method is called when the user clicks on the Email option of the Quote. The URL endpoint for this method is <https://localhost:/GenerateQuote/EmailQuote/{id}>. This method retrieves the complete quote object from the DB using the parameter Id passed as a path variable. The complete quote object is then passed to *CreateQuoteDocument* method of *GenerateQuote* Service. The *CreateQuoteDocument* method will create the quote for the specified quote and returns the server path of generated file to the action method. The server path of quote is then passed to *SendEmail* method of *EmailSender* Service along with complete Quote object and User object. The *SendEmail* method of *EmailSender* Service will send an email to the customer attaching the pdf version of the Quote. Once email is sent, the document is then deleted from the server path. The detailed description of *EmailSender* Service and its methods is given below.

```
public async Task<IActionResult> EmailQuote(int Id)
{
    string UserId = HttpContext.Session.GetString("UserId");
    if (UserId == null)
    {
        return RedirectToAction("Login", "User");
    }
    if (Id == 0)
    {
        return RedirectToAction(nameof(ProcessQuote), new { Id = 0 });
    }

    var Quote = await _context.Quotes
        .Include(q => q.Customer)
        .Include(q => q.InvestmentParam)
        .Include(q => q.CostSummary)
        .Include(q => q.CostProjections)
        .Include(q => q.Components)
        .Where(q => q.QuoteId == Id)
        .FirstOrDefaultAsync();
    var User = await _context.Users.FirstOrDefaultAsync(u => u.UserId == UserId);

    string DestinationFile = GenerateQuoteService.CreateQuoteDocument(Quote, User, "pdf");

    await EmailSenderService.SendEmailAsync(Quote.Customer.EmailId, "Environmental Energy LTD - Quote Reference " + Quote.QuoteReference.Replace("_", "/"), DestinationFile, Quote, User);

    GenerateQuoteService.DeleteQuote(DestinationFile);

    return View("ProcessQuote");
}
```

6. **QuoteDetails (HTTP Get):** This method is called when the user clicks on the View option of the Quote. The URL endpoint for this method is <https://localhost:/GenerateQuote/QuoteDetails/{id}>.

```
public async Task<IActionResult> QuoteDetails(int Id)
{
    if (Id == 0)
    {
        return View("Index");
    }

    var quote = await _context.Quotes
        .Include(q => q.Customer)
        .Include(q => q.InvestmentParam)
        .Include(q => q.CostSummary)
        .Include(q => q.CostProjections)
        .Include(q => q.Components)
        .Where(q => q.QuoteId == Id)
        .FirstOrDefaultAsync();

    return View(quote);
}
```

This method will return the Quote Details View that contains complete details of the quote such as customer details, system details, electricity cost details, annual packages

details, finance details, cost summary, and cost projections. It also includes the options for the user to edit the quote, download the quote, and email the quote to customer.

environmentalenergies

[Home](#)
[Generate Quote](#)
[History](#)
[Dashboard](#)
[Logout](#)

Quote Details - 2022/5/S-B-EV/AN

Edit

Download

Email

Customer Details

First Name	Last Name	Email	
Akhil	Nuthakki	akhil.nuthakki1@gmail.com	
Phone Number	Address	Postcode	
7438221589	5 DANBURY PLACE	LE5 0AZ	

System Details

Master MPAN	Master Project Type	Roof Mount Type	
MPAN TEXT	S-B-EV	In-Roof	
Panel Make	Panel Watts	No of Panels	System Size
TBC	330	12	3.96
Orientation	Inclination	System Cost	Shade Factor
35	10	3400	10

Electricity and Cost Details

Current Electricity Cost	Green Levy Taxes Cost	Total Electricity Cost
--------------------------	-----------------------	------------------------

