Istituto di Istruzione Superiore
"A. Badoni"

Tesi di maturità

Alternanza scuola-lavoro estesa presso *Agomir S.p.A.*

Sviluppo di applicazioni multipiattaforma per dispositivi mobili

Claudio Maggioni

15 maggio 2018

Traduzione in italiano

Revisione 6

Indice

1 Informazioni generali sul lavoro svolto

A partire dal giorno 19/10/2017, collaboro con l'azienda *Agomir S.p.A*[10] (produttrice di software, sistemi e servizi per piccole e medie imprese) svolgendo un'attività di alternanza scuola-lavoro estesa con cadenza settimanale (giovedì e venerdì pomeriggio). Mi occupo di sviluppo software di tipo gestionale, nello specifico di applicazioni per smartphone multipiattaforma (cioè compatibili sia con Android che con iOS) utilizzando strumenti come *Ionic Framework*[1] e *Apache Cordova*[3].

Tale approccio al mondo *mobile* facilita lo sviluppo, perché al posto di usare API e meccanismi legati alla piattaforma è possibile utilizzare tecnologie note e standard come HTML e Javascript. Naturalmente tali applicazioni richiedono più risorse e tendono ad essere meno fluide, ma questo aspetto è meno rilevante in contesti gestionali come quelli affrontati da Agomir, non legati ad esempio al mondo dei videogiochi o all'elaborazione real-time.

2 Gestione dei progetti

Non ho diretto controllo manageriale su ciò che sviluppo in azienda, in quanto ho un ruolo simile a quello di un dipendente. L'incarico di gestire l'andamento dei progetti spetta a Mario Goretti, A.D. dell'azienda e capo del settore di sviluppo software gestionale (SWG), e ai suoi collaboratori.

In generale sviluppo i progetti da solo. Collaboro con il collega Daniele Crippa per l'interfacciamento con i software aziendali esistenti e per l'organizzazione di nuovi progetti, nonché consigli e dritte varie.

Nonostante non abbia controllo totale è comunque mia responsabilità fare stime orarie sul lavoro da svolgere nonché definire passi e *milestone* per i vari progetti.

Per aumentare la forza lavoro per lo sviluppo di applicazioni *mobile* ho coordinato momenti di formazione ad alcuni dipendenti nei quali ho mostrato il principale funzionamento delle tecnologie che uso.

3 Il progetto principale: *InteGRa Mobile*

Questa applicazione, una volta completata, dovrebbe permettere ad utenti in mobilità di interfacciarsi con alcune funzioni del prodotto ERP di punta di Agomir: il gestionale *InteGRa*[1]. Nel dettaglio, sarà possibile accedere alle seguenti sezioni:

Ordini cliente per registrare ordini di prodotti a clienti;

Soggetti per consultare informazioni anagrafiche di clienti e fornitori;

Agenda per consultare e aggiungere eventi nel calendario presente nell'ERP, il quale si può integrare con *Outlook*;

Magazzini gestire e inventariare scorte in magazzino.

[1] Sito internet di InteGRa ERP: `https://integra.agomir.com/`

In aggiunta, sarà possibile anche registrare le ore per interventi in trasferta, funzione già implementata nell'applicazione *InteGRa.Service*, sviluppata nei periodi di alternanza precedenti. Tale lavoro non è direttamente implementabile in *InteGRa Mobile* a causa di differenze consistenti nelle architetture dei due prodotti.

Inoltre, l'applicazione sarà in grado di funzionare in modo limitato anche senza connessione diretta ad *InteGRa*, permettendo la sincronizzazione dei dati modificati con il gestionale in un momento futuro.

