Facultad de Ingeniería
Universidad de la República

Modelado y visualización de datos utilizando d3.js

Proyecto de Grado

Ingeniería en Computación

Agustín Prado
Santiago Bianchi

Tutores: Ewelina Bakala y Juan José Prada

2018

Resumen

Este trabajo consiste en el desarrollo de visualizaciones dinámicas e interactivas para representar datos de un caso de estudio seleccionado, utilizando la herramienta *d3.js*.
Con este fin, se da una introducción teórica de las etapas más importantes del proceso de visualización de datos. También se hace una introducción a *d3.js*, comparando sus principales funcionalidades con alternativas actuales, fundamentando el hecho de que sea una de las herramientas para crear visualizaciones web más poderosas de la actualidad.
Una parte muy importante de este proyecto consiste en la selección de un caso de estudio que permita explotar la mayor cantidad posible de características de *d3.js*, y que a la vez sea interesante su aplicación.
Luego de haber analizado distintas opciones - fundamentalmente estudiando los datos abiertos del Catálogo Nacional -, nos inclinamos por la Encuesta Metropolitana de Movilidad de Montevideo. Se detalla su modelo de datos y principales características, explicando también el fundamento de esta selección. Se aplica la teoría de visualización al caso de estudio seleccionado, se diseña, implementa y publica[1] una aplicación *web* con las visualizaciones representando sus datos, construidas con *d3.js*.
Finalmente se plantean las conclusiones obtenidas a partir del proyecto y posibles trabajos a futuro.

Palabras clave: *d3.js*, visualización de datos, modelado de datos, aplicación *web*

[1] http://proygrado18.s3-website-sa-east-1.amazonaws.com

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se presentan la motivación y los objetivos del proyecto. Se plantea y define el problema, los resultados esperados y un cronograma de las tareas realizadas. Además se describe la organización general del documento.

1.1. Motivación

En la sociedad de la información y el conocimiento en la que estamos inmersos, la idea de que la información es poder está muy extendida, pero esto no es del todo correcto según sugiere el autor Ignasi Alcalde[1]. Por lo que sería más adecuado, según el autor, decir que el conocimiento es poder y no solo la información por si misma.
Cada vez se maneja mayor cantidad de datos, sin embargo, tal y como se afirma en la web datos.gob.es[2], una de las críticas más comunes en torno a la existencia de este gran volumen es su poca y mala utilización.
Estamos rodeados de datos, pero estamos de acuerdo con lo expuesto por Ignasi Alcalde[3], en el hecho de que estos por si solos no tienen ningún sentido si no somos capaces de analizarlos, darles el contexto adecuado y transformarlos para generar información.

d3.js nos ayuda a darle vida a los datos utilizando *SVG*(*Scalable Vector Graphics*), *CSS3* y *HTML5*, combinando poderosos componentes de visualización, con un enfoque basado en datos para la manipulación del *DOM*(*Document Object Model*), lo cuál permite que la estructura, estilo y contenido de un documento sean

modificados dinámicamente.

Según Albert Obiols[4], el objetivo principal de la visualización es comunicar información e ideas complejas de forma clara, precisa y eficiente; ayudando a los usuarios a analizar y razonar sobre datos y evidencias.
La visualización de datos es a la vez un arte y una ciencia, como dice el refrán "una imagen vale más que mil palabras", o adaptada a nuestros tiempos "una imagen vale más que mil líneas de datos".

Solo cuando realizamos un análisis y modelado que permita una correcta interpretación de los datos, éstos cobran sentido y se transforman en información y conocimiento.

1.2. Objetivos

El objetivo general de este proyecto es:

- Desarrollo de visualizaciones dinámicas e interactivas para representar datos de un caso de estudio elegido, utilizando la herramienta *d3.js*.

Como objetivos específicos, se tienen:

- Modelado de datos y diseño de la base de datos.
- Familiarización con la biblioteca *d3.js* y las etapas de visualización de datos.
- Diseño e implementación de una aplicación *web* responsiva e interactiva de visualización de datos que permita un fácil entendimiento de los datos del caso de estudio.

1.3. Resultados esperados

El resultado esperado de este proyecto es aplicar las etapas de modelado y visualización de datos a un caso de estudio especifico para su posterior representación, utilizando la herramienta de generación de gráficos *web d3.js*, construyendo una aplicación *web* como vehículo para este propósito.

1.4. Cronograma

En esta sección se presenta el cronograma seguido en el desarrollo de este proyecto.

Para facilitar su visualización, enumeramos las actividades realizadas para luego representarlas en la tabla 1.1.

1. Investigación para selección de caso de estudio.
2. Relevamiento de los datos del catálogo de datos abiertos del Uruguay.
3. Investigación del estado del arte sobre visualización y modelado de datos.
4. Investigación sobre *d3.js* y herramientas alternativas.
5. Experimentación con prototipos en *d3.js* utilizando mapas.
6. Modelado de datos del caso de estudio.
7. Diseño de la solución.
8. Implementación de la solución.
9. Preparación del informe final.
10. Preparación de presentación y defensa oral.

Act.	Abr/May	Jun/Jul	Ago/Set	Oct/Nov	Dic/Ene	Feb/Mar	Abr/May
1	x	x					
2	x	x					
3	x	x					
4	x	x					
5	x	x					
6			x				
7			x	x			
8				x	x	x	
9			x	x	x	x	x
10							x

Tabla 1.1: Cronograma de actividades

1.5. Estructura del documento

El documento se divide en ocho capítulos. En el capítulo dos se realiza una revisión del estado del arte. Se realiza una introducción básica sobre visualización de datos, necesaria para entender el proyecto. Se describe la herramienta *d3.js* y se realiza una revisión de las posibles alternativas a las herramientas a utilizar.

En el capítulo tres se describe el proceso seguido hasta la selección del caso de estudio elegido.

Luego, en el capítulo cuatro se describe el modelado de datos realizado para el caso de estudio elegido, y la estructura de datos junto con las transformaciones realizadas.

En el capítulo cinco se define el diseño de la solución, su arquitectura y el diseño de la base de datos.

En el capítulo seis se describe la implementación de la solución y todas las herramientas a utilizar para realizar el desarrollo, el ambiente de trabajo, la base de datos y las transformaciones de datos realizadas.

El capítulo siete contiene los resultados, es decir, las visualizaciones obtenidas mediante la creación de la herramienta de visualización. Aquí se muestran todos los resultados obtenidos.

Finalmente, en el capítulo ocho se encuentran las conclusiones del proyecto y se comentan los posibles trabajos futuros vinculados a este proyecto.

Al final del documento se pueden encontrar los Anexos, Glosario y Bibliografía.

Capítulo 2

Estado del Arte

En este capítulo se presenta los conceptos teóricos de visualización que consideramos más importantes para entender el proyecto. Se realiza un estudio de las herramientas disponibles para su desarrollo, mencionando sus ventajas y desventajas.

2.1. Introducción a la visualización de datos

Para entender por que es importante la visualización de datos, es necesario entender la diferencia entre datos e información.

