PROGETTO CEA RESUME

MODELING(TUTTI INSIEME):

Classe User –

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;
@Column(nullable = false)
private String name;
@Column(nullable = false)
private String surname;
@Column(nullable = false, unique = true)
private String username;
@Column(nullable = false)
private String password;
```

Classe Administrator extends User

```
public class Administrator extends User{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @OneToMany(mappedBy = "administrator")
    private List<Secretary> secretaries;
    @OneToMany(mappedBy = "administrator")
    private List<Condominium> condominiums;
    @Column(nullable = false)
    private Role role = Role.ADMIN;
    private boolean isAvailable = true;
```

Classe Customer extends User

```
public class Customer extends User{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false, unique = true)
    private String taxCode;
    @OneToMany(mappedBy = "customer")
    private List<Apartment> apartments;
    @Column(nullable = false)
    private Role role = Role.CUSTOMER;
    private boolean isAvailable = true;
```

Classe Secretary extends User

```
public class Secretary extends User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private Role role = Role.SECRETARY;
    private boolean isAvailable = true;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    private Administrator administrator;
    @OneToMany(mappedBy = "secretary")
    private List<Intervention> intervention;
```

CLASSE Technician extends User

```
public class Technician extends User{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private static int max_intervention_for_technician;
    @Column(nullable = false)
    private Role role = Role.TECHNICIAN;
    private boolean isAvailable = true;

    @OneToMany(mappedBy = "technician")
    private List<Intervention> interventions;
```

CLASSE Apartment

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;
@Column(name = "unit_number", nullable = false)
private int unitNumber;
@Column(name = "floor_number", nullable = false)
private int floorNumber;
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn
private Customer customer;
@OneToOne(fetch = FetchType.LAZY)
private Scan meter;
@ManyToOne(fetch = FetchType.LAZY)
private Condominium condominium;
@OneToMany(mappedBy = "apartment")
private List<Intervention> interventions;
@Column(nullable = false)
private boolean isAvailable = true;
}
```

CLASSE Condominium

```
public class Condominium {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private String address;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn
    private Administrator administrator;
    @OneToMany(mappedBy = "condominium")
    private List<Apartment> apartments;
```

```
    @Column(nullable = false)
    private boolean isAvailable = true;
}
```

CLASSE Bill

```
public class Bill {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private double cost;
    @Column(nullable = false)
    private LocalDate paymentDay;
    @Column(nullable = false)
    private LocalDate DeliveringDay;
    @Column(nullable = false)
    private boolean isAvailable = true;
    @ManyToOne(fetch = FetchType.LAZY)
    private Scan meter;
}
```

CLASSE Intervention

```
public class Intervention {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private LocalDate interventionDate;
    @Column(nullable = false)
    private boolean isAvailable;
    @Column(nullable = false)
    private TypeOfIntervention type;
    @Column(nullable = false)
    private StatusIntervention status;

    @ManyToOne(fetch = FetchType.LAZY)
    private Technician technician;
    @ManyToOne(fetch = FetchType.LAZY)
    private Apartment apartment;
    @ManyToOne(fetch = FetchType.LAZY,  cascade = {CascadeType.PERSIST,
CascadeType.MERGE})
    Secretary secretary;
}
```

CLASSE Scan

```
public class Scan {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(nullable = false)
    private double mcLiter = 0;
    @Column(nullable = false)
    private boolean isAvailable = true;
```

```
    @OneToOne(mappedBy = "meter")
    private Apartment apartment;
    @OneToMany(mappedBy = "meter")
    private List<Bill> bills;

}
```

ENUM StatusIntervention

```
public enum StatusIntervention {

    COMPLETED("COMPLETED"),
    POSTPONED("POSTPONED"),
    PENDING("PENDING"),
    ACCEPTED("ACCEPTED"),
    CANCEL("CANCEL");

    private String status;

    private StatusIntervention(String status) {
       this.status = status;
    }

    public String getStatus() {
       return status;
    }


}
```

ENUM TypeOfIntervention

```
public enum TypeOfIntervention {

    METER_READING("METER READING"),
    FIXING_UP("FIXING UP");


    private String type;

    private TypeOfIntervention(String type) {
       this.type = type;
    }

    public String getType() {
       return type;
    }



}
```

ENUM Role

```
public enum Role {
    ADMIN("ADMIN"),
    CUSTOMER("CUSTOMER"),
    SECRETARY("SECRETARY"),
    TECHNICIAN("TECHNICIAN");

    @Getter
    private String role;
}
```