Email Button in the Quote Details View uses AJAX script which is a web development technique commonly used to create interactive web applications. Using AJAX, the java script written in JS file will make a request to the server, receives the response, and updates the view with the received response. So, using the AJAX script, the entire Quote Details page will not reload when user access the functionality of email to user.

```
$.ajax({
  url: EmailQuoteUrl,
  type: 'GET',
  data: { id: _id },
  contentType: 'application/json; charset=utf-8',
  datatype: 'json',
  success: function (result) {
    if (result.message == 'SUCCESS') {
      document.getElementById('StatusMessage').innerText = 'Email sent successfully !!';
      document.getElementById('StatusDiv').classList.remove('text-danger');
      document.getElementById('StatusDiv').classList.add('text-success');
      document.getElementById('loadingIcon').classList.remove('loading-icon-visible');
      document.getElementById('loadingIcon').classList.add('loading-icon-invisible');
    } else {
      document.getElementById('StatusMessage').innerText = 'Email sent failed !!';
      document.getElementById('StatusDiv').classList.remove('text-success');
      document.getElementById('StatusDiv').classList.add('text-danger');
      document.getElementById('loadingIcon').classList.remove('loading-icon-visible');
      document.getElementById('loadingIcon').classList.add('loading-icon-invisible');
    }
  },
  error: function (result) {
    document.getElementById('StatusMessage').innerText = 'Email sent failed !!';
    document.getElementById('StatusDiv').classList.remove('text-success');
    document.getElementById('StatusDiv').classList.add('text-danger');
    document.getElementById('loadingIcon').classList.remove('loading-icon-visible');
    document.getElementById('loadingIcon').classList.add('loading-icon-invisible');
  }
});
```

Calculate Cost Model Service:

CalculateCostModel Service has a *CalculateandUpdateCostSummaryandProjections* method with a quote object as its input parameter. This method will calculate the cost projections and cost summary and update the summary object and cost projections object of the quote and returns the complete quote object to save it to the DB. The process of calculation of the cost projections involves calculation of cost of electricity, Insurance Cost, Maintenance Cost, monitoring only cost, Total Annual Cost for next 20 years based on price inflation. It also involves the calculation of Electricity Output based on the Annual Generation (MCS Standard) of the zone and deprecation of system for next 20 years. Finally, it calculates the Total Revenue, Cash Flow, and the Final Balance for each year for next 20 years. Cost Summary is calculated based on the loop of these cost projections.

```
for (int i = 1; i <= YEARS_OF_COSTMODEL; i++)
{
    CostProjection costmodel = new CostProjection();
    costmodel.Year = i;
    if (i == 1)
    {
        costmodel.CurrentCostPerKwh = quote_inputs.TotalCurrentElectricityCost;
        costmodel.OutputKwh = Math.Round(quote_inputs.SystemSize * quote_inputs.AnnualGeneration);
        costmodel.CapitalOutlay = quote_inputs.SystemCost;
        costmodel.InsuranceCost = quote_inputs.AnnualInsurance;
        costmodel.MaintenanceCost = quote_inputs.AnnualMaintenancePackage;
        costmodel.MonitoringOnlyCost = quote_inputs.AnnualMonitoringOnlyPackage;

        // Financed Model Calculations
        costmodel.LoanRepaymentAmount = i > quote_inputs.LoanPeriodYears ? 0 : Math.Round(((costmodel.CapitalOutlay * quote_inputs.PercentageFinanced) / 100) / quote_inputs.LoanPeriodYears);
        costmodel.BorrowedAmount = Math.Round(((costmodel.CapitalOutlay * quote_inputs.PercentageFinanced) / 100) - costmodel.LoanRepaymentAmount);
        costmodel.InterestAmount = costmodel.BorrowedAmount > 0 ? Math.Round(((costmodel.CapitalOutlay * quote_inputs.PercentageFinanced) / 100) * quote_inputs.BorrowingCost) / 100 : 0;

        costmodel.TotalAnnualCost = Math.Round(costmodel.InsuranceCost + costmodel.MaintenanceCost + costmodel.MonitoringOnlyCost + costmodel.LoanRepaymentAmount + costmodel.InterestAmount);
        costmodel.RevenueFromSavings = Math.Round((costmodel.OutputKwh - (costmodel.OutputKwh * quote_inputs.PercentageElectricityExported) / 100) * costmodel.CurrentCostPerKwh);
        costmodel.RevenueFromExport = Math.Round((costmodel.OutputKwh * quote_inputs.AssumedExport * quote_inputs.ExportTariff) / 100);
        costmodel.TotalAnnualRevenue = costmodel.RevenueFromExport + costmodel.RevenueFromSavings;
        costmodel.CashFlow = costmodel.TotalAnnualRevenue - costmodel.TotalAnnualCost;
        costmodel.FinalBalance = Math.Round(costmodel.CashFlow - (quote_inputs.SystemCost - ((costmodel.CapitalOutlay * quote_inputs.PercentageFinanced) / 100)));
    }
    else
    {
        costmodel.CurrentCostPerKwh = Math.Round(projections.ElementAt(i - 2).CurrentCostPerKwh + ((quote_inputs.AnnualFuelPriceInflation / 100) * projections.ElementAt(i - 2).CurrentCostPerKwh), 3);
        costmodel.OutputKwh = Math.Round(quote_inputs.SystemSize * quote_inputs.AnnualGeneration * Math.Pow(1 - DEPRECIATION, i - 1));
        costmodel.CapitalOutlay = quote_inputs.SystemCost;
        costmodel.InsuranceCost = Math.Round(projections.ElementAt(i - 2).InsuranceCost + (projections.ElementAt(i - 2).InsuranceCost * (quote_inputs.RPI / 100)));
        costmodel.MaintenanceCost = Math.Round(projections.ElementAt(i - 2).MaintenanceCost + (projections.ElementAt(i - 2).MaintenanceCost * (quote_inputs.RPI / 100)));
        costmodel.MonitoringOnlyCost = Math.Round(projections.ElementAt(i - 2).MonitoringOnlyCost + (projections.ElementAt(i - 2).MonitoringOnlyCost * (quote_inputs.RPI / 100)));

        // Financed Model Calculations
        costmodel.LoanRepaymentAmount = i > quote_inputs.LoanPeriodYears ? 0 : Math.Round(((costmodel.CapitalOutlay * quote_inputs.PercentageFinanced) / 100) / quote_inputs.LoanPeriodYears);
        costmodel.BorrowedAmount = projections.ElementAt(i - 2).BorrowedAmount - costmodel.LoanRepaymentAmount;
        costmodel.InterestAmount = projections.ElementAt(i - 2).BorrowedAmount > 0 ? Math.Round((projections.ElementAt(i - 2).BorrowedAmount * quote_inputs.BorrowingCost) / 100) : 0;

        costmodel.TotalAnnualCost = costmodel.InsuranceCost + costmodel.MaintenanceCost + costmodel.MonitoringOnlyCost + costmodel.LoanRepaymentAmount + costmodel.InterestAmount;
        costmodel.RevenueFromSavings = Math.Round((costmodel.OutputKwh - (costmodel.OutputKwh * quote_inputs.PercentageElectricityExported) / 100) * costmodel.CurrentCostPerKwh);
        costmodel.RevenueFromExport = Math.Round((costmodel.OutputKwh * quote_inputs.AssumedExport * quote_inputs.ExportTariff) / 100);
        costmodel.TotalAnnualRevenue = costmodel.RevenueFromExport + costmodel.RevenueFromSavings;
        costmodel.CashFlow = costmodel.TotalAnnualRevenue - costmodel.TotalAnnualCost;
        costmodel.FinalBalance = projections.ElementAt(i - 2).FinalBalance + costmodel.CashFlow;
    }
}
```