Un dato es un elemento aislado, recabado para un cierto fin, pero que no ha pasado por un proceso que lo relacione con otros de manera funcional para un fin previsto, según afirma Ignasi Alcalde [1]. Por otro lado, la información es la interpretación y relacionamiento de un conjunto de datos referentes a un tema.

Esto quiere decir que la información se refiere a los datos que han sido procesados y presentados de tal manera que pueden ser entendidos e interpretados por el receptor. La presentación de los datos procesados, es decir la información, es una parte fundamental en la generación de conocimiento por parte del receptor.

La visualización consiste en transmitir esta información mediante imágenes que facilitan la extracción de significado [1]. Una enorme parte de nuestro cerebro está dedicada a procesar información que viene a través de la vista, por lo que es muy

importante saber aprovechar esta capacidad para comunicarnos con más eficacia.

2.1.1. Etapas en la visualización de datos

A continuación se detallan las etapas del proceso de visualización de datos, partiendo desde los datos crudos hasta la representación e interacción, como lo describe Ben Fry [5].

Cabe aclarar que estas etapas son genéricas por lo que en algunos casos es posible que se apliquen parcialmente o de manera modificada.

Acquire: Consiste en la obtención de los datos crudos.

Parse: Se basa en la representación de los datos recopilados mediante una estructura definida y con un formato específico.

Filter: Consiste en remover o ignorar datos innecesarios.

Mine: Aplicación de métodos estadísticos y de *data mining* que permitan encontrar patrones o situar los datos en un contexto matemático.

Represent: Representación mediante la elección de modelos visuales.

Refine: Mejora de la representación básica con el fin de hacerla más clara y atractiva visualmente.

Interact: Interacción mediante el agregado de métodos que permitan manipular los datos y controlar sus características visibles.

Esta sección se referencia a lo largo del documento a medida que se aplica cada etapa al caso de estudio elegido.

2.2. Herramientas de visualización

Si bien la herramienta de visualización a utilizar es *d3.js*, se entiende pertinente estudiar otras herramientas, por lo que en esta sección se describen y comparan las principales herramientas existentes para la generación de visualizaciones interactivas web mediante *Javascript* y otros lenguajes, a partir de un conjunto de

datos de entrada.

Las librerías más utilizadas basadas en *Javascript* son - junto con *d3.js* - *Raphaël* y *Processing.js*.

También existen opciones para generar visualizaciones sin utilizar *Javascript*, mediante *Adobe Flash* y su lenguaje de programación *ActionScript*, una tecnología ya obsoleta de la empresa Adobe. Existe también un *toolkit* basado en Java para la creación interactiva de aplicaciones de visualización de la información llamado *Prefuse*.

2.2.1. *d3.js*

d3.js es una librería escrita en el lenguaje de programación interpretado *Javascript* para manipular documentos basándose en datos. Permite realizar representaciones visuales, dinámicas e interactivas en navegadores web, utilizando *HTML5*, *CSS3* y *SVG*, de manera de manipular el *DOM* ajustándolo a los datos deseados.

d3.js emplea una técnica llamada *chain syntax*, en la cual los métodos son encadenados entre sí con puntos, permitiendo realizar varias acciones en una sola línea.

Su primer versión se remonta al año 2011, es desarrollada y mantenida por Mike Bostock, tiene un tipo de licencia *BSD* [6]. Su desarrollo comenzó en 2009, primero con la librería *Protovis* que luego quedó obsoleta para continuar con *d3.js*, según la pagina del creador [1].

Es una de las herramientas más utilizadas hoy en día para la generación de visualizaciones en la web. Entre sus principales funcionalidades se destacan:

1. *Forces*: Característica que permite la creación de grafos dirigidos con aplicación de fuerzas utilizando integración del método de Verlet con velocidad. El método de Verlet es un método numérico utilizado para integrar las ecuaciones de movimiento de Newton [7]. Se utiliza con frecuencia para calcular trayectorias de partículas en simulaciones de dinámica molecular y gráficos computacionales.

2. *Geographies*: Característica que permite crear proyecciones geográficas a partir de datos de entrada.

[1] http://mbostock.github.io/d3/tutorial/protovis.html

3. *Dragging*: Característica que permite *drag and drop* de elementos SVG, HTML o Canvas con el ratón o pantalla táctil.

4. *Zoom*: Característica que permite el encuadre y zoom de elementos SVG, HTML o Canvas con el ratón o la pantalla táctil.

5. *Voronoi Diagrams*: Característica que permite la construcción de un diagrama de Voronoi para un conjunto de puntos dados. El diagrama de Voronoi de un conjunto de puntos en el plano es la división de dicho plano en regiones, de tal forma, que a cada punto le asigna una región del plano formada por los puntos que son más cercanos a él que a ninguno de los otros objetos.

Entre las muchas características a favor que tiene *d3.js*, se pueden nombrar la gran cantidad de ejemplos en la web, aunque muchos son escritos por el autor. También la gran comunidad que utiliza *d3.js* permite que haya muchos recursos de aprendizaje como guías y cursos, así como rápidas respuestas en los foros de preguntas. Al seguir *d3.js* los estándares *web* actuales, no hace necesaria la utilización de *framework*s propietarios.

Como contrapartida, el aprendizaje del uso de la librería *d3.js* no es tan simple.

2.2.2. *Raphaël*

Raphaël[2] es una pequeña biblioteca de *Javascript* que funciona con gráficos vectoriales en la *web* para la generación de visualizaciones.

Raphaël usa *SVG* como base para crear gráficos, por lo que cada objeto gráfico que cree también es un objeto *DOM*, puede adjuntar controladores de eventos de *Javascript* o modificarlos más tarde. El objetivo de *Raphaël* es proporcionar un adaptador que haga que el dibujo compatible con arte vectorial sea compatible con todos los navegadores [8].

2.2.3. *Processing.js*

Processing.js[3] se basa en el popular lenguaje de programación *Processing*, diseñado para la *web* mediante el cual se pueden generar visualizaciones. *Processing.js* hace que las visualizaciones de datos, arte digital, animaciones interactivas, gráficos educativos, videojuegos, etc. funcionen utilizando estándares *web* y sin ningún

[2]https://dmitrybaranovskiy.github.io/raphael/
[3]https://processing.org/

complemento.

Desarrollado originalmente por Ben Fry y Casey Reas, *Processing* comenzó como un lenguaje de programación de código abierto basado en Java para ayudar a las comunidades de arte electrónico y diseño visual a aprender los conceptos básicos de la programación de computadoras en el contexto de arte visual. *Processing.js* permite que el código de procesamiento sea ejecutado por cualquier navegador compatible con *HTML5* [9].

2.2.4. *Prefuse*

Prefuse[4] es un entorno de software basado en Java que permite la creación interactiva de aplicaciones de visualización de información. Se puede utilizar para crear aplicaciones independientes, componentes visuales y *applet*s.

Pretende simplificar los procesos de visualización, control y asignación de datos, así como la interacción del usuario. Permite la navegación *rizoma* siendo este un método para crear dinámicamente una interfaz de navegación para sistemas de datos, tales como bases de datos y sitios *web* en el que los vínculos de navegación que se presentan al usuario no están predefinidos, sino que se generan en respuesta al comportamiento de los usuarios y al análisis de otros datos [10].