3.1 Architettura software

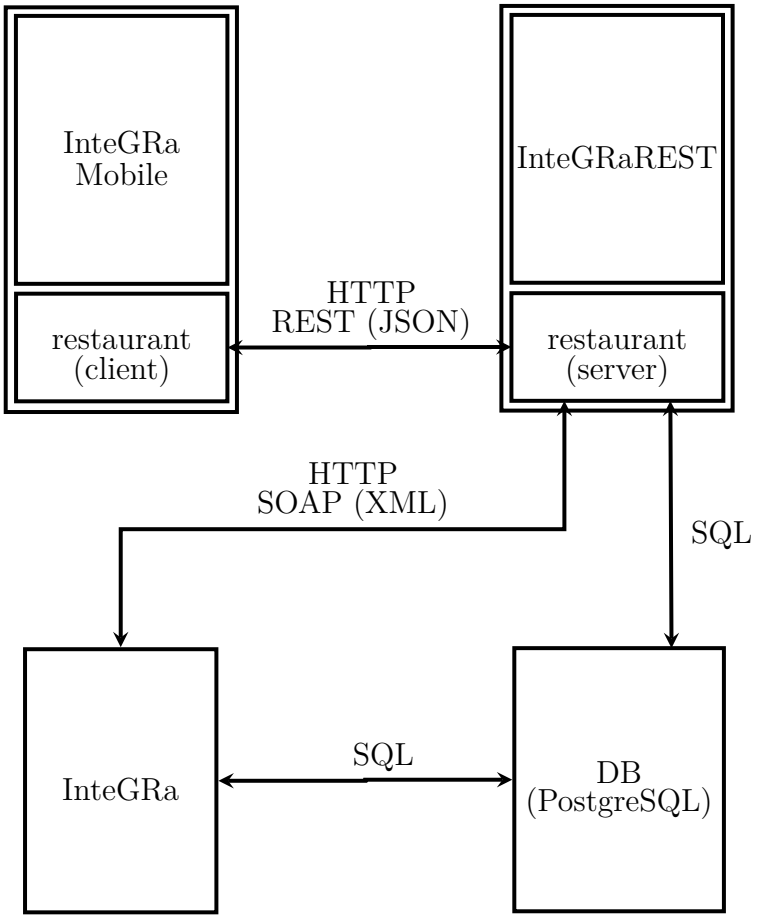

Figura 1: L'architettura di *InteGRa Mobile* raffigurata con un diagramma

Nella figura 1 si può notare come l'insieme dei componenti software all'interno del progetto siano organizzati e comunichino tra loro. Tale architettura è stata realizzata da me, basandosi sull'esperienza acquisita con il progetto *InteGRa.Service*. Segue una spiegazione sintetica (organizzata in punti) del diagramma.

- L'applicazione non comunica direttamente con il gestionale, ma tramite una serie di chiamate REST[13] ad un endpoint lato server (chiamato *InteGRaREST*) che svolge il ruolo di intermediario.

- Tale componente è una Java WebApplication che utilizza una libreria sviluppata internamente (chiamata *restaurant*) per offrire le rotte accessibili al client e per interagire con il database di *InteGRa*, una normale istanza di *PostgreSQL*.
- Nella fase di comunicazione tramite HTTP, i dati in input vengono trasmessi come `application/x-www-form-urlencoded` mentre i dati in output vengono trasmessi come JSON[11], e le sessioni vengono identificate con un token presente come parametro nella query string di ciascuna richiesta;
- *InteGRaREST* effettua autonomamente query SQL al database per recuperare velocemente informazioni aventi struttura dati e logiche di memorizzazione semplici;
- Nel caso sia necessario interagire con record complessi, *InteGRaREST* comunica con *InteGRa* attraverso chiamate SOAP[14];
- *InteGRa Mobile*, l'applicazione per dispositivi mobili, comunica con *InteGRaREST* grazie a una versione client di *restaurant*, in grado di gestire (in modo limitato) operazioni eseguite in mancanza di connessione alla rete.

3.1.1 *restaurant* – lato server

La libreria *restaurant*[2] viene importata come una normale dipendenza Maven[6]. Essa dipende da:

Hibernate ORM[9] un famoso Object/Relation Mapper[12] per Java, utile ad interrogare il database *PostgreSQL* con facilità;

GSON[8] una libreria di Google per serializzare e deserializzare oggetti Java in JSON;

Jersey[2] un'implementazione dell'API JAX-RS per realizzare gli endpoint REST. Ciò avviene senza servlet ma tramite semplici metodi contenuti in classi senza "padre" dette *controller*, opportunamente decorati con le annotazioni nel package `javax.ws.rs`.

Le operazioni richieste per interrogare la base di dati sono ulteriormente semplificate dal codice contenuto in *restaurant* su due livelli:

- È possibile eseguire normali operazioni di CRUD[15] nonchè costruire manualmente query tramite i metodi statici della classe *CRUDUtils*, che lavorano in input e in output con istanze delle classi "entità";
- Inoltre, è possibile creare nuovi *controller* con facilità ereditando dalle classi astratte *JsonProducer* (per implementare funzioni al di fuori di CRUD), *ReadableRESTController* (per implementare operazioni in sola lettura, equivalenti alla "Read" in CRUD) e *CRUDRESTController* (per implementare CRUD completi), che offono metodi in grado di generare autonomamente risposte in JSON, sia in caso di successo che in errore.

[2]Il nome *restaurant* è un gioco di parole sull'acronimo REST.

Seguono le interfacce delle classi *CRUDUtils*, *JsonProducer*, *ReadableRESTController* e *CRUDRESTController*, complete di Javadoc (in inglese, come presente nei sorgenti originali).

```
/**
 * Something that generates JSON responses
 */
public abstract class JsonProducer {

    protected static final Logger log = ...;

    protected static final Gson GSON = ...;

    /**
     * Generates a json response with status 200 containing an object
     *
     * @param o the contents of the response
     * @return JSON response
     */
    protected Response jsonResponse(Object o) { ... }

    /**
     * Generates a json response with custom status containing an object
     *
     * @param o the contents of the response
     * @param status status of the response
     * @return JSON response
     */
    protected Response jsonResponse(Object o, int status) { ... }

    /**
     * Generates a json response containing a library standard error
     *
     * @param error the error
     * @return JSON error response
     */
    protected Response jsonErrResponse(Error error) { ... }

    /**
     * Generates a json response containing a custom error
     *
     * @param error the error
     * @return JSON error response
     */
    protected Response jsonErrResponse(ErrorDetails error) { ... }

    /**
     * Generates an error response if the parameter given as argument
```

```
     * is null or an empty string
     *
     * @param s parameter
     * @param propName parameter name, used in the error description
     * @return JSON error response or null if the parameter is valid
     */
    protected Response require(Object s, String propName) { ... }
}
```