Generate Quote Service:

GenerateQuote Service has a *CreateQuoteDocument* method with a quote object, user object, and format type as its input parameters. *CreateQuoteDocument* method creates a quote document of passed quote in server path and returns the server path of the file back to the caller method. First, it copies the quote template from the resources folder of the application into the system user profile path and rename it with the provided quote reference. Once copied, Microsoft.Office.Interop.Word Application object will be created and assigned with the copied quote specific template. Document Object of the Word is used to access and open the word document provided with the server path of file. Word Application is then used to update all the pages of the document in two ways. One way is to find and replace the text using *Find.Execute()* Method of Selection Object. Other way is to update the tables by accessing every cell of the table. Once updated, the word document is saved in the format provided as in the input parameter. It will be saved as .docx for downloading purpose and .pdf for sending an email to customer. After saving the document in the provided format, the method will return the server path of file to the caller method.

```

public class GenerateQuoteService
{
    public static string CreateQuoteDocument(Quote Quote, User User, string format)
    {
        string SourceTemplate = AppDomain.CurrentDomain.BaseDirectory + "Resources\\Quote_Template.docx";
        string DestinationDirectory = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) + "\\Quote Generation Tool\\Quotes";
        if (!Directory.Exists(DestinationDirectory))
        {
            Directory.CreateDirectory(DestinationDirectory);
        }

        string DestinationFile = DestinationDirectory + "\\Quote_" + Quote.QuoteReference + ".docx";
        DeleteQuote(DestinationFile);
        System.IO.File.Copy(SourceTemplate, DestinationFile);

        Word.Application OWord = new Word.Application();
        Word.Document ODocument = OWord.Documents.Open(DestinationFile);
        ODocument.Activate();
        OWord.Visible = false;

        UpdateHeader(OWord, Quote);
        UpdateQuoteDocument_Page1(OWord, Quote, User);
        UpdateQuoteDocument_Page2(OWord, Quote);
        UpdateQuoteDocument_Page3(OWord, Quote);
        UpdateQuoteDocument_Page4_Page5(OWord, Quote);
        UpdateQuoteDocument_Page12(OWord, Quote);

        if (format.Equals(".docx"))
        {
            ODocument.Save();
            ODocument.Close();
            OWord.Quit();
        }
        else
        {
            Word.WdSaveFormat wdSaveFormatPDF = Word.WdSaveFormat.wdFormatPDF;
            object wdSaveFormat = wdSaveFormatPDF;
            object SaveAsFile = DestinationDirectory + "\\Quote_" + Quote.QuoteReference + ".pdf";

            DeleteQuote(SaveAsFile.ToString());
        }
    }
}