2.2.5. *Adobe Flash y ActionScript*

ActionScript [11] es un lenguaje interpretado basado en el estándar *ECMAScript*, mediante el cual están escritas las animaciones que se reproducen utilizando el reproductor *Flash Player*.

Adobe Flash [12] puede ofrecer una experiencia muy rica en cuanto a animaciones, transiciones y manejo de música. Ha estado allí desde el primer día para complementar *HTML*.

Especialmente a partir de los últimos años, también se ha convertido en la forma por defecto de mostrar vídeos en sitios *web* (*YouTube*, *Vimeo*, etc.) debido a sus capacidades de compresión y empaquetado, y es una gran manera de solucionar problemas comunes de *códec* de vídeo, mostrando pantalla completa y otras características.

[4] https://prefuse.org/

La dificultad de aprendizaje de *ActionScript* no es muy grande, pero claramente esto depende en qué tipo de animaciones se quieran crear.
Para comenzar, la empresa Adobe quitó el soporte oficial y marcó como obsoleta a *Flash*[5], y planea finalizar su distribución a finales de 2020.

Otro factor negativo es el *SEO*, es decir la ubicación en buscadores como *Google* para el contenido dentro de la película Flash. La forma correcta de usar una película Flash en una página *web* es tener una alternativa de *HTML* en el código *HTML*, tanto por razones de *SEO* como de accesibilidad, y usar *Javascript* para insertar dinámicamente su película *Flash*.

También, otro de los problemas con *Flash* es que está incluido en el navegador *web* como un tiempo de ejecución autónomo completo, lo que significa que funcionará igual en un reproductor *Flash* independiente. El efecto de esto es que si enfoca la película *Flash*, se pierden todos los accesos directos y el enfoque del teclado del navegador web, y debe hacer clic fuera del área de *Flash* para volver a enfocar.

Teniendo en cuenta el análisis anterior, podemos ver que *Flash* tuvo su momento de gloria y fue bastante largo, pero actualmente ya se está haciendo obsoleto por las razones anteriormente mencionadas, por lo que no consideramos que en la actualidad sea una alternativa real y vigente a *d3.js*.

2.3. Comparación de las herramientas

Se entiende interesante resumir la comparación de las principales características utilizadas de las herramientas mencionadas, esta comparación se puede ver en la Tabla 2.1.

[5]https://theblog.adobe.com/adobe-flash-update/

	Características				
	Forces	*Geographies*	*Dragging*	*Zoom*	*Voronoi Diagrams*
d3.js	Si	Si	Si	Si	Si
Raphaël	Si	Si	Si	Si	No
Processing.js	Si	Si	Si	Si	Si
Prefuse	Si	No	Si	Si	No
Adobe Flash y AS	No	Si	Si	Si	Si

Tabla 2.1: Tabla comparativa de herramientas.

Capítulo 3

Selección del caso de estudio

En este capítulo se describe el proceso que lleva a la selección de la encuesta metropolitana de movilidad como caso de estudio para este proyecto.

3.1. Proceso de selección del caso de estudio

Se comenzó con investigación de distintas fuentes de datos, con el fin de encontrar datos interesantes y de un volumen aceptable, que permita realizar visualizaciones de calidad.

Se analizan datos sobre distintas de ciudades alrededor del mundo publicados por la *web Nomadlist* [1]. *Nomadlist* es una *web* que publica indicadores de distintas ciudades del mundo, más de 900, basándose en la calidad para realizar trabajo remoto desde ellas y comparándolas. Si bien estos datos son muy tentadores, acceder a ellos es posible pero no completamente legal ya que se trata de una aplicación con licencia propietaria.

Se estudia el catálogo de datos abiertos del Uruguay [2], el cual contiene diversos conjuntos de datos de distintas agrupaciones e instituciones nacionales. El volumen de los datos aquí publicados es muy grande, y el hecho de ser datos sobre nuestro país que refleja la vida diaria de la población, hace muy interesante su representación.

[1] https://nomadlist.com/
[2] https://catalogodatos.gub.uy

De esta forma se ahonda en la investigación sobre el catálogo de datos abiertos comenzando con un relevamiento de los datos publicados en la web, el cuál se incluye en el anexo de este informe.

Dentro del catálogo se encuentra la Encuesta Metropolitana de Movilidad del año 2016, o Encuesta de Origen Destino, siendo este el caso de estudio seleccionado para el presente proyecto.

Esta selección se debe a la gran cantidad de datos existentes sobre la vida diaria de la población, entre los que se encuentran el movimiento dentro de la ciudad, abriendo la posibilidad de representarlo en un mapa utilizando características geográficas de *d3.js*. También poder representar características de hogares, actividades diarias de personas, etc.

La representación de esta información abre un abanico muy grande de posibles usos productivos, como una mejor planificación del trafico a determinadas horas, modificación de líneas de transporte publico, entre otras.

La Encuesta fue diseñada y la muestra seleccionada por el Instituto Nacional de Estadística junto con expertos extranjeros según [13], hecho que le da respaldo y veracidad a los datos publicados.

El hecho de la existencia de estos datos públicos y la posibilidad de facilitar su interpretación y uso una vez analizados y puestos en contexto mediante visualizaciones interactivas, hacen a la Encuesta de Origen Destino un excelente caso de estudio.

3.2. Encuesta metropolitana de movilidad

La Encuesta metropolitana de movilidad es un relevamiento de datos realizado por Facultad de Ciencias Económicas y de Administración de la Universidad de la República, entre Agosto y Noviembre de 2016, y contó con el apoyo del Programa de Naciones Unidas para el Desarrollo [3].

Fueron consultados 2500 hogares del área metropolitana de Montevideo, obte-

[3] http://www.montevideo.gub.uy/institucional/noticias/en-movimiento/

niendo para la suma de los integrantes de cada hogar, un total de 12531 viajes. El diseño y la implementación fueron financiados por *CAF*(*Banco de Desarrollo de América Latina*)[4].

También se relevaron los medios de transporte que usan los ciudadanos cotidianamente para realizar sus viajes, así como el origen, destino y horas de inicio de estos, y los tiempos de demora.

La encuesta también recogió información sobre hábitos y opiniones acerca de la calidad, oferta y aspectos a modificar del transporte público en el área, así como otros temas de la movilidad.

3.3. Motivación y Metodología de la encuesta

En los últimos años, el Área Metropolitana de Montevideo (AMM) experimentó cambios económicos, sociales y urbanos significativos cuyo efecto sobre la movilidad es indiscutible[13].

La encuesta estuvo a cargo de la Facultad de Ciencias Económicas y de Administración de la Universidad de la República, junto a expertos internacionales.
El equipo se encargó de seleccionar y contactar los hogares a los que se les realizó la encuesta, de tal manera que la información sea estadísticamente representativa del conjunto de la población del Área Metropolitana.

El principal objetivo de la encuesta es conocer en qué condiciones se mueven las personas en la ciudad y cuáles son los principales obstáculos que deben enfrentar.