CONTROLLER

AllController(da fare la login e da assegnare ad un team)

```
@RestController
@RequestMapping("/all")
public class AllController {

    @Autowired
    private AllFacade allFacade;

    public ResponseEntity<LoginDTOResponse> login(@Valid @RequestBody
LoginDTORequest request){
        return null; //TODO
    }

    @PostMapping("/register/user")
    public ResponseEntity<String> registerAccount(@Valid @RequestBody
RegisterUserDto request){
        return
ResponseEntity.status(HttpStatus.ACCEPTED).body(allFacade.registerUser(request))
;
    }



}
```

CustomerController(Mark,Michele,Gianmarco)

```
@RestController
@RequestMapping("/customer")
public class CustomerController {
    @Autowired
    CustomerFacade customerFacade;
}
```

-può richiedere un intervento(metodo che prende in ingresso un id di segretaria e ritorna un dto di intervento)

-può controllare lo stato di un suo intervento(metodo che prende in ingresso un id di intervento e restituisce un dto di intervento)

-può eseguire l'autolettura del suo contatore(metodo void che restituisce una lettura aggiornata del suo contatore)

-può richiedere sia una lista di bollette che una bolletta singola tramite il suo id

SecretaryController(Alberto,Matteo,Emanuele,Simone)

```
@RestController
@RequestMapping("/secretary")
public class SecretaryController {

    @Autowired
    SecretaryFacade secFac;

    @GetMapping("/getAllBillsOfCondominium/{idCondominium}")
    public ResponseEntity<List<BillDTOResponse>>
getAllBillsOfCondominium(@PathVariable long idCondominium){
       return
ResponseEntity.status(HttpStatus.OK).body(secFac.getAllBillsOfCondominium(idCond
ominium));
    }//RITORNA UNA LISTA DI TUTTE LE BOLLETTE DEL CONDOMINIO

    @GetMapping("/getInterventionListPerType/{interv}")
    public ResponseEntity<List<InterventionDTOResponse>>
getInterventionListPerType(@PathVariable TypeOfIntervention interv){
       return
ResponseEntity.status(HttpStatus.OK).body(secFac.getInterventionListPerType(inte
rv));//RITORNA UNA LISTA DI INTERVENTI DIVISI PER TIPO
    }

    @GetMapping("/getScans")
    public ResponseEntity<List<ScanDTOResponse>> getScans(){
       return ResponseEntity.status(HttpStatus.OK).body(secFac.getScans());
    } //DTO CON CONSUMO IN LITRI DEL CONDOMINIO, INDIRIZZO DEL CONDOMINIO
       //ED UNA LISTA DI TUTTE LE BOLLETTE DI TUTTI GLI INQUILINI DEL
CONDOMINIO

}
```

-può modificare il limite degli interventi giornalieri dei tecnici

-può assegnare un intervento ad un tecnico(cambiare il loro stato)

-può modificare l’ intervento.

-

TechnicianController(Alessio,Massimo,Nicolo,Luca)

-può completare o rinviare un intervento

-può eseguire la lettura di tutti i contatori singoli del condominio

-Può vedere la lista degli interventi che deve effettuare

-può vedere la lista dei condomini associati agli interventi da effettuare

--

AdministratorController(Valerio,Gabriele,Edoardo)

-può splittare le bollette condominiali in bollette dei singoli utenti

-può vedere la lista dei condomini

-può associare e rimuovere customer agli appartamenti-(modificare gli appartamenti)

-può rimuovere sia tecnici, che customer, che segretarie

--

EXTRAS.

-Exception Handler(thanks mark)

```
@RestControllerAdvice
public class ExceptionHandlerCustom {
    @ExceptionHandler(NotValidDataException.class)
    public ResponseEntity<MessageDto> handleInvalidData(NotValidDataException
e){
        MessageDto m = new MessageDto(e.getMessage(),
HttpStatus.BAD_REQUEST.value(), e.getOggetto(),LocalDateTime.now());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(m);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<MessageDto>
```

```
validationError(MethodArgumentNotValidException e){
       Map<String,String> map = e.getBindingResult()
             .getFieldErrors().stream()
             .collect(Collectors.toMap(FieldError::getField,
DefaultMessageSourceResolvable::getDefaultMessage));
       List<String> errori = new ArrayList<>();
       for (String s : map.keySet()) {
          errori.add(s+": "+map.get(s));
       }
       MessageDto m = new MessageDto(errori, HttpStatus.BAD_REQUEST.value());
       return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(m);
    }
}
```

--SECURITY(da assegnare)

--CRITERIA QUERY(tutti fanno le proprie)

--JUNIT(andrà implementato singolarmente dai team)

--SWAGGER(andrà implementato singolarmente dai team)

