Sync Smart Home API Docs

None

Table of contents

1. Sync Smart Home API Documentation

2. API Reference

2.1 `app.main`

This module initializes & configures the FastAPI application

It acts as a middleware between the front-end & database, handling: - API route inclusion - CORS middleware for front-end communication - Database initialization & application life-cycle management

2.1.1 `app.main.RootResponse`

Bases: `BaseModel`

Response model for the root endpoint

Attributes:

Name	Type	Description
message	str	Welcome message

> **Source code in `app/main.py`** ⌄
> ```
>  95    class RootResponse(BaseModel):
>  96        """
>  97        Response model for the root endpoint
>  98
>  99        Attributes:
> 100            message (str): Welcome message
> 101        """
> 102        message: str
> ```

2.1.2 `app.main.lifespan(app_context)` `async`

Defines application's lifespan event handler

- Initialization the database at startup
- Sets up application-wide state variables
- Logs startup & shutdown events

Parameters:

Name	Type	Description	Default
app_context	FastAPI	The FastAPI application instance	*required*

Yields:

Name	Type	Description
None		Allows FastAPI to manage application life-cycle

Raises:

Type	Description
Exception	If database initialization fails

> **Source code in** `app/main.py` ⌄

```
37   @asynccontextmanager
38   async def lifespan(app_context: FastAPI):
39       """
40       Defines application's lifespan event handler
41
42       - Initialization the database at startup
43       - Sets up application-wide state variables
44       - Logs startup & shutdown events
45
46       Args:
47           app_context (FastAPI): The FastAPI application instance
48
49       Yields:
50           None: Allows FastAPI to manage application life-cycle
51
52       Raises:
53           Exception: If database initialization fails
54       """
55       try:
56           await init_db()
57           app_context.state.custom_attribute = "value"    # Placeholder for future app-wide state
58           logger.info("Application startup complete.")
59           yield
60       except Exception as e:
61           logger.error("Failed to initialize the database: %e.")
62           raise e
63       finally:
64           logger.info("Application is shutting down.")
```

2.1.3 `app.main.read_root()`

Root endpoint

Returns:

Name	Type	Description
RootResponse	RootResponse	A welcome message for the API

> **Source code in** `app/main.py` ⌄

```
104   @app.get("/", response_model = RootResponse)
105   def read_root() -> RootResponse:
106       """
107       Root endpoint
108
109       Returns:
110           RootResponse: A welcome message for the API
111       """
112       return RootResponse(message = "Welcome to the Sync Smart Home.")
```

2.2 `app.routes.user_routes`

This module defines user-related API routes for registration, authentication & role-based access

It includes: - User registration with hashed password storage - User authentication & JWT token issuance - Admin dashboard access (restricted to users with the "admin" role)

2.2.1 `app.routes.user_routes.get_admin_dashboard()` async

Admin dashboard endpoint (restricted access)

Returns:

Name	Type	Description
dict		A welcome message confirming admin access

⁊Source code in `app/routes/user_routes.py` ⌄

```
 92   @router.get("/admin/dashboard", dependencies = [Depends(role_required("admin"))])
 93   async def get_admin_dashboard():
 94       """
 95       Admin dashboard  endpoint (restricted access)
 96
 97       Returns:
 98           dict: A welcome message confirming admin access
 99       """
100       return {"message": "Welcome, admin!"}
```

2.2.2 `app.routes.user_routes.login(form_data=Depends())` `async`

Authenticate user & return a JWT token

Parameters:

Name	Type	Description	Default
`form_data`	`OAuth2PasswordRequestForm`	User-provided login credentials	`Depends()`

Returns:

Name	Type	Description
`dict`		A dictionary containing the access token & token type

Raises:

Type	Description
`HTTPException(400)`	If the username or password is incorrect

⁊Source code in `app/routes/user_routes.py` ⌄

```
 67   @router.post("/token")
 68   async def login(form_data: OAuth2PasswordRequestForm = Depends()):
 69       """
 70       Authenticate user & return a JWT token
 71
 72       Args:
 73           form_data (OAuth2PasswordRequestForm): User-provided login credentials
 74
 75       Returns:
 76           dict: A dictionary containing the access token & token type
 77
 78       Raises:
 79           HTTPException (400): If the username or password is incorrect
 80       """
 81       user = users_collection.find_one({"email": form_data.username})
 82       if not user or not verify_password(form_data.password, user["password_hash"]):
 83           raise HTTPException(status_code = 400, detail = "Invalid username or password")
 84
 85       access_token = create_access_token(
 86           data = {"sub": user["email"], "role": user.get("role", "user")},
 87           expires_delta = timedelta(minutes = 30)
 88       )
 89
 90       return {"access_token": access_token, "token_type": "bearer"}
```

2.2.3 `app.routes.user_routes.register_user(user, background_tasks)` `async`

Register a new user in the database

Parameters:

Name	Type	Description	Default
user	UserCreate	User registration data including email, password & optional role	*required*

Returns:

Name	Type	Description
UserResponse		The created user information excluding password

Raises:

Type	Description
HTTPException(400)	If the email is already registered
HTTPException(500)	If there is an error inserted the user into the database