La encuesta fue de carácter domiciliario[13]: un encuestador identificado visitó los hogares seleccionados en la muestra y realizó una serie de preguntas.
Se recolectó información básica como edad y nivel educativo de todos los miembros del hogar, además de la información de viajes de todos los integrantes del hogar mayores de cuatro años, según se puede apreciar en [14].
La información recabada es confidencial y su tratamiento es exclusivamente estadístico, es decir, que no fuero recabados datos identificativos sobre ninguno de los hogares encuestados, acorde a lo publicado en [13].
El Área Metropolitana incluye la totalidad de Montevideo y localidades de Ca-

[4]http://www.montevideo.gub.uy/institucional/noticias/de-aca-para-alla

nelones (en varios corredores metropolitanos, como el de Ciudad de la Costa o el de La Paz, y de San José, como Ciudad del Plata y Libertad), que presentan una interacción metropolitana importante con la capital.
Mediante los datos brindados por la encuesta se podrán construir indicadores y modelos que arrojen el estado de situación actual, así como proyectar cómo se movilizarán las personas en el futuro bajo diversos escenarios, lo cuál permitirá prever necesidades a mediano y largo plazo en materia de infraestructura, transporte público y ubicación de nuevos servicios, de acuerdo a lo publicado en [15].

Todos los datos recabados en la encuesta no tienen ningún sentido si no somos capaces de contextualizar y visualizarlos adecuadamente, para lo cuál se desarrolla una aplicación *web* utilizando la librería *d3.js*, y así sacar máximo provecho a los datos relevados por la encuesta.

Capítulo 4

Modelado de datos

El modelado de datos consiste en construir un modelo que permita describir los elementos de la realidad que intervienen en un problema dado, y la forma en que se relacionan esos elementos entre sí.

En este capítulo se describe el modelo o estructura de datos utilizada para el caso de estudio elegido, y también su formato y relaciones entre entidades. Se describen también las transformaciones necesarias hasta llegar a los datos listos para ser utilizados por la aplicación. Se abarca la etapa *Parse* descripta en la Sección 2.1.

4.1. Estructura de datos

Los datos crudos a utilizar como dominio para construir la herramienta de visualización son, como ya se mencionó, los de la Encuesta Metropolitana de Movilidad. Consisten en archivos en formato *CSV* y *TXT*. Cada archivo *CSV* está acompañado de su correspondiente diccionario, siendo este un archivo *TXT* que explica el formato y contenido.

El conjunto de datos en formato *CSV* incluido en la encuesta se compone por:

- *Base Hogar Habitos.csv*
- *Base Personas.csv*
- *Base Viajes.csv*

- *Base Etapas.csv*

Estos archivos vienen acompañados por sus respectivos diccionarios:

- *Dicc Base Hog_ Hab.txt*
- *Dicc Base Personas.txt*
- *Dicc Base viajes.txt*
- *Dicc Base Etapas.txt*
- *Dicc Transversales.txt*

Cada base nombrada anteriormente hace referencia a un componente de la Encuesta, la base de Hogares por ejemplo es el archivo *CSV* que contiene toda la información sobre indicadores correspondientes a Hogares. Es importante dejar esto en claro ya que cuando se describe la base de datos más adelante en este informe, se sigue utilizando esta nomenclatura y puede confundirse el termino "base". En el diccionario transversales se explica mediante qué atributos se relacionan cada una de las bases que forman la encuesta de la siguiente manera:

El atributo *NFORM* está presente en todas las bases e identifica a un formulario correspondiente a un hogar, ya que las encuestas se realizan en el contexto de un hogar. Dentro de un hogar existen personas identificadas dentro de este contexto mediante números consecutivos, representados por el atributo *nnper*, único dentro del hogar. Este atributo junto con el identificador de formulario *NFORM*, permite calcular el atributo *IDPER* como *NFORM*100+nnper*, identificando así a la persona globalmente.

Para cada persona pueden existir uno o mas viajes, identificados en el contexto de una persona mediante números consecutivos representados por el atributo *nvj*. Este atributo junto con el identificador global de persona permiten construir el identificador global de cada viaje representado por el atributo *IDPERV*, calculado como *IDPER*100+nvj*.

Análogamente para las etapas de cada viaje, se tiene el número consecutivo de etapa *netp* dentro de cada viaje, y el identificador global de cada etapa se construye como *IDPERVE=IDPERV*100+netp*.

Además se utilizan datos de barrios, localidades, líneas y calles del área metropolitana de Montevideo:

- *barrios.geojson*
- *localidades.geojson*
- *calles.geojson*
- *lineas.geojson*

4.2. Transformación y procesamiento de los datos

Para la creación de la base de datos sobre la cual se basa la solución es necesario modificar el modelo de datos original. Para esto, se realizan varias transformaciones al conjunto de datos inicial, agregando atributos auxiliares, claves foráneas y cambio de unidades, mediante *script*s o pequeños programas, los cuáles se detallan a continuación.

Todos los *script*s para las transformaciones se implementaron en el lenguaje de programación *Python* ya que permite procesar y modificar documentos en formato *JSON* extensos, y los integrantes estamos familiarizados con dicho lenguaje. En el repositorio del proyecto [1] se incluyen todos los datos de entrada originales, los *script*s de transformaciones y sus salidas.

4.2.1. Conversión de archivos de datos de *CSV* a *JSON*

Esta transformación consiste en convertir los archivos en formato *CSV* a formato *JSON*, y de esta forma facilitar las posteriores transformaciones a realizar. Para esto se utiliza la herramienta en línea gratuita *CSV to JSON Converter* [2], la cuál tiene como entrada un archivo *CSV* y como salida el mismo archivo en formato *JSON*.

Los archivos en formato *JSON* son más fáciles de manipular mediante distintos lenguajes de programación, lo que nos permite realizar las transformaciones necesarias de manera más sencilla que hacerlas en su formato original *CSV*.

[1] https://github.com/aguprado/proyecto-grado-18
[2] http://www.convertcsv.com/csv-to-json.htm

4.2.2. Conversión de diccionarios de *TXT* a *JSON*

De la misma manera que los archivos de datos *CSV* se convirtieron en archivos *JSON*, también se creó un archivo *JSON* a partir de los archivos de los diccionarios, originalmente en formato *TXT*, con la diferencia de que la transformación fue realizada a mano.

Una vez obtenidos todos los diccionarios en el formato deseado, se crea una tabla en la base de datos para almacenarlos, donde cada fila corresponde a un diccionario diferente, lo cuál se ve más adelante en este capítulo cuando se describe la base de datos.

4.2.3. Agregado de barrio origen y destino a cada viaje de la base de viajes

Una de las visualizaciones que forma parte de la solución consiste en un mapa de calor con el destino de los viajes a lo largo del día. El mapa sobre el cual se aplica la visualización es un mapa político de Montevideo, dividido en barrios y localidades aledañas a la ciudad, que son parte de los departamentos de San José y Canelones, ya que muchos viajes tienen origen y/o destino en estas localidades.

Para dibujar este mapa, *d3.js* toma como entrada un archivo *GeoJSON* que contiene las coordenadas de cada barrio y localidad aledaña, el cuál es generado a partir de un archivo *shapefile* de los barrios de Montevideo y sus localidades aledañas.