--UTILPATH(andrà implementato singolarmente dai team)

--REPOSITORIES

(da ripartire in base alle esigenze)

Repository che mancano da assegnare:

-Apartment(Team Customer)

-Bill(Team Administrator)

-Condominium(Team Administratror)

-Scan(Team Technician)

-Intervention(Team Secretary)

--ServiceDefinitions

(ogni team scrive qui sotto i nomi dei metodi che utilizza nei service)

--ServiceImpls

(ogni team si gestisce le proprie)

--Facade

(qui si inseriscono i controlli)

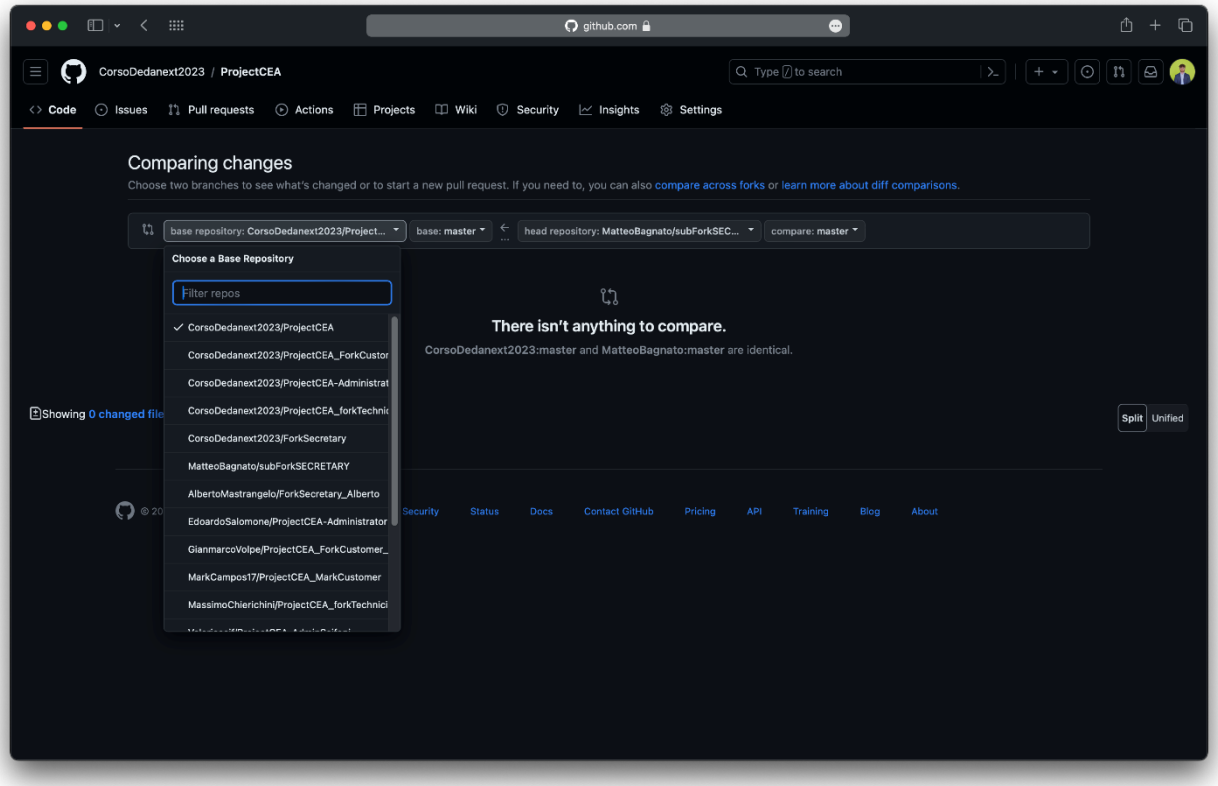
SEGUITE LA CONVENZIONE CHE SENNO MATTEO SI INCAZZA.

-PER GIT.

-ci sarà una fork principale, da dove si creeranno 4 fork di team, una per team, e da lì i membri creeranno una fork personale.

Si eseguono i push sulla propria fork personale, e si farà la pull request sulla fork di team, NON SULLA PRINCIPALE.

Ogni team si gestisce le pull request del suo branch, e si dovrà coordinare per la pull request sul branch principale.



PROJECTCEA NON SI DEVE MAI USARE COME DESTINAZIONE DEI COMMIT, SELEZIONARE LA FORK DEL TEAM

Per le fork di team è consigliato farsi approvare dagli altri membri del team le proprie pull request, mentre per quanto riguarda le pull request sulla fork principale(che verranno effettuate dal team), è OBBLIGATORIA l'approvazione di un altro team per la pull request.