Source code in `app/routes/user_routes.py` ⌄

```python
@router.post("/register", response_model = UserResponse)
async def register_user(user: UserCreate, background_tasks: BackgroundTasks):
    """
    Register a new user in the database

    Args:
        user (UserCreate): User registration data including email, password & optional role

    Returns:
        UserResponse: The created user information excluding password

    Raises:
        HTTPException (400): If the email is already registered
        HTTPException (500): If there is an error inserted the user into the database
    """
    # Check if email exists already
    if users_collection.find_one({"email": user.email}):
        raise HTTPException(status_code = 400, detail = "Registration failed, please try again.")

    # Prepare user data for insertion
    user_data = {
        "email": user.email,
        "password_hash": hash_password(user.password),
        "is_verified": False,
        "created_at": datetime.now(timezone.utc),
        "updated_at": datetime.now(timezone.utc),
        "role": user.role or "user"
    }

    # Database user insert
    try:
        result = users_collection.insert_one(user_data)
    except Exception as exc:
        raise HTTPException(status_code = 500, detail = f"Failed to register user: {exc}") from exc

    # Generate verification token & schedule when verification email sends
    token = generate_verification_token(user.email)
    background_tasks.add_task(send_verification_email, user.email, token)

    return UserResponse(
        id = str(result.inserted_id),
        role = user_data["role"],
        email = user.email,
        is_verified = False,
        created_at = user_data["created_at"]
    )
```

2.3 `app.routes.data_routes`

This module defines API routes for managing & retrieving energy consumption data

It includes: - An endpoint for adding new energy data (admin-only access) - An endpoint for fetching aggregated energy consumption data with filtering options - An admin dashboard route (restricted to users with an "admin" role)

2.3.1 `app.routes.data_routes.add_energy_data(data)` `async`

Add new energy consumption data to the database (admin-only)

Parameters:

Name	Type	Description	Default
data	EnergyData	The energy data record to be added	*required*

Returns:

Name	Type	Description
dict		A confirmation message upon successfully insertion

Source code in `app/routes/data_routes.py`

```python
18   @router.post("/add", dependencies = [Depends(role_required("admin"))])
19   async def add_energy_data(data: EnergyData):
20       """
21       Add new energy consumption data to the database (admin-only)
22
23       Args:
24           data (EnergyData): The energy data record to be added
25
26       Returns:
27           dict: A confirmation message upon successfully insertion
28       """
29       energy_collection.insert_one(data.model_dump())      # Insert into MongoDB
30       return {"message": "Energy data added successfully"}
```

2.3.2 `app.routes.data_routes.get_admin_dashboard()` `async`

Admin dashboard endpoint (restricted access)

Returns:

Name	Type	Description
dict		A welcome message confirming admin access

Source code in `app/routes/data_routes.py`

```python
107   @router.get("/admin/dashboard", dependencies = [Depends(role_required("admin"))])
108   async def get_admin_dashboard():
109       """
110       Admin dashboard endpoint (restricted access)
111
112       Returns:
113           dict: A welcome message confirming admin access
114       """
115       return {"message": "Welcome, admin!"}
```

2.3.3

```
app.routes.data_routes.get_aggregated_data(start_date=Query(None, description='Start date (YYYY-MM-DD)'),
end_date=Query(None, description='End date (YYYY-MM-DD)'), device_id=Query(None, description='Device ID
filter'), location=Query(None, description='Location filter'), interval='day')  async
```

Retrieve aggregated energy consumption data based on time interval & filters

Parameters:

Name	Type	Description	Default
start_date	Optional[str]	The start date for filtering (YYYY-MM-DD)	Query(None, description='Start date (YYYY-MM-DD)')
end_date	Optional[str]	The end date for filtering (YYYY-MM-DD)	Query(None, description='End date (YYYY-MM-DD)')
device_id	Optional[str]	The ID of the device to filter data	Query(None, description='Device ID filter')
location	Optional[str]	The location filter	Query(None, description='Location filter')
interval	Literal['hour', 'day', 'week']	The time interval for aggregation (defaults to "day")	'day'

Returns:

Name	Type	Description
dict		Aggregated energy consumption data grouped by the selected interval