Listato 1: Interfaccia della classe *JsonProducer*

Per una migliore lettura del listato 1, si precisa che la classe *Error* è una enumerazione Java contenente messaggi di errore standard. La classe *ErrorDetails*, invece, permette al codice esterno di realizzare messaggi di errore personalizzati. Inoltre, l'oggetto statico `GSON` di classe *Gson* è il serializzatore a JSON offerto dalla libreria omonima.

```
/**
 * A REST Controller only readable
 * @param <T> The readable entity class
 */
public abstract class ReadableRESTController<T extends Readable,
    U extends PrimaryKey<T>> extends JsonProducer {

    /**
     * Interface used to carry a function capable of filtering a list
     * of entities after it has been retrieved from the database
     *
     * @param <T> The readable entity class
     */
    public interface ListFilter<T extends Readable> {
        List<T> filter(List<T> toFilter);
    }

    /**
     * Returns the class of the readable entity. Used as parameter for
     * CRUDUtils
     *
     * @return the class of the readable entity
     */
    protected abstract Class<T> getBeanClass();

    protected final Logger log = ...;

    /**
     * Returns a response with a standard unfiltered, unpaginated list
     * of entities
```

```
     *
     * @return JSON response, success or not
     */
    protected Response list() { ... }

    /**
     * Returns a list of entities, customizable in many ways
     *
     * @param filter map of names of fields and their values which must
     * be equal in the fetched entities (null accepted if
     * not used)
     * @param otherFilter list of additional filters (Hibernate Criterion
     * objects, null accepted if not used)
     * @param order order of the entities (Hibernate Order, null accepted
     * if not used)
     * @param afterDb implementation of interface containing a function
     * capable of filtering a list of entities (null accepted
     * if not used)
     * @param page the page number to fetch (starts at 1, null accepted
     * if not used)
     * @return JSON response, success or not
     */
    protected Response list(Map<String, Object> filter,
        List<Criterion> otherFilter, Order order,
        ListFilter<T> afterDb, Integer page) { ... }

    /**
     * Returns a response with a single entity given its primary key
     *
     * @param filterOne map of names of columns and their values containing
     * the primary key
     * @return JSON response, success or not
     */
    protected Response get(Map<String, Object> filterOne) { ... }

    /**
     * Returns a response with a single entity given its primary key
     *
     * @param pk the primary key
     * @return JSON response, success or not
     */
    protected Response get(PrimaryKey<T> pk) { ... }

    /**
     * Returns the existing primary keys given a set of primary keys sent
     * as a JSON array
     *
     * @param requestBody the request body with JSON data
     * @return JSON array of the existing primary keys or error response
```

```
     */
    protected Response existing(InputStream requestBody) { ... }
}
```

Listato 2: Interfaccia della classe *ReadableRESTController*

Anche il listato 2 necessita di una breve spiegazione. La classe astratta *Readable*, vincolo sul primo parametro generico della classe, è il padre di ogni classe "entità", mentre *PrimaryKey<T>* è il padre della classe contenente la chiave primaria di *T*. Un'istanza della classe figlia di *PrimaryKey<T>* è contenuta come attributo nella classe "entità" *T*, che a sua volta estende *Readable*.

```
/**
 * A REST Controller readable and writable
 * @param <T> The readable and writable entity class
 */
public abstract class CRUDRESTController<T extends CRUDable, U extends
    PrimaryKey<T>> extends ReadableRESTController<T, U> {

    /**
     * A standard edit route body. Edits an entity and returns a JSON
     * response, both on success and on error.
     *
     * @param filterOne map containing columns and relative values of the
     * primary key of the entity to edit
     * @param form request data containing the fields to edit
     * @return JSON response, positive or negative
     */
    protected Response edit(Map<String, Object> filterOne,
                            MultivaluedMap<String, String> form) { ... }

    /**
     * A standard edit route body. Edits an entity and returns a JSON
     * response, both on success and on error.
     *
     * @param pKey the entity primary key
     * @param form request data containing the fields to edit
     * @return JSON response, positive or negative
     */
    protected Response edit(PrimaryKey<T> pKey,
        MultivaluedMap<String, String> form) { ... }

    /**
     * A standard delete route body. Deletes an entity and returns a JSON
     * response, both on success and on error.
     *
     * @param pKey the entity primary key
```

```
     * @return JSON response, positive or negative
     */
    protected Response delete(PrimaryKey<T> pKey) { ... }

    /**
     * A standard delete route body. Deletes an entity and returns a JSON
     * response, both on success and on error.
     *
     * @param filterOne map containing columns and relative values of the
     * primary key of the entity to delete
     * @return JSON response, positive or negative
     */
    protected Response delete(Map<String, Object> filterOne) { ... }

    /**
     * A standard create route body. Creates an entity and returns a JSON
     * response, both on success and on error.
     *
     * @param form request data containing the entity column values
     * @return JSON response, positive or negative
     */
    protected Response create(MultivaluedMap<String, String> form) { ... }

    ...
}
```