La encuesta de viajes no contiene barrios, sino que tiene la zona censal origen y la zona censal destino de cada viaje. Esta información, aunque es útil, no está expresada de la mejor manera dado que las zonas censales no son utilizadas en absoluto como referencias geográficas, al menos por la mayoría de la gente.

Para poder construir esta visualización se requiere aplicar la primer transformación de la base de viajes: agregar a cada viaje su barrio de origen y destino a partir de la zona censal de origen y destino que incluye cada viaje.

Para lograr este fin surge la necesidad de conocer que zonas censales son parte de cada barrio de la ciudad y localidades aledañas. Tal cálculo se realiza utilizando la herramienta *QGIS*, y realizando un *spatial join*, utilizando el criterio *nearest*, entre un archivo *shapefile* de las zonas censales de la ciudad y el archivo *shapefile* de

barrios y localidades, se obtienen que zonas censales están comprendidas en cada barrio o localidad. Los archivos *shapefiles* de barrios, localidades y zonas censales se obtuvieron desde [3].

Se agrega el arreglo de zonas censales que forman parte de cada barrio y localidad al archivo *GeoJSON* de barrios y localidades descripto anteriormente y se guarda en la base de datos para su posterior utilización. Cabe aclarar que la lista de zonas censales no es utilizada en ningún momento por *d3.js* sino que es utilizada por la lógica que colorea el mapa de calor, como se describe en detalle en el capítulo 7.

4.2.4. Conversión de formato de coordenadas geográficas de *UTM* a *DD*

UTM (Sistema de Coordenadas Universal Transversal de Mercator, en inglés Universal Transverse Mercator) y *DD* (Grados decimales, en inglés Decimal Degrees), son dos unidades mediante las cuales se pueden representar coordenadas geográficas.

Al archivo *GeoJSON* de barrios y localidades se le realizó un cambio de formato a sus coordenadas geográficas, transformándolas de *UTM* a *DD*, ya que *d3.js* utiliza el formato *DD* como entrada de coordenadas geográficas, y los datos de los archivos originales contienen las coordenadas en formato *UTM*.

4.2.5. Agregado de líneas de ómnibus a viajes

También forma parte de la solución una visualización que consiste en mostrar sobre un mapa de la ciudad las líneas de ómnibus, por lo que surge la necesidad de añadir la línea de ómnibus utilizada para cada viaje de la base viajes que haya sido realizado en dicho medio de transporte.

Esta transformación se realiza con la ayuda de la base de etapas. La base de etapas contiene información sobre las etapas de cada viaje utilizando el identificador del viaje como clave foránea, pudiendo tener distintos medios de transporte en cada etapa.

De esta manera, si al menos una etapa del viaje se realiza en ómnibus, se puede

[3] http://www.ine.gub.uy/web/guest/vectoriales

saber que línea de ómnibus se utiliza y a que hora, lo que permite dibujar en el mapa que líneas de ómnibus son utilizadas, cada hora del día.

Si bien el atributo línea es accesible para un viaje a través de sus etapas, simplifica mucho la lógica de la visualización tener la línea directamente en cada viaje.

4.2.6. Simplificación de datos para aplicación de filtros

Una de las características de la aplicación desarrollada es la capacidad de aplicar filtros a los datos que se utilizan como entrada para las diferentes visualizaciones. Para lograr y simplificar esto, se realizan otras transformaciones a los datos.

A la encuesta de viajes se le agrega el identificador de persona para cada viaje, el cual referencia a la persona que lo realiza, y de esta manera se pueden filtrar los viajes en función de características de la persona. Del mismo modo y para la encuesta de hogar hábitos, se le agregó a cada entrada de la base que referencia un hogar, un arreglo de identificadores de personas, que referencia el conjunto de personas que habitan el hogar, y así poder filtrar estos datos según características de personas.

4.2.7. Intersección de líneas y calles

Para realizar la visualización que refleja el impacto de la utilización de las líneas en las calles por las cuáles éstas pasan, es necesario realizar un procesamiento y transformación tanto a las calles como a las líneas.

Utilizando la biblioteca *shapely* de *Python*, se intersectan geométricamente las calles con las líneas, de manera de agregar para cada línea un arreglo con todas las calles por las cuáles esta pasa, y a cada calle un arreglo con todas las líneas que pasan por ella, teniendo en cuenta no solo los tramos de calles por las cuáles las líneas pasan, sino también las calles que se cruzan, ya que se entiende que también se ven impactadas a los efectos de analizar el congestionamiento.

Luego de realizadas todas las transformaciones descriptas, se obtienen los archivos en formato *JSON* mediante los cuales se crea la base de datos relacional que utiliza la aplicación web, esto se explica mas adelante en el informe.

4.2.8. *Scripts* y tiempos de ejecución

En esta sección se describen que transformaciones contienen cada uno de los distintos *scripts* implementados y su tiempo de ejecución. Se puede acceder a ellos en el repositorio del código del proyecto [4].

En la Tabla 4.1 se pueden ver que transformación se aplica en cada uno de los *scripts* implementados.

Transformaciones implementadas por *scripts*	
Transformación	***Script***
Agregado de zonas censales a barrios	transformaciones_barrios_localidades.py
Agregado de zonas censales a localidades	transformaciones_barrios_localidades.py
Unión de *JSON* de barrios con localidades	transformaciones_barrios_localidades.py
Cambio de formato de UTM a DD	transformaciones_barrios_localidades.py
Agrega a cada calle, las líneas que la usan	calles_lineas.py
Agrega a cada línea, las calles que usa	lineas_calles.py
Agrega arreglo de *ids* de personas a hogar	transformaciones_encuesta_hogar.py
Agrega *id* de barrio origen a cada viaje	transformaciones_encuesta_viajes.py
Agrega *id* de barrio destino a cada viaje	transformaciones_encuesta_viajes.py
Agrega *id* de persona a cada viaje	transformaciones_encuesta_viajes.py
Agrega línea de ómnibus a cada viaje	transformaciones_encuesta_viajes.py

Tabla 4.1: Tiempo de ejecución por *scripts*

En la Tabla 4.2 se pueden ver los tiempos de ejecución de los *scripts* ejecutados en una *Macbook Pro i7, 16GB RAM, SSD, con MacOS Mojave.*

[4] https://github.com/aguprado/proyecto-grado-18/tree/master/datos/scripts_transformaciones

Transformaciones implementadas con *scripts*	
Script	**Tiempo de ejecución**
transformaciones_barrios_localidades.py	28,9 segundos
calles_lineas.py	3870,5 segundos
lineas_calles.py	693,3 segundos
transformaciones_encuesta_hogar.py	224,0 segundos
transformaciones_encuesta_viajes.py	2660,8 segundos

Tabla 4.2: Tabla de transformaciones implementadas por *scripts*

Capítulo 5

Diseño de la solución

En este capítulo se presenta el diseño de la solución, a la cual también llamamos herramienta de visualización o aplicación *web* según se adapte mejor al contexto. Se describen los pasos y decisiones tomadas en cada etapa y para cada elemento de la arquitectura de la aplicación, y también el modelado de datos del caso de estudio para hacer posible su utilización.