Source code in `app/routes/data_routes.py` ⌄

```python
32    @router.get("/aggregate")
33    async def get_aggregated_data(
34        start_date: Optional[str] = Query(None, description = "Start date (YYYY-MM-DD)"),
35        end_date: Optional[str] = Query(None, description = "End date (YYYY-MM-DD)"),
36        device_id: Optional[str] = Query(None, description = "Device ID filter"),
37        location: Optional[str] = Query(None, description = "Location filter"),
38        interval: Literal["hour", "day", "week"] = "day",
39    ):
40        """
41        Retrieve aggregated energy consumption data based on time interval & filters
42
43        Args:
44            start_date (Optional[str]): The start date for filtering (YYYY-MM-DD)
45            end_date (Optional[str]): The end date for filtering (YYYY-MM-DD)
46            device_id (Optional[str]): The ID of the device to filter data
47            location (Optional[str]): The location filter
48            interval (Literal["hour", "day", "week"]): The time interval for aggregation (defaults to "day")
49
50        Returns:
51            dict: Aggregated energy consumption data grouped by the selected interval
52        """
53        query = {}
54
55        if start_date and end_date:
56            query["timestamp"] = {
57                "$gte": datetime.strptime(start_date, "%Y-%m-%d"),
58                "$lte": datetime.strptime(end_date, "%Y-%m-%d")
59            }
60        if device_id:
61            query["device_id"] = device_id
62        if location:
63            query["location"] = location
64
65        time_group = {
66            "year": {"$year": "$timestamp"},
67            "month": {"$month": "$timestamp"},
68            "day": {"$dayOfMonth": "$timestamp"}
69        }
70
71        if interval == "hour":
72            time_group = {
73                "device_id": "$device_id",
74                "year": {"$year": "$timestamp"},
75                "month": {"$month": "$timestamp"},
76                "day": {"$dayOfMonth": "$timestamp"},
77                "hour": {"$hour": "$timestamp"}
78            }
79        elif interval == "day":
80            time_group = {
81                "device_id": "$device_id",
82                "year": {"$year": "$timestamp"},
83                "month": {"$month": "$timestamp"},
84                "day": {"$dayOfMonth": "$timestamp"}
85            }
86        elif interval == "week":
87            time_group = {
88                "device_id": "$device_id",
89                "year": {"$year": "$timestamp"},
90                "week": {"$isoWeek": "$timestamp"}
91            }
92
93        aggregation_pipeline = [
94            {"$match": query},
95            {"$group": {
96                "_id": {
97                    "device_id": "$device_id",   # Group by device
98                    **time_group                  # Group by time interval
99                }, "total_energy": {"$sum": "$energy_consumed"}
100           }}
101       ]
102
103       result = list(energy_collection.aggregate(aggregation_pipeline))
104
105       return {"aggregated_data": result}
```

2.4 `app.routes.report_routes`

This module provides API endpoints for generating energy consumption reports

It includes: - An endpoint to generate reports in CSV or PDF format, optionally filtered by a data range - Reports are stored in the `generated_reports` directory

2.4.1

```
app.routes.report_routes.generate_report(format=Query('csv', enum=['csv', 'pdf']), start_date=Query(None,
description='Start date (YYYY-MM-DD)'), end_date=Query(None, description='End date (YYYY-MM-DD)')) async
```

Generate an energy consumption report in CSV or PDF format

Parameters:

Name	Type	Description	Default
format	str	The desired report format, either "csv" or "pdf"	Query('csv', enum=['csv', 'pdf'])
start_date	str	The start date for filtering data (YYYY-MM-DD)	Query(None, description='Start date (YYYY-MM-DD)')
end_date	str	The end date for filtering data (YYYY-MM-DD)	Query(None, description='End date (YYYY-MM-DD)')

Returns:

Name	Type	Description
FileResponse		The generated report file

Raises:

Type	Description
HTTPException(400)	If an invalid date format is provided
HTTPException(404)	If no energy data is available for the selected range

Source code in `app/routes/report_routes.py` ⌄

```python
26    @router.post("/report", dependencies = [Depends(role_required("admin"))])
27    async def generate_report(
28            format: str = Query("csv", enum = ["csv", "pdf"]),
29            start_date: str = Query(None, description = "Start date (YYYY-MM-DD)"),
30            end_date: str = Query(None, description = "End date (YYYY-MM-DD)")
31    ):
32        """
33        Generate an energy consumption report in CSV or PDF format
34
35        Args:
36            format (str): The desired report format, either "csv" or "pdf"
37            start_date (str, optional): The start date for filtering data (YYYY-MM-DD)
38            end_date (str, optional): The end date for filtering data (YYYY-MM-DD)
39
40        Returns:
41            FileResponse: The generated report file
42
43        Raises:
44            HTTPException (400): If an invalid date format is provided
45            HTTPException (404): If no energy data is available for the selected range
46        """
47        try:
48            energy_data = get_energy_data(start_date, end_date)
49        except ValueError as exc:
50            raise HTTPException(status_code = 400, detail = str(exc)) from exc
51
52        if not energy_data:
53            raise HTTPException(status_code = 404, detail = "No energy data available for the selected range.")
54
55        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
56        filename = f"{REPORTS_DIR}/energy_report_{timestamp}.{format}"
57
58        if format == "csv":
59            # Convert data to a DataFrame & format timestamps
60            # df = pd.DataFrame(energy_data)
61            # df["timestamp"] = pd.to_datetime(df["timestamp"]).dt.strftime("%Y-%m-%d %H:%M:%S")  # Format timestamp
62            # df.columns = ["Device ID", "Timestamp", "Energy Consumed (kWh)", "Location"]
63            df = pd.DataFrame(energy_data)
64            df.drop(columns=["timestamps"], errors="ignore", inplace=True)  # Drop extra column
65            df.columns = ["Device ID", "Timestamp", "Energy Consumed (kWh)", "Location"]
66            df.to_csv(filename, index = False)
67        elif format == "pdf":
68            # Create a PDF report with a structure table
69            doc = SimpleDocTemplate(filename, pagesize=letter)
70            elements = []
71
72            # Prepare table data
73            data = [["Device ID", "Timestamp", "Energy Cosumed (kWh)", "Location"]]
74            for entry in energy_data:
75                data.append([
76                    str(entry.get("device_id", "N/A")),
77                    str(entry.get("timestamp", "")).strftime('%Y-%m-%d %H:%M:%S') if entry.get("timestamp") else "N/A",
78                    str(entry.get("energy_consumed", "N/A")),
79                    str(entry.get("location", "N/A"))
80                ])
81
82            # Create & style table
83            table = Table(data)
84            table.setStyle(TableStyle([
85                ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
86                ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
87                ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
88                ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
89                ('FONTSIZE', (0, 0), (-1, 0), 12),
90                ('BOTTOMPADDING', (0, 0), (-1, 0), 10),
91                ('GRID', (0, 0), (-1, -1), 1, colors.black),
92                ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.whitesmoke, colors.lightgrey])
93            ]))
94
95            elements.append(table)
96            doc.build(elements)
97
98        return FileResponse(
99            path  = os.path.abspath(filename),
100           filename = os.path.basename(filename),
101           media_type = "application/octet-stream"
102       )
```