Listato 3: Interfaccia della classe *CRUDRESTController*

Si precisa che la classe *CRUDable* è un estensione della classe *Readable* che contiene metodi per l'inizializzazione dell'oggetto sulla base di dati in input, contenuti in *MultivaluedMap<String, String>*. Di conseguenza, le classi "entità" che estendono CRUDable possono essere usate per operazioni di scrittura, come quelle implementate da *CRUDRESTController*.

```
/**
 * Database access class. Access statically
 */
public abstract class CRUDUtils {

    /**
     * Interface containing a function able to generate an Hibernate Query
     * object and execute it by getting a generic response given the
     * session
     *
     * @param <T> the response type
     */
```

```
public interface HBMInteraction<T> {
    T execute(Session s);
}

/**
 * Interface containing a function able to generate an Hibernate Query
 * object and execute it by getting a list of rows given the session
 */
public interface HBMQuery extends HBMInteraction<List<Object[]>> {
    List<Object[]> execute(Session s);
}

/**
 * Number of items on a single page. Used when pagination is requested
 */
public static final int ITEMSPERPAGE = ...;

...

private static final Logger log = ...;

/**
 * Save a writable entity on the database
 *
 * @param o the entity to save
 * @param <T> type of the entity. Must be readable and writable
 * @return boolean true = success, false = error
 */
public static <T extends CRUDable> boolean persist(final T o) { ... }

/**
 * Delete a writable entity on the database
 *
 * @param toDelete the entity to delete
 * @param <T> type of the entity. Must be readable and writable
 * @return boolean true = success, false = error
 */
public static <T extends CRUDable> boolean delete(final T toDelete)
    { ... }

...

/**
 * Execute a query which returns data of a generic type
 *
 * @param query HBMInteraction implementation
 * @param <T> The returning type
 * @return the data
 */
```

```
public static <T> T executeHBMInteraction(HBMInteraction<T> query)
    { ... }

/**
 * Get a list of entities
 *
 * @param tClass class of the entities
 * @param restrictions map of names of fields and their values which
 * must be equal in the fetched entities
 * (null accepted if not used)
 * @param criteria list of additional filters (Hibernate Criterion
 * objects, null accepted if not used)
 * @param <T> entity type. Must be readable
 * @param o order of the entities (Hibernate Order, null accepted if
 * not used)
 * @param page the page number to fetch (starts at 1, null accepted if
 * not used)
 * @return the list of entities
 */
public static <T extends Readable> List<T> get(Class<T> tClass,
    Map<String, Object> restrictions, List<Criterion> criteria,
    Order[] o, Integer page) { ... }

/**
 * Get a list of entities
 *
 * @param tClass class of the entities
 * @param <T> entity type. Must be readable
 * @return the list of entities
 */
public static <T extends Readable> List<T> get(Class<T> tClass) { ... }


/**
 * Get a single entity
 *
 * @param tClass class of the entities
 * @param restrictions map of names of fields and their values which
 * must be equal in the fetched entities
 * (null accepted if not used)
 * @param <T> entity type. Must be readable
 * @return the list of entities
 */
public static <T extends Readable> T getOne(Class<T> tClass,
    Map<String, Object> restrictions) { ... }

/**
 * Get a single entity
 *
```

```
     * @param tClass class of the entities
     * @param pk the primary key object of the entity
     * @param <T> entity type. Must be readable and contain a primary key
     * object
     * @return the list of entities
     */
    public static <T extends Readable> T getOne(Class<T> tClass,
        PrimaryKey<T> pk) { ... }
}
```

Listato 4: Interfaccia della classe *CRUDUtils*

Oltre a quanto mostrato, *restaurant* contiene vari metodi utilità per la gestione di file `.properties`, per la validazione dei dati in input, per la registrazione della *timestamp* di creazione o modifica di un record, e altro ancora.