```

Email Sender Service:

EmailSender Service consists of two methods named *SendEmailAsync* and *CreateEmailBody*. *SendEmailAsync* is a synchronous method that has input parameters of email, subject, file Attachment, Quote Object, and User Object. It uses *SmtpClient* class of System.Net.Mail namespace that allows applications to send emails by using Simple Mail Transfer Protocol. This method uses *CreateEmailBody* method to create the HTML body of the template using *AlternateView* of System.Net.Mail to format view of an email message.

```

public class EmailSenderService
{
    private const string MAIL_HOST = "smtp.outlook.com";
    private const string EMAIL_USERNAME = "No-reply.ee-quote-tool@outlook.com";
    private const string EMAIL_PASSWORD = "UOLee@2022";

    public static async Task<bool> SendEmailAsync(string email, string subject, string AttachmentFilePath, Quote Quote, User User)
    {
        try
        {
            SmtpClient client = new SmtpClient(MAIL_HOST, 587);
            client.DeliveryMethod = SmtpDeliveryMethod.Network;
            client.UseDefaultCredentials = false;
            NetworkCredential credentials = new NetworkCredential(EMAIL_USERNAME, EMAIL_PASSWORD);
            client.EnableSsl = true;
            client.Credentials = credentials;
            MailMessage Email = new MailMessage(EMAIL_USERNAME, email);
            Email.Subject = subject;
            Email.IsBodyHtml = true;
            AlternateView HTMLView = CreateEmailBody(Quote, User);
            Email.AlternateViews.Add(HTMLView);
            Email.Attachments.Add(new Attachment(AttachmentFilePath));

            await client.SendMailAsync(Email);

            Email.Attachments.Dispose();
            Email.Dispose();

            return true;
        }
        catch
        {
            return false;
        }
    }
}

```

History Controller:

History Controller handles the requests of retrieval of history of quotes and dashboard that shows the analytical representation of data of the generated quotes. The detailed description of two action methods and URL's endpoints to execute these actions is given below:

1. Index (*HTTP Get*): The URL end point for this action method is <https://localhost:44375/History/Index>. It returns the Index view of the controller that returns the paged list of history of quotes and a form to filter or search for the quote based on the parameters such as Quote Reference, Customer Email and Customer Postcode.

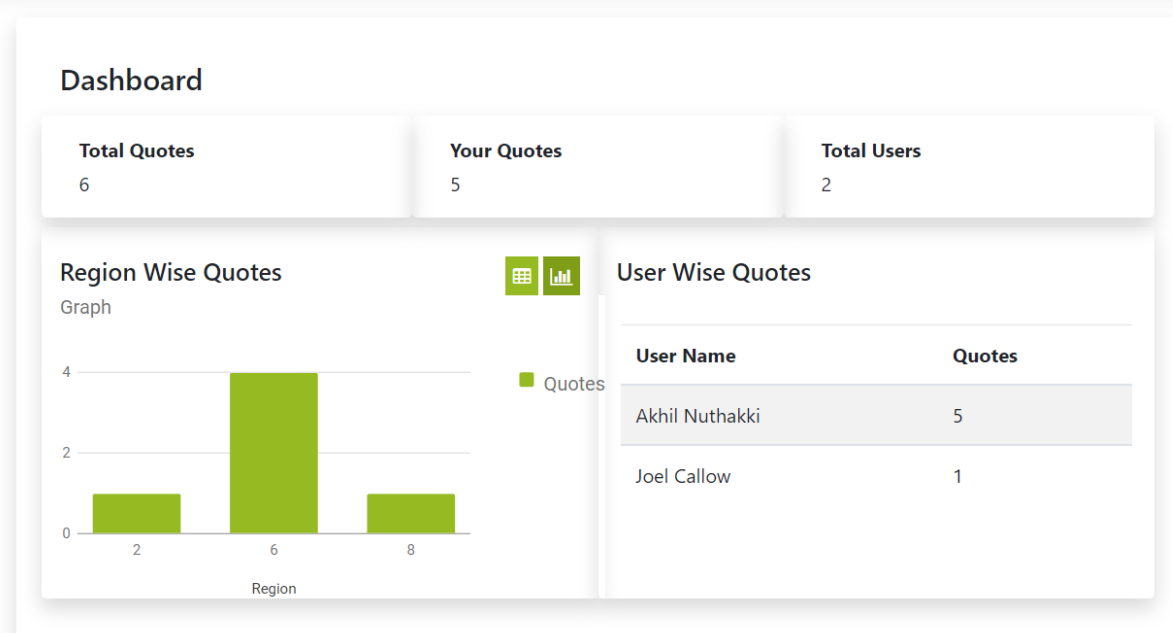
Quote Reference	Address	Quote Generated Date
2022/1/S-B-EV/AN	5 DANBURY PLACE LES OAZ	21/04/2022
2022/2/S-B/AN	5 DANBURY PLACE LES OBZ	20/04/2022
2022/3/S-B-EV/AN	5 DANBURY PLACE LES OAZ	27/04/2022