La herramienta de visualización propuesta se basa en una aplicación *web* típica, con arquitectura cliente servidor sobre la cual se profundiza en las siguientes secciones.

A lo largo de este capítulo también se describe la arquitectura de la solución, explicando cada elemento a alto nivel sin entrar en detalle de su implementación.

Luego se detalla el ambiente de trabajo sobre el cual se construye la solución y cada uno de los elementos de la arquitectura. Además se describen el diseño de la base de datos, las tablas y relaciones que la conforman y finalmente donde y como está desplegada la aplicación.

5.1. Arquitectura de la solución

La arquitectura escogida para construir la solución es una típica Cliente-Servidor orientada a servicios.

Hoy en día, debido a la mayor capacidad de procesamiento de los dispositivos personales, gran parte del este procesamiento se ha ido trasladando al cliente, utilizando el servidor solo como acceso a la base de datos.

En la aplicación construida, el procesamiento está distribuido de manera similar entre el cliente y el servidor.

Parte del procesamiento de los datos ejecutado tanto por el cliente como por el servidor corresponde a la etapa *Mine* descrita en Sección 2.1.

Los datos utilizados por la aplicación son almacenados en una base de datos relacional que convive en la misma máquina virtual que aloja el servidor con los servicios. Si bien es posible tener el servidor de base de datos separado del servidor de servicios, en esta primer versión se decide alojarlos en la misma máquina virtual o nodo, al ser innecesario su desacople, siendo esta opción recomendable cuando existe un elevado volumen de consultas. Queda como trabajo a futuro ejecutar el servidor de base de datos y el servidor en distintos nodos.

5.2. Cliente

El cliente es una aplicación *web* dinámica que corre en un navegador web. Cada visualización es accesible mediante una única *URL*. El diseño de la aplicación *web* es adaptativo. Que el diseño sea adaptativo significa que cambia la organización de los componentes en función de la resolución de la pantalla del dispositivo desde el cual se accede, variando en función de su tamaño.

La aplicación es probada en dispositivos móviles tanto como en dispositivos de escritorio, utilizando los navegadores *Google Chrome*, *Mozilla Firefox* y *Safari* en sus últimas versiones. El dispositivo móvil para probar su correcto funcionamiento es *Apple Iphone 8* con el sistema operativo *iOS versión 12.2* y *Samsung S9* con sistema operativo *Android* es su versión 9.

5.3. Servidor

El servidor provee de servicios que son accedidos por el cliente. Estos servicios son públicos, sin autenticación, sin estado e idempotentes, cada uno accesible mediante una *URL* única. En nuestra solución, el servidor solo expone servicios de

lectura de datos y no de modificación o eliminación.

El servidor se comunica y consume datos de la base de datos, los procesa según corresponda y los envía al cliente.

5.4. Base de datos

En la base de datos se almacenan los datos a ser consumidos por el servidor, quien los procesa y luego envía al cliente *web* para generar las visualizaciones.

5.4.1. Diseño de la base de datos

La representación de las bases de la encuesta en la base de datos se realiza creando una tabla para cada base, es decir una tabla para la base de Hogares, una tabla para la base de Personas, una tabla para la base de Viajes, una tabla para la base de Etapas, y finalmente una tabla para almacenar los distintos diccionarios de la encuesta.

Es importante distinguir una base de la encuesta y la base de datos. Base de datos hay una sola, pero la encuesta tiene muchas bases: Hogares, Viajes, Personas y Etapas. También se crea una tabla para almacenar la información referente a barrios y localidades, a las líneas de ómnibus y a las calles de la ciudad de Montevideo, ya que estos datos se utilizan para la generación de ciertas visualizaciones.

Las tablas incluidas en la base de datos son las siguientes:

- **barrios_y_localidades**: Contiene los datos de barrios y localidades, siendo cada fila un barrio o localidad. Entre sus atributos están las coordenadas geográficas que utiliza *d3.js* para dibujar el mapa en la visualización.
- **encuesta_etapas**: Contiene los datos referentes a la encuesta de etapas, donde cada fila representa una etapa de un viaje dado. El atributo *NFORM* es una clave foránea a la tabla de encuesta de hogares que indica a que hogar pertenece la persona que realiza el viaje.
- **encuesta_hogar**: Contiene los datos referentes a la encuesta de hogar hábitos, donde cada fila representa un hogar, contiene un atributo adicional que referencia a las personas que forman parte del núcleo del hogar.

- **encuesta_personas**: Contiene los datos referentes a la encuesta de personas, donde cada fila representa una persona encuestada. El atributo *NFORM* es una clave foránea a la tabla de encuesta de hogares que indica a que hogar pertenece la persona.

- **encuesta_viajes**: Contiene los datos referentes a la encuesta de viajes, donde cada fila representa un viaje realizado por una persona. Contiene varios atributos adicionales que se explicaron anteriormente en la sección de Transformación de datos: barrio_origen, barrio_destino, id_persona y línea. El atributo *NFORM* es una clave foránea a la tabla de encuesta de hogares que indica a que hogar pertenece la persona que realiza el viaje.

- **encuesta_diccionarios**: Contiene los datos referentes a los diccionarios de cada base de la encuesta: Personas, Etapas, Hogar Hábitos y Viajes, donde cada fila es un diccionario. Los diccionarios para cada encuesta son creados a mano a partir de los archivos en formato *TXT*, incluidos en los datos originales de la encuesta, guardándolos en formato *JSON*. Se modifican algunas preguntas para contextualizarlas. A modo de ejemplo, algunas preguntas solo toman sentido en función de la pregunta anterior, en este caso, a la pregunta se le agrega: "Con respecto a la pregunta anterior..".

- **líneas_calles**: Contiene los datos referentes las líneas de ómnibus con un arreglo de las calles por las que pasa cada línea. El atributo que referencia al nombre de la línea se encuentra dentro del atributo *properties*, que es de tipo *JSON*. El atributo *geometry* es el que utiliza *d3.js* para dibujar el mapa de líneas.

- **calles_líneas**: Contiene los datos referentes a cada cuadra de calle de la ciudad, siendo cada fila de esta tabla una cuadra de una calle determinada. Cada fila también incluye un arreglo de las líneas de ómnibus que atraviesan la calle. El atributo que referencia al nombre de la calle se encuentra dentro del atributo *properties*, que es de tipo *JSON*, compartiéndose el nombre de la calle entre todas las cuadras que forman parte de la misma. El atributo *geometry* es el que utiliza *d3.js* para dibujar el mapa de calles.

En la Figura 5.1 se puede ver un modelo de entidad relación de la base de datos.