3.1.2 *restaurant* – lato client

```
export class RESTService {

  ...

  /**
   * Read all the pages of an entity in order to save them for offline
   * readings.
   *
   * @param {{ new(): T }} - class of the table to read
   * @param {any} - key data for the search. Must not contain filters
   * or a field with key '_page'
   * @return {Promise<boolean>} boolean promise that resolves when the
   * reading has been done, true if succesful,
   * false if not
   */
  public bulkRead<T extends Table>(type: TableStatic, restrictions?: any):
    Promise<boolean> { ... }

  /**
   * Retrieves a list of entities. When online and requested without
   * filters, the list is saved in the offline storage. When offline,
   * the data from the offline storage is read and filtered accordingly.
   *
   * Request criteria regarding directly properties of the entity MUST
   * have the following format:
   *
   * 'attr-' + name_of_property[.nested] [+ '||' + other_property] +
   * ('-like' | '-eq' | '-flag' | '-gt' | '-lt' | '-gte' | '-lte')
   *
```

```
 * Multiple field names separed by '||' state that the search must be
 * done on multiple fields with a logic OR.
 *
 * The different suffixes denote different logical criteria. Look up
 * the documentation of meetsCriteriaFilter*(...) methods in
 * order to get detailed info on each one.
 *
 * @param {{ new(): T }} type - type of the table to read
 * @param {number} page - number of the page starting at 1 (in offline
 * mode, a 20 entities slice; in online mode,
 * as defined by the server). -1 is for no
 * pagination
 * @param {any} restrictions - restrictions required by the server,
 * and filters.
 * @param {boolean} offline - whether to fetch (true) or not (false)
 * offline entities to sync. Defaults to true
 * @return {Promise<T[]>} - promise of the expanded list, unsuccesful
 * when the storage readings are unsuccesful
 */
public read<T extends Table>(type: TableStatic, page: number,
  restrictions?: any, addOffline?: boolean): Promise<T[]> { ... }

/**
 * Asks to the server which entity instances of the ones
 * saved in the offline storage exist and deletes the
 * ones that do not exist on the server.
 *
 * @param {TableStatic} type - the entity type
 * @return {Promise<void>} promise rejecting only if an error occurs
 */
private async existing(type: TableStatic): Promise<void> { ... }

/**
 * Reads a single entity, even in the offline storage if offline.
 *
 * @param {{ new(): T }} type - class of the entity to read
 * @param {any} keyData - the primary key fields and their values
 * in a JS Object, plus the 'id' field of
 * type string used in the end bit of the
 * request path.
 * @return {Promise<T>} - promise of the element, unsuccesful if
 * not found.
 */
public readOne<T extends Table>(type: TableStatic | string,
  keyData: any): Promise<T> { ... }

/**
 * Updates an entity on the server or, if offline, saves it in the
 * offline sync queue.
```

```
   *
   * @param {{ new(): T }} type - class of the entity to be saved
   * @param {T} value - the entity to save
   * @param {any} keyData - the primary key fields and their values
   * in a JS Object, plus the 'id' field of
   * type string used in the end bit of the
   * request path.
   * @return {Promise<T>} - promise of the saved element.
   */
  public update<T extends Table>(type: TableStatic | string, value: T,
    keyData: any): Promise<T> { ... }

  /**
   * Creates an entity on the server or, if offline, saves it in the
   * offline sync queue.
   *
   * @param {{ new(): T }} type - class of the entity to be deleted
   * @param {string} id - string used in the end bit of the request path.
   * @param {any} keyData - the primary key fields and their values in
   * a JS Object
   * @return {Promise<T>} - promise of the element deleted
   */
  public create<T extends Table>(type: TableStatic | string, value: T):
    Promise<T> { ... }

  /**
   * Deletes an entity on the server or, if offline, marks it for
   * deletion in the offline sync queue.
   *
   * @param {{ new(): T }} type - class of the entity to be saved
   * @return {Promise<T>} - promise of the element created.
   */
  public delete<T extends Table>(type: TableStatic | string, keyData: any):
    Promise<T> { ... }
}
```