Index method uses *PagedList View Model* which is created specifically for this view from the reference [7]. It consists of Current Page Index, Total Pages, Boolean value of has Previous pages, Boolean value of has next pages and quotes list itself. The method as parameters of Page Index, and from the filter from such as Quote Reference, Customer Email, Customer Postcode. It checks if the any of the filter parameters is provided. If provided, algorithm filters the quotes with the parameters and return the paginated list of filtered data. If not provided, it returns the paginated list of the history of quotes in the order of most recent generated quotes. The URL end points for the paginated list is <https://localhost:44375/History/Index?pageNumber=2> and URL end point for the filtered search is <https://localhost:44375/History/Index?QuoteReference=&CustomerEmail=akhil.nutha kki1%40gmail.com&CustomerPostcode=>.

```
public async Task<IActionResult> Index(string QuoteReference, string CustomerEmail, string CustomerPostcode, int pageNumber)
{
    var quotes = from q in _context.Quotes select q;
    if (!string.IsNullOrEmpty(QuoteReference) || !string.IsNullOrEmpty(CustomerEmail) || !string.IsNullOrEmpty(CustomerPostcode))
    {
        if(!string.IsNullOrEmpty(QuoteReference))
        {
            QuoteReference = QuoteReference.Replace("/", "_");
        }
        ViewData["QuoteReference"] = QuoteReference;
        ViewData["CustomerEmail"] = CustomerEmail;
        ViewData["CustomerPostcode"] = CustomerPostcode;
        quotes = _context.Quotes.Where(quote => quote.QuoteReference.Contains(QuoteReference) || quote.Customer.EmailId.Contains(Cus
    }


    if (pageNumber < 1) pageNumber = 1;
    int pageSize = 10;
    return View(await PagedList<Quote>.CreateAsync(quotes.Include(quote => quote.Customer).AsNoTracking(), pageNumber, pageSize)
}
```

2. Dashboard (*HTTP Get*): The URL end point for this action is <https://localhost/History/Dashboard>. It returns the Dashboard View of the controller that contains the analytical presentation of data of quote history. It consists of total quotes generated so far using the application, total number of application users, total number of quotes generated by the account user, user-wise count of quotes, and zone-wise count of quotes. The user-wise count of quotes is a table of number of quotes generated by each user using the tool. The region-wise count of quotes is a table of number of quotes generated for each zone. The region wise quotes data is also shown in the bar graph. The bar graph is implemented by using Google Charts. This will help the administrators or admin of the application quickly analyse the data for improvement of business in certain zones.



Mapping Data Upload Desktop Application:

The essential requirement of the project is to consolidate the existing mapping data in spreadsheet and design the normalized data structure to use in the calculation of cost models and quote generation. The Postcode Region Mapping Data and Irradiance datasets which are available in the spreadsheets need to update into the data tables for the calculations and automations. This functionality is not included in the web application as this is a one-time activity performed initially when the database is setup. Both Postcode Region Mapping and Irradiance data doesn't need a timely update to its data.