2.5 `app.models.device`

This module defines the EnergySummary model for analyzing & reporting energy consumption

The EnergySummary model: - Aggregates energy consumption data over specified time periods - Supports reporting & historical data analysis requirements - Enables cost estimation based on energy usage - Facilitates comparisons between current & past usage periods - Provides the data foundation for energy-saving suggestions & insights

2.5.1 `app.models.device.Device`

Bases: `BaseModel`

Pydantic model representing a smart device within the home automation system

Tracks device information, current status, & location within the home

Used for device control, automation, & energy consumption tracking

Attributes:

Name	Type	Description
id	str	Unique identifier for the device
name	str	User-friendly name for the device
type	str	Category of device (e.g., "light", "thermostat", "appliance")
room_id	Optional[str]	ID of the room where the device is located
status	str	Current operational status ("on", "off", or custom states)
is_active	bool	Whether the device is currently available in the system

Source code in `app/models/device.py` ⌄

```python
14   class Device(BaseModel):
15       """
16           Pydantic model representing a smart device within the home automation system
17
18           Tracks device information, current status, & location within the home
19
20           Used for device control, automation, & energy consumption tracking
21
22           Attributes:
23               id (str): Unique identifier for the device
24               name (str): User-friendly name for the device
25               type (str): Category of device (e.g., "light", "thermostat", "appliance")
26               room_id (Optional[str]): ID of the room where the device is located
27               status (str): Current operational status ("on", "off", or custom states)
28               is_active (bool): Whether the device is currently available in the system
29       """
30       id: str
31       name: str
32       type: str
33       room_id: Optional[str]
34       status: str              # "on"/"off"/custom states
35       is_active: bool = True
```

2.6 `app.models.energy_data`

This module defines the EnergyData model used for storing energy consumption records

The model: - Represents energy consumption data for a specific device - Includes metadata such as timestamp & location

2.6.1 `app.models.energy_data.EnergyData`

Bases: `BaseModel`

Pydantic model representing energy consumption data for a smart device

Attributes:

Name	Type	Description
device_id	str	Unique identifier for the device
timestamp	datetime	The date & time of the recorded energy consumption
energy_consumed	float	The amount of energy consumed in kilowatt-hours (kWh)
location	Optional[str]	The optional physical location of the device

Source code in `app/models/energy_data.py` ⌄

```python
12    class EnergyData(BaseModel):
13        """
14        Pydantic model representing energy consumption data for a smart device
15
16        Attributes:
17            device_id (str): Unique identifier for the device
18            timestamp (datetime): The date & time of the recorded energy consumption
19            energy_consumed (float): The amount of energy consumed in kilowatt-hours (kWh)
20            location (Optional[str]): The optional physical location of the device
21        """
22        device_id: str
23        timestamp: datetime
24        energy_consumed: float        # tracking in kWh (kilowatt hours)
25        location: Optional[str] = None
```

2.7 `app.models.energy_summary`

This module defines the EnergySummary model for analyzing & reporting energy consumption

The EnergySummary model: - Aggregates energy consumption data over specified time periods - Supports reporting & historical data analysis requirements - Enables cost estimation based on energy usage - Facilitates comparisons between current & past usage periods - Provides the data foundation for energy-saving suggestions & insights

2.7.1 `app.models.energy_summary.EnergySummary`

Bases: `BaseModel`

Pydantic model representing aggregated energy usage data for reporting

Provides summarized energy consumption data for specific time periods, enabling users to track trends & compare usage over time

Attributes:

Name	Type	Description
user_id	str	ID of the user this summary belongs to
period	str	Time period of the summary ("daily", "weekly", "monthly")
start_date	datetime	Beginning of the summary period
end_date	datetime	End of the summary period
total_consumption	float	Total energy consumed in kWh during the period
cost_estimate	Optional[float]	Estimated cost of energy used
comparison_to_previous	Optional[float]	Percentage change compared to previous period

> **Source code in** `app/models/energy_summary.py` ⌄

```
15  class EnergySummary(BaseModel):
16      """
17      Pydantic model representing aggregated energy usage data for reporting
18
19      Provides summarized energy consumption data for specific time periods,
20          enabling users to track trends & compare usage over time
21
22      Attributes:
23          user_id (str): ID of the user this summary belongs to
24          period (str): Time period of the summary ("daily", "weekly", "monthly")
25          start_date (datetime): Beginning of the summary period
26          end_date (datetime): End of the summary period
27          total_consumption (float): Total energy consumed in kWh during the period
28          cost_estimate (Optional[float]): Estimated cost of energy used
29          comparison_to_previous (Optional[float]): Percentage change compared to previous period
30      """
31      user_id: str
32      period: str      # daily, weekly, monthly
33      start_date: datetime
34      end_date: datetime
35      total_consumption: float
36      cost_estimate: Optional[float]
37      comparison_to_previous: Optional[float]      # percentage change
```