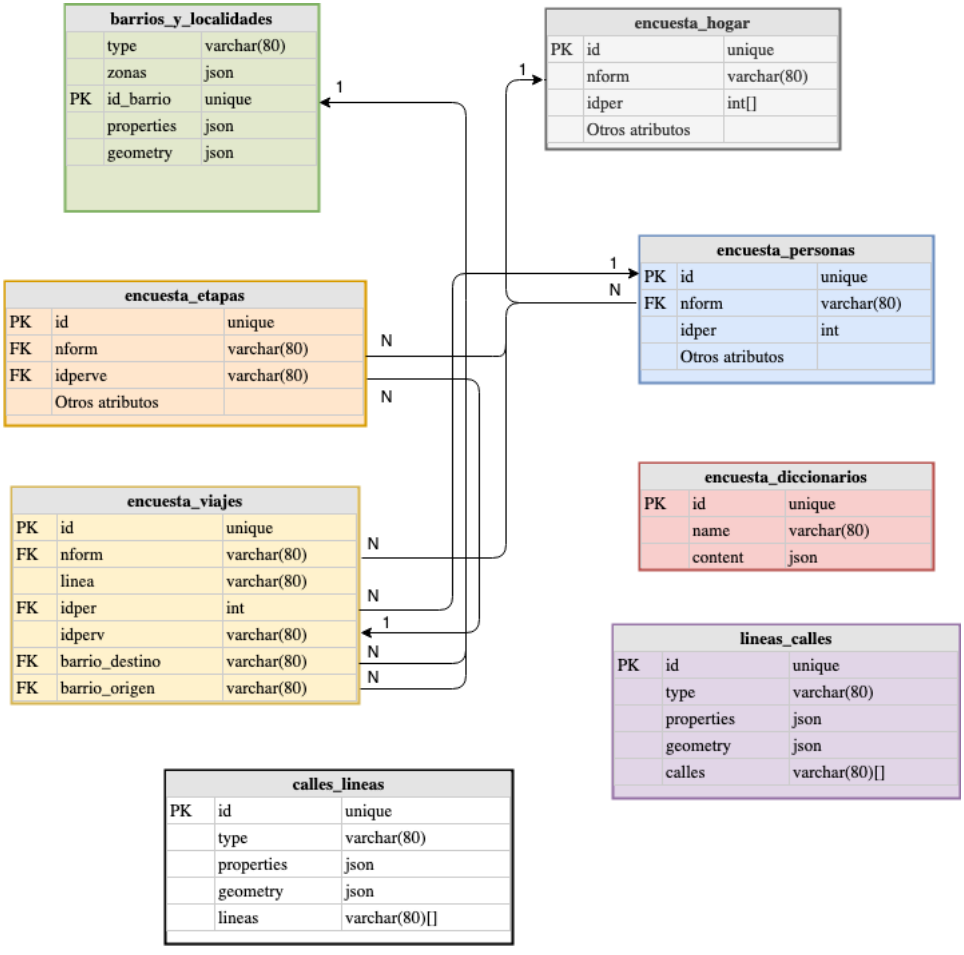

Figura 5.1: Diagrama de base de datos

La creación de carga de la base de datos se describe en el capítulo *Implementación de la solución.*

La representación de los datos mediante una base de datos relacional, con una estructura y formato definido es parte de la etapa de *Parse* descripta en la Sección 2.1.

5.4.2. Soporte para la modificación de datos de entrada

El desarrollo de *script*s para la adaptación del dominio de datos al modelo de base de datos final, da lugar a la posibilidad de modificar estos datos de entrada por una encuesta anterior o futura, siempre y cuando su formato sea el mismo.

De esta manera se pueden crear varias instancias de la aplicación con datos diferentes y así poder comprar la evolución de los indicadores de la encuesta con el tiempo.

Capítulo 6

Implementación de la solución

En este capítulo se presenta la implementación de la solución enfocándose en las tecnologías utilizadas, el ambiente de trabajo para su construcción y el ambiente de ejecución.

6.1. Ambiente de trabajo

Las herramientas utilizadas para la construcción de la aplicación son:

- *IDE*s (Entorno de Desarrollo Interactivo, en inglés *Integrated Development Environment*): *Sublime Text 3*[1] y *Visual Studio Code*[2]

  Son dos interfaces de desarrollo ampliamente utilizadas en la actualidad.

- Qlik [3]: Herramienta de análisis de datos, en la que luego de cargar los datos es posible inmediatamente generar visualizaciones de un conjunto de gráficos disponibles, además de permitir utilizar información procedente de distintas fuentes. Se utiliza *Qlik* ya que brinda la posibilidad de obtener una visión global sobre los datos de la encuesta, investigarlos y ver un panorama general de las respuestas.

[1] https://www.sublimetext.com/3
[2] https://code.visualstudio.com/
[3] https://www.qlik.com/

- QGIS [4]: Sistema de Información Geográfica (SIG) de código libre, que permite manejar formatos vectoriales. Utilizando el *plugin spatial join* podemos realizar combinaciones espaciales entre capas vectoriales y añadir atributos de una capa vectorial a otra en función de las intersecciones entre ellas.

El repositorio del código de la aplicación se aloja en *Github*, siendo este un repositorio público en una cuenta personal de los integrantes del equipo.[5]

6.2. Implementación

En esta sección profundizaremos en la implementación de los elementos que forman parte de la arquitectura de la solución: Cliente, Servidor y Base de Datos.

6.2.1. Cliente

El cliente está construido utilizando el *framework web Angular 5.2*[6] basado en componentes, escrito en *Typescript* y siendo la versión 5.2 la última estable disponible al comenzar el desarrollo del proyecto.

El fundamento detrás de la elección de este *framework web* es la experiencia personal de los integrantes del equipo con dicho *framework*, y el hecho de que es uno de los mejores disponibles junto con la existencia de un módulo oficial para Angular que provee la interfaz a *d3.js*.

Se utiliza *angular-cli*, siendo esta una herramienta que provee una interfaz de línea de comandos para ayudar en la creación rápida de código *boilerplate* para proyectos, componentes, servicios y demás.

La interfaz de usuario está construida siguiendo lineamientos básicos sobre usabilidad web. Para ello se utiliza el *framework* para construir interfaces *Bootstrap* en su versión más reciente, la 4.0, y también *JQuery* para hacer manejo del *DOM* de la web, ambos módulos auxiliares son elegidos debido a la experiencia de los integrantes del equipo con ellos.

[4] https://www.qgis.org
[5] https://github.com/aguprado/proyecto-grado-18
[6] https://angular.io/

El *ruteo* de la aplicación está implementada utilizando el modulo *Router* de *Angular*, y cada visualización tiene una ruta propia para acceder directamente mediante una *URL*.

Cada visualización está implementada en un componente *web* creado para este fin; el cabezal también esta implementado como un componente separado. Cada componente se crea utilizando un comando de la herramienta *angular-cli* y que genera el código *boilerplate* para el componente, y este consta de una carpeta que contiene los siguientes archivos:

- Un archivo en formato *HTML* que implementa la vista del componente.
- Un archivo en formato *ts* que implementa la lógica del componente.
- Un archivo en formato *CSS* que contiene los estilos particulares del componente.

La aplicación consta de un archivo global de estilos en cascada *CSS* que contiene los estilos globales de la aplicación, es decir, que no son particulares de cada componente.

La aplicación *web* contiene componentes llamados servicios. Estos son archivos en formato *ts* que definen funciones auxiliares que proveen de funcionalidades compartidas y utilizadas por todos o algunos de los componentes, como son acceso a las funciones para mostrar u ocultar el *loading spinner*, o las funciones para aplicar filtros a los datos, y finalmente la interfaz para acceder a los servicios expuestos por la *API REST*.