Listato 5: Interfaccia della classe *RESTService*

3.2 Stato del progetto

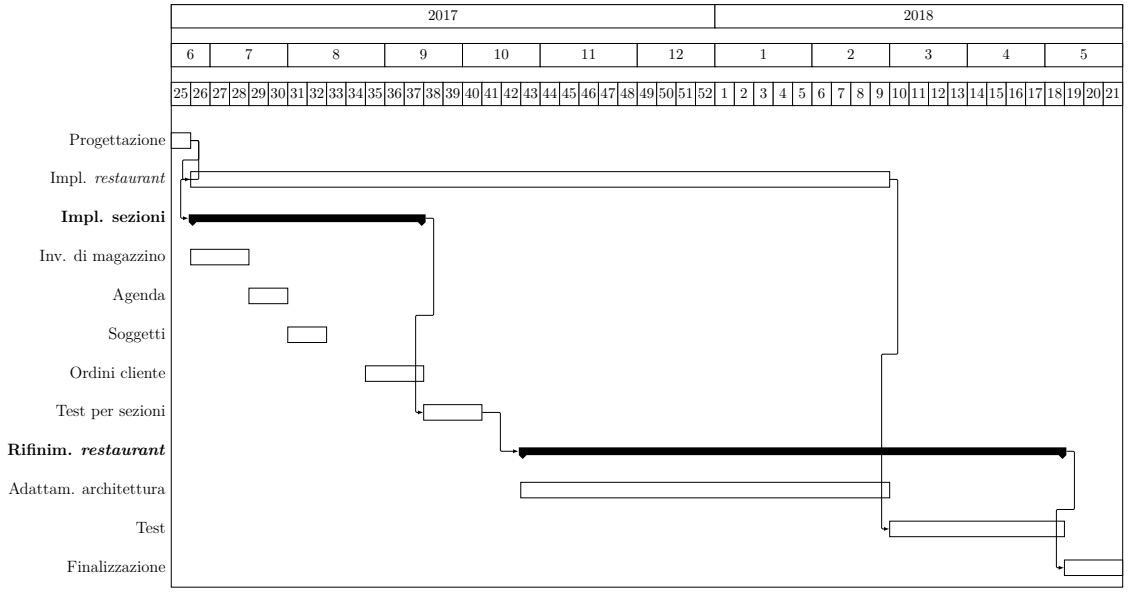

Figura 2: Il diagramma di Gantt del progetto *InteGRa Mobile*

Nella lettura del diagramma di Gantt in figura 2 è necessario tenere a mente che il completamento dell'intera applicazione non è certo per la fine di maggio. Ciò che dovrà essere necessariamente completato è la parte più importante del progetto: *restaurant*, il motore di sincronizzazione online/offline e libreria generica che permette di implementare velocemente nuove sezioni dell'applicazione.

3.3 Fasi del progetto

Le fasi del progetto, di cui date di inizio e fine sono state specificate nel diagramma in figura 2, sono:

Progettazione delineazione delle funzionalità da implementare nell'applicazione e prime bozze dell'interfaccia utente;

Implementazione *restaurant* implementazione della libreria, su cui si basano tutte le sezioni del programma. Vista l'importanza di questo componente, gli sviluppi vengono continuati in parallelo con l'implementazione delle sezioni;

Implementazione sezioni creazione di interfaccia e logica per le funzionalitá previste, cioé:

- Magazzini;
- Agenda;
- Soggetti;
- Ordini cliente;

Test per le sezioni messa alla prova degli algoritmi e della UI per efficacia ed efficienza, nonché eventuali correzioni;

Rifinimento *restaurant* messa alla prova delle scelte architetturali e algoritmi usati nella libreria e rifinimento di essi; Fase divisa in:

Miglioramento architettura analisi del protocollo per individuare punti di debolezza e possibili ottimizzazioni;

Test per *restaurant* messa alla prova dei cambiamenti fatti;

Finalizzazione task finali del progetto, tra cui branding, operazioni pre-rilascio, e gestione di eventuali personalizzazioni richieste dai clienti.

4 Il progetto secondario: *Guac Remote*

Questo progetto è destinato ad un'importante azienda del territorio, produttrice di macchine equilibratrici. Tale applicazione dovrebbe fungere da client di desktop remoto (come *TeamViewer*[3]) per il PC presente nel loro prodotto di punta, un sistema di calibrazione delle ruote di autoveicoli funzionante mediante telecamere. Il programma dovrebbe permettere all'operatore di tale prodotto di interagire con il software presente nel macchinario, senza scendere dal veicolo.

A causa di urgenza del committente, lo sviluppo di questo progetto ha interrotto e attualmente interrompe gli sviluppi per *InteGRa Mobile*.

4.1 Architettura software

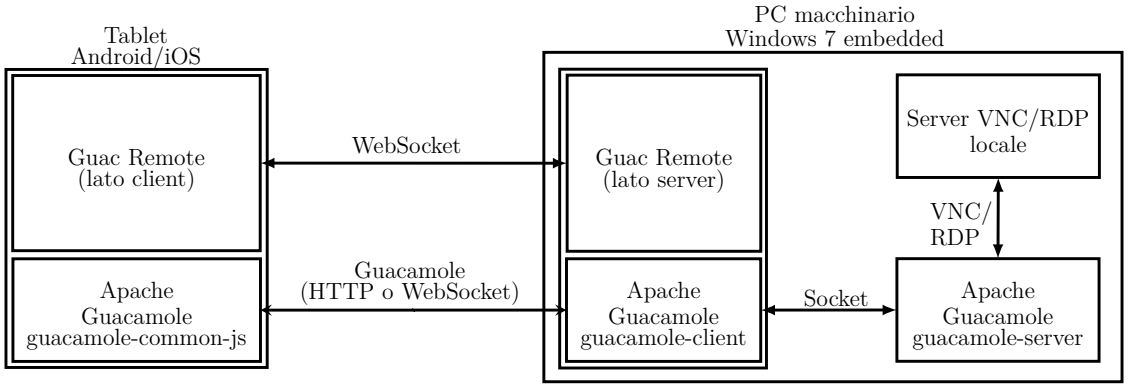

Figura 3: L'architettura di *Guac Remote*

Guac Remote è basata su un'applicativo e libreria per la connessione a computer remoto chiamato *Apache Guacamole*[4]. Tale software è costituito da due parti: [7]

***guacamole-server* (o *guacd*)** un servizio scritto in C che funge da adattatore tra il protocollo *guacamole* e i protocolli VNC, RDP o SSH, utilizzando questi ultimi per stabilire connessioni con gli host remoti;

[3]TeamViewer: `https://www.teamviewer.com/it/`

guacamole-client una WebApplication scritta tramite *Java servlet* che fornisce un'interfaccia web per interagire con *guacamole-server* e connettersi ai PC remoti.