2.8 `app.models.profile`

This module defines the Profile model for managing user profiles in the Smart Family Energy Manager

The Profile model: - Represents different user types within the system (adult/elderly, child) - Stores accessibility preferences & personalization settings - Manages access control for features like device control & energy data - Supports the admin's ability to create & manage multiple profiles per household

2.8.1 `app.models.profile.Profile`

Bases: `BaseModel`

Pydantic model representing a user profile in the Smart Family Energy Manager system

Profiles are created by an admin & can be customized for different family member with varying levels of access to system features & data

Attributes:

Name	Type	Description
user_id	str	Unique identifier linking this profile to a user account
name	str	The display name for this profile
age	Optional[int]	The age of the profile user, used to tailor experience
profile_type	str	Type of profile - "adult", "elderly", or "child"
accessibility_settings	Dict	Dictionary containing accessibility preferences
can_control_devices	bool	Whether this profile can control smart devices
can_access_energy_data	bool	Whether this profile can view energy consumption data
can_manage_notifications	bool	Whether this profile can manage notification settings

Source code in `app/models/profile.py` ⌄

```
13    class Profile(BaseModel):
14        '''
15        Pydantic model representing a user profile in the Smart Family Energy Manager system
16
17        Profiles are created by an admin & can be customized for different family member
18            with varying levels of access to system features & data
19
20        Attributes:
21            user_id (str): Unique identifier linking this profile to a user account
22            name (str): The display name for this profile
23            age (Optional[int]): The age of the profile user, used to tailor experience
24            profile_type (str): Type of profile - "adult", "elderly", or "child"
25            accessibility_settings (Dict): Dictionary containing accessibility preferences
26            can_control_devices (bool): Whether this profile can control smart devices
27            can_access_energy_data (bool): Whether this profile can view energy consumption data
28            can_manage_notifications (bool): Whether this profile can manage notification settings
29        '''
30        user_id: str
31        name: str
32        age: Optional[str]
33        profile_type: str       # "adult"/"elderly"/"child"
34        accessibility_settings: dict = {}
35        can_control_devices: bool = False
36        can_access_energy_data: bool = False
37        can_manage_notifications: bool = False
```

2.9 `app.models.room`

This module defines the Room model for organizing smart devices by location within the home

The Room model: - Provides a way to group devices by physical location - Supports the add/delete room functionality requirement - Enables room-based energy consumption analytics - Helps users organize & manage their smart home more efficiently - Facilitates location-specific automation rules & schedules

2.9.1 `app.models.room.Room`

Bases: `BaseModel`

Pydantic model representing a room or area within the smart home

Rooms provide organizational structure for devices & help contextualize energy usage data by location

Attributes:

Name	Type	Description
id	str	Unique identifier for the room
name	str	User-friendly name for the room (e.g., "Living Room", "Kitchen")
created_by	str	ID of the user who created this room

Source code in `app/models/room.py` ⌄

```
13    class Room(BaseModel):
14        """
15        Pydantic model representing a room or area within the smart home
16
17        Rooms provide organizational structure for devices & help contextualize
18            energy usage data by location
19
20        Attributes:
21            id (str): Unique identifier for the room
22            name (str): User-friendly name for the room (e.g., "Living Room", "Kitchen")
23            created_by (str): ID of the user who created this room
24        """
25        id: str
26        name: str
27        created_by: str     # user_id
```

2.10 `app.models.schedule`

This module defines the Schedule model for automating device operations within the smart home

The Schedule model: - Enables users to set timed operations for connected devices - Supports the scheduling & automation functional requirements - Allows for energy optimization through scheduled device usage - Provides time-based control of devices for convenience & efficiency - Implements the core functionality of device automation based on user-defined schedules

2.10.1 `app.models.schedule.Schedule`

Bases: `BaseModel`

Pydantic model representing an automated schedule for a smart device

Allows users to create automated routines for devices to operate at specified times, supporting energy efficiency & convenience

Attributes:

Name	Type	Description
device_id	str	ID of the device to be controlled by this schedule
start_time	datetime	Time when the scheduled operation should begin
end_time	datetime	Time when the scheduled operation should end
start_date	datetime	Date when the schedule becomes active
end_date	datetime	Date when the schedule expires
created_by	str	ID of the user who created this schedule
is_active	bool	Whether this schedule is currently enabled

Source code in `app/models/schedule.py`

```
14   class Schedule(BaseModel):
15       """
16       Pydantic model representing an automated schedule for a smart device
17
18       Allows users to create automated routines for devices to operate at
19           specified times, supporting energy efficiency & convenience
20
21       Attributes:
22           device_id (str): ID of the device to be controlled by this schedule
23           start_time (datetime): Time when the scheduled operation should begin
24           end_time (datetime): Time when the scheduled operation should end
25           start_date (datetime): Date when the schedule becomes active
26           end_date (datetime): Date when the schedule expires
27           created_by (str): ID of the user who created this schedule
28           is_active (bool): Whether this schedule is currently enabled
29       """
30       device_id: str
31       start_time: datetime
32       end_time: datetime
33       start_date: datetime
34       end_date: datetime
35       created_by: str        # user_id
36       is_active: bool = True
```

2.11 `app.models.user`

This model defines user-related data models for account creation & response handling