 EE Data Upload Form

environmentalenergies

Mapping Data Upload Application

Browse and upload Postcode Region Mapping Excel

Browse

Upload

Browse and upload Irradiance Datasets Excel

Browse

Upload

This desktop application to upload the mapping data is a simple click-on desktop application that contains the functionality to browse the excel files and upload the contents of the data in their respective data tables. The format of the excel files plays an important role in the upload algorithm. The application uses Microsoft.Office.Interop.Excel to automate the data from the excel. The Application Object of assembly Microsoft.Office.Interop.Excel contains Workbook, Worksheets and Range of the data in the Worksheet to access the data. The data once retrieved from the excel sheet is inserted into data tables using EF Core.

```

Excel.Application OExcel = new Excel.Application();
Excel.Workbook OWorkbook = OExcel.Workbooks.Open(RegionMap_File_Input.Text);
try
{
    Excel.Worksheet OWorksheet = OWorkbook.Worksheets.Item[1];

    int rows = OWorksheet.UsedRange.Rows.Count;

    backgroundRegionMapUpload.ReportProgress(0, "fetching data from excel...");

    for (int i = 2; i <= rows; i++)
    {
        RegionMap map = new RegionMap();
        map.Postcode = OWorksheet.UsedRange.Cells[i, 1].Value.ToString();
        map.Region = OWorksheet.UsedRange.Cells[i, 2].Value.ToString();

        RegionMappings.Add(map);
    }

    OWorkbook.Close();
    OExcel.Quit();

    backgroundRegionMapUpload.ReportProgress(50, "uploading into database...");

    try
    {
        using (var _context = new EEDbContext())
        {
            _context.RegionMappings.AddRange(RegionMappings);
            _context.SaveChanges();
        }
    }
    catch (Exception exception)

```

7. Future Work

The scope of future work in this application is very high. Currently, the application has its own user authentication, quote generation for the provided inputs, history of generated quotes and analytical presentation of data of generated quotes. But there is high scope in future developments to make this full-scale digitalized business process. Please find the scope of future work in this application that can unleash the full potential of this application in the below points.

1. **User Authentication:** In the current version of the application, the user authentication and user registration are internal and in line with the application. But it can be replaced with Microsoft Identity provider. Environmental Energies Ltd is using Microsoft Services for their internal and external communication services and have their organizational licenses with the Microsoft. We can use Microsoft Identity provider to authenticate identity information rather than with internal app authentication. This reduces the effort of the user to login into application with additional username and password which needs to be remembered. By using Microsoft IDP, we can reduce inline application development efforts in dealing with forget password and reset password features.
2. **Dynamic Sun Path Diagram:** Sun path diagram in the current quote generation is a static image. Sun path diagram based on the location of site explains the shade factor which effects the Annual electricity generation of the solar system. The shade factor is provided as input percentage value from the user. But it could be improved to a dynamic image that user can click on shade parts of the site in the input form page over a dynamic vector image to calculate the shade factor and use the same dynamic image to update in the quote. The dynamic image provides a better understanding of the shade factor effecting the annual electricity generation.
3. **Complete Process of Quote:** The current process involves user providing the investment inputs and customer details to generate the quote and automatically send an email to the customer. And this process of the quote stops right there in the application. But generally, after customer receives the quote, they accept and approve the quote and will pay some deposit to the company to start the work. Once deposit payment is received, Environmental Energies will start the installation process and setup the whole system at the site. So, the entire process of the quote generation, acceptance of the quote by the customer, payment by the customer and updates of the installation process to the customer through an automatic email trigger whenever employees update the status can be included in the application to make this entire business process into a digitalized automated business process.

8. References

1. Existing spreadsheets and word documents provided by the Environmental Energies Ltd. (accessed 14/02/2022)
2. Get started with ASP.NET Core MVC - <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-6.0&tabs=visual-studio> (accessed 14/02/2022)
3. Learn about entity framework - <https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli> (accessed 03/03/2022]
4. Learn about word solutions - <https://docs.microsoft.com/en-us/visualstudio/vsto/word-solutions?view=vs-2022> (accessed 25/03/2022)
5. Learn about word object model overview - <https://docs.microsoft.com/en-us/visualstudio/vsto/word-object-model-overview?view=vs-2022> (accessed 25/03/2022)
6. Database Engine Tutorials - <https://docs.microsoft.com/en-us/sql/relational-databases/database-engine-tutorials?view=sql-server-ver15> (accessed 25/02/2022)
7. Sort, filter, page, group - <https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-6.0> (accessed 26/04/2022)
8. AES-256 Encryption CS class - <https://gist.github.com/mhingston/a47caa21298950abc4d8422d98b7437e> (accessed 15/03/2022)