Data la natura open-source del progetto, sia *guacamole-server* che *guacamole-client* possono essere usati come libreria per la realizzazione di software derivati. In particolare, *guacamole-client* può essere scomposto nelle librerie *guacamole-common-js*, che contiene il codice Javascript per il client, e *guacamole-common*, che fornisce classi Java per la connessione con *guacamole-server*. [5]

In *Guac Remote*, *Apache Guacamole* è usato per fornire accesso remoto al PC presente nel macchinario, che contiene l'applicativo per la calibrazione, all'applicazione installata sul tablet. Nel dettaglio, *guacamole-common-js* è usato nell'applicazione *mobile* fornire un'interfaccia touchscreen per interagire con l'host remoto, mentre *guacamole-common*, tramite una piccola WebApplication, assieme a *guacamole-server* sono installati sul PC.

In aggiunta, tablet e PC possono comunicare informazioni aggiuntive tramite una WebSocket creata al momento della connessione, necessaria per alcune estensioni al protocollo richieste dal cliente.

4.2 Stato del progetto

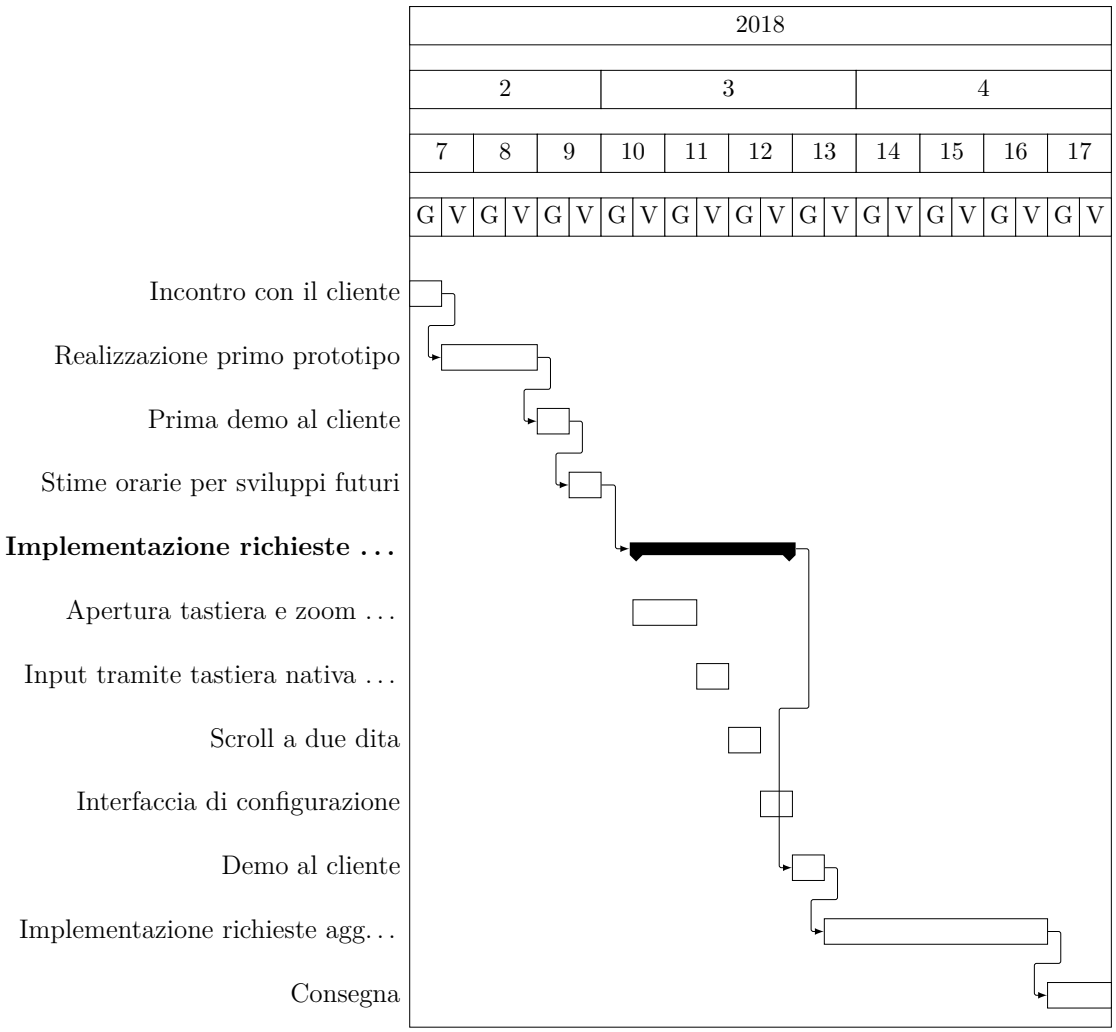

Figura 4: Il diagramma di Gantt del progetto *Guac Remote*

Questo progetto, al giorno 06/04, non ha ritardi. L'applicazione é stata mostrata in demo il giorno 29/03, ed il cliente si ritiene soddisfatto del lavoro fatto fino ad ora.