It includes: - `UserCreate` : A model for user registration input with email & password validation - `UserResponse` : A model for returning user details in API responses

2.11.1 `app.models.user.UserCreate`

Bases: `BaseModel`

Pydantic model for user registration input

Attributes:

Name	Type	Description
email	EmailStr	The user's email address
password	str	The user's password (validated for strength)
role	Optional[str]	The user's role, defaulting to "user"

> **Source code in** `app/models/user.py` ⌄

```python
12    class UserCreate(BaseModel):
13        '''
14        Pydantic model for user registration input
15
16        Attributes:
17            email (EmailStr): The user's email address
18            password (str): The user's password (validated for strength)
19            role (Optional[str]): The user's role, defaulting to "user"
20        '''
21        email: EmailStr
22        password: str
23        role: Optional[str] = "user"    # Default role
24
25        @field_validator("password")
26        @classmethod
27        def validate_password(cls, value: str) -> str:
28            """
29            Validate the password to ensure it meets security requirements
30
31            Args:
32                value (str): The password provided by the user
33
34            Returns:
35                str: The validated password
36
37            Raises:
38                ValueError: If the password doesn't meet security requirements
39            """
40            if len(value) < 8:
41                raise ValueError("Password must be at least 8 characters long.")
42            if not any(char.isdigit() for char in value):
43                raise ValueError("Password must contain at least 1 number.")
44            if not any(char.isalpha() for char in value):
45                raise ValueError("Password must contain at least 1 letter.")
46            if not any(char.islower() for char in value):
47                raise ValueError("Password must contain at least 1 lower letter.")
48            if not any(char.isupper() for char in value):
49                raise ValueError("Password must contain at least 1 upper letter.")
50            if not any(char in '!@#$%^&*()_+-=[]{}|;\':",.<>?/' for char in value):
51                raise ValueError("Password must contain at least 1 special character.")
52            return value
53
54        @field_validator("email")
55        @classmethod
56        def validate_email(cls, value: EmailStr) -> EmailStr:
57            """
58            Validate the user's email address
59
60            Args:
61                value (EmailStr): The email address provided
62
63            Returns:
64                EmailStr: The validated email
65
66            Raises:
67                ValueError: If the email doesn't contain `@` or a valid domain
68            """
69            if not "@" in value:
70                raise ValueError("Email must contain @.")
71            if not "." in value.split("@")[1]:
72                raise ValueError("Email must contain a dot.")
73            return value
```

`app.models.user.UserCreate.validate_email(value)` `classmethod`

Validate the user's email address

Parameters:

Name	Type	Description	Default
value	EmailStr	The email address provided	*required*

Returns:

Name	Type	Description
EmailStr	EmailStr	The validated email

Raises:

Type	Description
ValueError	If the email doesn't contain @ or a valid domain

Source code in `app/models/user.py` ⌄

```python
54   @field_validator("email")
55   @classmethod
56   def validate_email(cls, value: EmailStr) -> EmailStr:
57       """
58       Validate the user's email address
59
60       Args:
61           value (EmailStr): The email address provided
62
63       Returns:
64           EmailStr: The validated email
65
66       Raises:
67           ValueError: If the email doesn't contain `@` or a valid domain
68       """
69       if not "@" in value:
70           raise ValueError("Email must contain @.")
71       if not "." in value.split("@")[1]:
72           raise ValueError("Email must contain a dot.")
73       return value
```

app.models.user.UserCreate.validate_password(value) classmethod

Validate the password to ensure it meets security requirements

Parameters:

Name	Type	Description	Default
value	str	The password provided by the user	*required*

Returns:

Name	Type	Description
str	str	The validated password

Raises:

Type	Description
ValueError	If the password doesn't meet security requirements

> **Source code in** `app/models/user.py` ⌄

```
25    @field_validator("password")
26    @classmethod
27    def validate_password(cls, value: str) -> str:
28        """
29        Validate the password to ensure it meets security requirements
30
31        Args:
32            value (str): The password provided by the user
33
34        Returns:
35            str: The validated password
36
37        Raises:
38            ValueError: If the password doesn't meet security requirements
39        """
40        if len(value) < 8:
41            raise ValueError("Password must be at least 8 characters long.")
42        if not any(char.isdigit() for char in value):
43            raise ValueError("Password must contain at least 1 number.")
44        if not any(char.isalpha() for char in value):
45            raise ValueError("Password must contain at least 1 letter.")
46        if not any(char.islower() for char in value):
47            raise ValueError("Password must contain at least 1 lower letter.")
48        if not any(char.isupper() for char in value):
49            raise ValueError("Password must contain at least 1 upper letter.")
50        if not any(char in '!@#$%^&*()_+-=[]{}|;\':",.<>?/' for char in value):
51            raise ValueError("Password must contain at least 1 special character.")
52        return value
```

2.11.2 `app.models.user.UserResponse`

Bases: `BaseModel`

Pydantic model for user response data

Attributes:

Name	Type	Description
id	str	Unique identifier for the user
role	str	The assigned role of the user
email	EmailStr	The user's email address
is_verified	bool	Indicates whether the user's email address is verified (defaults to False)
created_at	Optional[datetime]	Timestamp of the user account creation