4.3 Fasi del progetto

Le fasi del progetto, di cui le date di inizio e di fine sono indicate nel diagramma della figura 4, sono:

Incontro con il cliente primo scambio di informazioni per capire gli obiettivi del progetto;

Realizzazione primo prototipo dimostrazione dell'efficacia del protocollo *guacamole* tramite un prototipo del prodotto;

Prima demo al cliente demo del prototipo al cliente;

Stime orarie per sviluppi futuri delineazione della tabella di marcia per gli sviluppi futuri;

Implementazione richieste del cliente sviluppo delle estensioni al protocollo richieste. Nel dettaglio, esse sono:

Apertura tastiera e zoom al "focus" di un input alla pressione di un campo di testo, l'applicazione *mobile* deve automaticamente ingrandire l'area selezionata e mostrare una tastiera;

Input tramite tastiera nativa Android o iOS sostituzione della tastiera su schermo di *guacamole-common-js* con quella nativa;

Scroll a due dita supporto della gesture di scroll verticale a due dita;

Interfaccia di configurazione creazione di una piccola finestra di configurazione, in cui inserire IP e porta del PC;

Demo al cliente dimostrazione degli sviluppi fatti al cliente e eventuale definizione di richieste aggiuntive;

Implementazione delle richieste aggiuntive del cliente

Consegna operazioni finali del progetto, tra cui branding e compilazione per rilascio.

Elenco delle figure

Elenco dei listati

Riferimenti bibliografici

[1] Ionic (precedentemente Drifty Co.) *Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular*. [Online; in data 30-aprile-2018]. URL: `https://ionicframework.com/framework`.

[2] Oracle Corporation. *Jersey*. [Online; in data 1-maggio-2018]. URL: `https://jersey.github.io`.

[3] The Apache Software Foundation. *Apache Cordova*. [Online; in data 30-aprile-2018]. URL: `https://cordova.apache.org`.

[4] The Apache Software Foundation. *Apache Guacamole©*. [Online; in data 1-maggio-2018]. URL: `https://guacamole.apache.org`.

[5] The Apache Software Foundation. *API Documentation – Apache Guacamole*. [Online; in data 9-aprile-2018, traduzione e rielaborazione proprie]. URL: `https://guacamole.apache.org/api-documentation/`.

[6] The Apache Software Foundation. *Maven – Welcome to Apache Maven*. [Online; in data 1-maggio-2018]. URL: `https://maven.apache.org`.

[7] The Apache Software Fundation. *Guacamole Manual, Chapter 1. Implementation and architecture*. [Online; in data 9-aprile-2018, traduzione e rielaborazione proprie]. URL: `https://guacamole.apache.org/doc/gug/guacamole-architecture.html`.

[8] Google Inc. *GitHub - google/gson: A Java serialization/deserialization library to convert Java Objects into JSON and back*. [Online; in data 1-maggio-2018]. URL: `https://github.com/google/gson`.

[9] Inc. Red Hat. *Your relational data. Objectively. - Hibernate ORM*. [Online; in data 1-maggio-2018]. URL: `https://hibernate.org/orm`.

[10] Agomir S.p.A. *Agomir S.p.A. - Software - Sistemi - Servizi*. [Online; in data 30-aprile-2018]. URL: `https://www.agomir.com`.

[11] Wikipedia. *JavaScript Object Notation — Wikipedia, L'enciclopedia libera.* [Online; in data 9-aprile-2018]. 2018. URL: https://it.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=94928405.

[12] Wikipedia. *Object-relational mapping — Wikipedia, L'enciclopedia libera.* [Online; in data 1-maggio-2018]. 2017. URL: https://it.wikipedia.org/w/index.php?title=Object-relational_mapping&oldid=92163909.

[13] Wikipedia. *Representational State Transfer — Wikipedia, L'enciclopedia libera.* [Online; in data 9-aprile-2018]. 2018. URL: https://it.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=95970771.

[14] Wikipedia. *SOAP — Wikipedia, L'enciclopedia libera.* [Online; in data 9-aprile-2018]. 2017. URL: https://it.wikipedia.org/w/index.php?title=SOAP&oldid=85936136.

[15] Wikipedia. *Tavola CRUD — Wikipedia, L'enciclopedia libera.* [Online; in data 1-maggio-2018]. 2017. URL: https://it.wikipedia.org/w/index.php?title=Tavola_CRUD&oldid=85784228.