> **Source code in** `app/models/user.py` ⌄

```
75    class UserResponse(BaseModel):
76        """
77        Pydantic model for user response data
78
79        Attributes:
80            id (str): Unique identifier for the user
81            role (str): The assigned role of the user
82            email (EmailStr): The user's email address
83            is_verified (bool): Indicates whether the user's email address is verified (defaults to False)
84            created_at (Optional[datetime]): Timestamp of the user account creation
85        """
86        id: str
87        role: str
88        email: EmailStr
89        is_verified: bool = False
90        created_at: Optional[datetime]
91
92        model_config = ConfigDict(from_attributes = True)
```

2.12 `app.db.database`

This module handles MongoDB database connections & operations

It provides: - Initialization of MongoDB collections & indexes - Functions to fetch energy consumption data with optional filtering

2.12.1 `app.db.database.get_energy_data(start_date=None, end_date=None)`

Fetch energy data consumption from MongoDB with optional date filtering

Parameters:

Name	Type	Description	Default
start_date	str	Start date in YYYY-MM-DD format	None
end_date	str	End date in YYYY-MM-DD format	None

Returns:

Type	Description
List[Dict]	List[Dict]: A list of energy consumption records

Raises:

Type	Description
ValueError	If the provided date format is incorrect

Source code in `app/db/database.py` ⌄

```python
88    def get_energy_data(start_date: Optional[str] = None, end_date: Optional[str] = None) -> List[Dict]:
89        """
90        Fetch energy data consumption from MongoDB with optional date filtering
91
92        Args:
93            start_date (str, optional): Start date in `YYYY-MM-DD` format
94            end_date (str, optional): End date in `YYYY-MM-DD` format
95
96        Returns:
97            List[Dict]: A list of energy consumption records
98
99        Raises:
100            ValueError: If the provided date format is incorrect
101        """
102        query = {}
103
104        if start_date and end_date:
105            try:
106                start_dt = datetime.strptime(start_date, "%Y-%m-%d")
107                end_dt = datetime.strptime(end_date, "%Y-%m-%d")
108                query["timestamp"] = {"$gte": start_dt, "$lte": end_dt}
109            except ValueError as exc:
110                raise ValueError("Invalid date format. Use `YYYY-MM-DD`.") from exc
111
112        energy_data = list(energy_collection.find(query, {"_id": 0}))   # Exclude MongoDB _id field
113
114        return energy_data
```

2.12.2 `app.db.database.init_db()` `async`

Initialize the database by creating necessary indexes

This function ensures: - Unique email constraint for users - Indexes on `device_id` & `timestamp` in the energy collection for optimized queries

Raises:

Type	Description
`RuntimeError`	If there is an error during database initialization

❝ Source code in `app/db/database.py` ⌄

```python
38    async def init_db():
39        """
40        Initialize the database by creating necessary indexes
41
42        This function ensures:
43        - Unique email constraint for users
44        - Indexes on `device_id` & `timestamp` in the energy collection for optimized queries
45
46        Raises:
47            RuntimeError: If there is an error during database initialization
48        """
49        try:
50            # User collection indexes
51            users_collection.create_index("email", unique = True)
52
53            # Energy data collection indexes
54            energy_collection.create_index("device_id")
55            energy_collection.create_index("timestamp")
56            energy_collection.create_index([("device_id", 1), ("timestamp", 1)])
57            energy_collection.create_index("location")
58
59            # Device collection indexes
60            devices_collection.create_index("id", unique=True)
61            devices_collection.create_index("room_id")
62            devices_collection.create_index("type")
63
64            # Profile collection indexes
65            profiles_collection.create_index("user_id")
66            profiles_collection.create_index([("user_id", 1), ("name", 1)], unique=True)
67
68            # Room collection indexes
69            rooms_collection.create_index("id", unique=True)
70            rooms_collection.create_index("created_by")
71
72            # Schedule collection indexes
73            schedules_collection.create_index("device_id")
74            schedules_collection.create_index("created_by")
75            schedules_collection.create_index("start_date")
76            schedules_collection.create_index("end_date")
77
78            # Energy summary collection indexes
79            summary_collection.create_index("user_id")
80            summary_collection.create_index([("user_id", 1), ("period", 1), ("start_date", 1)], unique=True)
81
82            logger.info("Database initialized successfully with all collections and indexes.")
83
84            print("Database initialized successfully.")
85        except Exception as e:
86            raise RuntimeError(f"Error during database initialization: {e}") from e
```

2.13 `app.core.config`

2.14 `app.core.security`

This module implements security features

Features include: - Password hashing - JWT token creation & verification - Role-based access control

2.14.1 `app.core.security.create_access_token(data, expires_delta=None)`

Create a JWT token with an expiration time

Parameters:

Name	Type	Description	Default
`data`	`dict`	The payload to encode in the token	*required*
`expires_delta`	`Optional[timedelta]`	The expiration time delta Defaults to `ACCESS_TOKEN_EXPIRE_MINUTES`	`None`

Returns:

Name	Type	Description
`str`		The encoded JWT token

Source code in `app/core/security.py` ⌄

```python
73    def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
74        """
75        Create a JWT token with an expiration time
76
77        Args:
78            data (dict): The payload to encode in the token
79            expires_delta (Optional[timedelta], optional): The expiration time delta
80                Defaults to `ACCESS_TOKEN_EXPIRE_MINUTES`
81
82        Returns:
83            str: The encoded JWT token
84        """
85        to_encode = data.copy()
86        expire = datetime.now(timezone.utc) + (expires_delta or timedelta(minutes = ACCESS_TOKEN_EXPIRE_MINUTES))
87        to_encode.update({"exp": expire})
88        return jwt.encode(to_encode, SECRET_KEY, algorithm = ALGORITHM)
```

2.14.2 `app.core.security.get_current_user(token=Depends(oauth2_scheme))`

Extract & validate the current user's JWT token

Parameters:

Name	Type	Description	Default
`token`	`str`	The OAuth2 token obtained from authentication	`Depends(oauth2_scheme)`

Returns:

Name	Type	Description
`dict`	`dict`	The decoded token payload

Raises:

Type	Description
`HTTPException`	If the token is invalid or missing required fields

```
109   def get_current_user(token: str = Depends(oauth2_scheme)) -> dict:
110       """
111       Extract & validate the current user's JWT token
112
113       Args:
114           token (str): The OAuth2 token obtained from authentication
115
116       Returns:
117           dict: The decoded token payload
118
119       Raises:
120           HTTPException: If the token is invalid or missing required fields
121       """
122       payload = verify_access_token(token)
123       if not payload or "role" not in payload:
124           raise HTTPException(status_code = status.HTTP_401_UNAUTHORIZED, detail = "Invalid token")
125       return payload
```

2.14.3 `app.core.security.hash_password(password)`

Hash a plain-text password using bcrypt

Parameters:

Name	Type	Description	Default
password	str	The password to hash	*required*

Returns:

Name	Type	Description
str	str	The hashed password

Raises:

Type	Description
ValueError	If an error occurs during hashing

Source code in app/core/security.py ⌄

```
24   def hash_password(password: str) -> str:
25       """
26       Hash a plain-text password using bcrypt
27
28       Args:
29           password (str): The password to hash
30
31       Returns:
32           str: The hashed password
33
34       Raises:
35           ValueError: If an error occurs during hashing
36       """
37       try:
38           return pwd_context.hash(password)
39       except Exception as e:
40           raise ValueError(f"Error hashing password: {e}") from e
```

2.14.4 `app.core.security.needs_rehash(hashed_password)`

Check if a stored hashed password required rehashing

Parameters:

Name	Type	Description	Default
hashed_password	str	The existing hashed password	*required*

Returns:

Name	Type	Description
bool	bool	True if rehashing is required & False if otherwise

Source code in `app/core/security.py` ⌄

```python
61    def needs_rehash(hashed_password: str) -> bool:
62        """
63        Check if a stored hashed password required rehashing
64
65        Args:
66            hashed_password (str): The existing hashed password
67
68        Returns:
69            bool: True if rehashing is required & False if otherwise
70        """
71        return pwd_context.needs_update(hashed_password)
```

2.14.5 `app.core.security.role_required(required_role)`

Dependency function to check to enforce role-based access control

Parameters:

Name	Type	Description	Default
required_role	str	The required user role	*required*

Returns:

Name	Type	Description
function		A dependency function that verifies the user's role

Raises:

Type	Description
HTTPException	If the user lacks the required role

```
127    def role_required(required_role: str):
128        """
129        Dependency function to check to enforce role-based access control
130
131        Args:
132            required_role (str): The required user role
133
134        Returns:
135            function: A dependency function that verifies the user's role
136
137        Raises:
138            HTTPException: If the user lacks the required role
139        """
140        def role_checker(current_user: dict = Depends(get_current_user)):
141            if current_user.get("role") != required_role:
142                raise HTTPException(
143                    status_code = status.HTTP_403_FORBIDDEN,
144                    detail = f"Permission denied: {required_role} role required"
145                )
146            return current_user
147        return role_checker
```

2.14.6 `app.core.security.verify_access_token(token)`

Verify & decode a JWT access token

Parameters:

Name	Type	Description	Default
token	str	The JWT token to verify	*required*

Returns:

Type	Description
Optional[dict]	Optional[dict]: The decoded token payload if valid, otherwise None

```
90     def verify_access_token(token: str) -> Optional[dict]:
91         """
92         Verify & decode a JWT access token
93
94         Args:
95             token (str): The JWT token to verify
96
97         Returns:
98             Optional[dict]: The decoded token payload if valid, otherwise None
99         """
100        try:
101            payload = jwt.decode(token, SECRET_KEY, algorithms = [ALGORITHM])
102            return payload
103        except JWTError:
104            return None
```

2.14.7 `app.core.security.verify_password(plain_password, hashed_password)`

Verify a plain-text password against a hashed password

Parameters:

Name	Type	Description	Default
plain_password	str	The input password	*required*
hashed_password	str	The stored hashed password	*required*

Returns:

Name	Type	Description
bool	bool	True if the password matches & False if otherwise

Raises:

Type	Description
ValueError	If an error occurs during verification

Source code in `app/core/security.py` ⌄

```
42   def verify_password(plain_password: str, hashed_password: str) -> bool:
43       """
44       Verify a plain-text password against a hashed password
45
46       Args:
47           plain_password (str): The input password
48           hashed_password (str): The stored hashed password
49
50       Returns:
51           bool: True if the password matches & False if otherwise
52
53       Raises:
54           ValueError: If an error occurs during verification
55       """
56       try:
57           return pwd_context.verify(plain_password, hashed_password)
58       except Exception as e:
59           raise ValueError(f"Error verifying password: {e}") from e
```