Los servicios existentes en la aplicación son:

- Interfaz de acceso a los servicios de filtros.
- Interfaz de acceso a los servicios para mostrar u ocultar el *loading spinner*.
- Interfaz de acceso a los servicios *REST*.

También existe un archivo de configuración en formato *ts* donde se define la *URL* del servidor, es decir la *URL* para acceder a los servicios *REST* implementados.

6.2.1.1. Interfaz gráfica

Utilizando el *framework Bootstrap* se implementa una típica interfaz web, con un cabezal o *header*, una barra lateral o *sidebar* y un *footer* según el ancho de la

pantalla del dispositivo desde donde se acceda.

El diseño responsivo se implementa utilizando la característica de columnas de *Bootstrap*.

Cabezal: desde el cabezal se pueden acceder a las diferentes visualizaciones disponibles, agrupadas en categorías, donde cada categoría es representada mediante un *dropdown*, o menú desplegable, desde donde se accede mediante un enlace a cada visualización.

También en el cabezal se incluye un *dropdown* para la selección de filtros a aplicar a los datos de entrada en caso de que corresponda, y también un menú desplegable con enlaces a la ayuda para utilizar la aplicación.

En el capítulo Visualizaciones Construidas, se profundiza sobre las funcionalidades incluidas en la barra lateral para cada visualización.

En la Figura 6.1 se puede ver la parte izquierda del cabezal de la aplicación con todas las categorías colapsadas a excepción de la categoría Viajes que está desplegada.

En la Figura 6.2 se puede ver la parte derecha del cabezal de la aplicación con el menú desplegable de Filtros desplegado, al lado del menú desplegable de Ayuda.

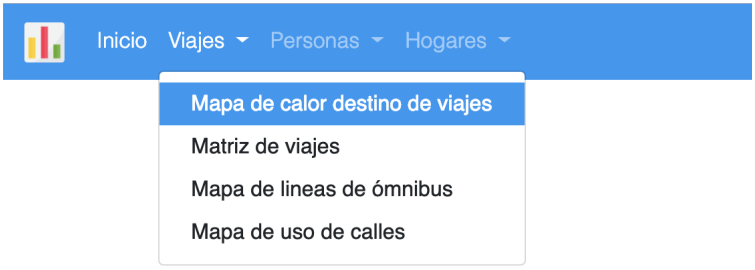

Figura 6.1: Parte izquierda del cabezal

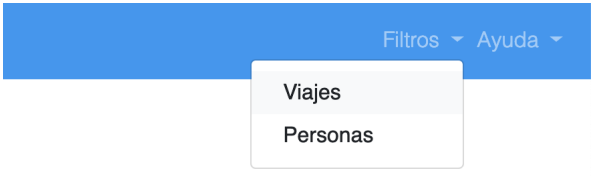

Figura 6.2: Parte derecha del cabezal

Para dispositivos móviles, se reemplazan los menú desplegables del cabezal por un *menú hamburguesa* típicamente utilizado en dispositivos móviles, desde donde se puede acceder a todas las categorías.

En la Figura 6.3 se puede ver el menú hamburguesa desplegado, y en la Figura 6.5 se lo puede ver colapsado. Arriba a la izquierda en ambas figuras se puede ver el icono o logo de la aplicación, haciéndole clic se accede a la pantalla de inicio.

Barra lateral: La interfaz consta de una barra lateral donde se exponen diferentes funcionalidades para interactuar con las visualizaciones según corresponda. Aparte de las funcionalidades para la interacción, también se muestran los filtros aplicados, con la posibilidad de eliminarlos.

En la Figura 6.4 se puede ver la barra lateral para la visualización de mapa de calor de destino de viajes.

Footer: Para las pantallas pequeñas, es decir dispositivos móviles, se sustituye la barra lateral por un *footer* desplegable, colapsado por defecto, el cual aumenta su altura al hacer "clic", quedando al descubierto su contenido. El *footer* tiene el 100 % del ancho de la pantalla, dejando también todo el ancho de la pantalla disponible para mostrar las visualizaciones.

Los mismos controles para interactuar con las visualizaciones que se muestran en la barra lateral son mostrados en el *footer* cuando corresponde, a excepción de los filtros aplicados.

En la Figura 6.5 se puede ver el *footer* colapsado, y, en la Figura 6.6, desplegado para la visualización de mapa de calor de destino de viajes.

Figura 6.3: Menú hamburguesa desplegado

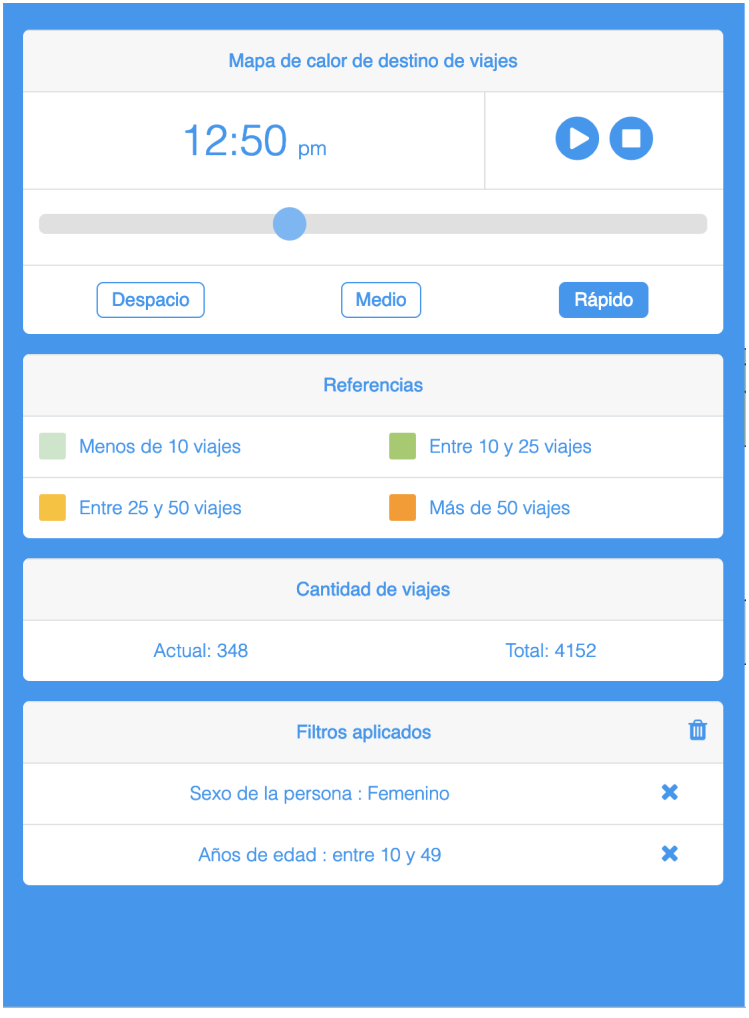

Figura 6.4: Barra lateral con filtros seleccionados

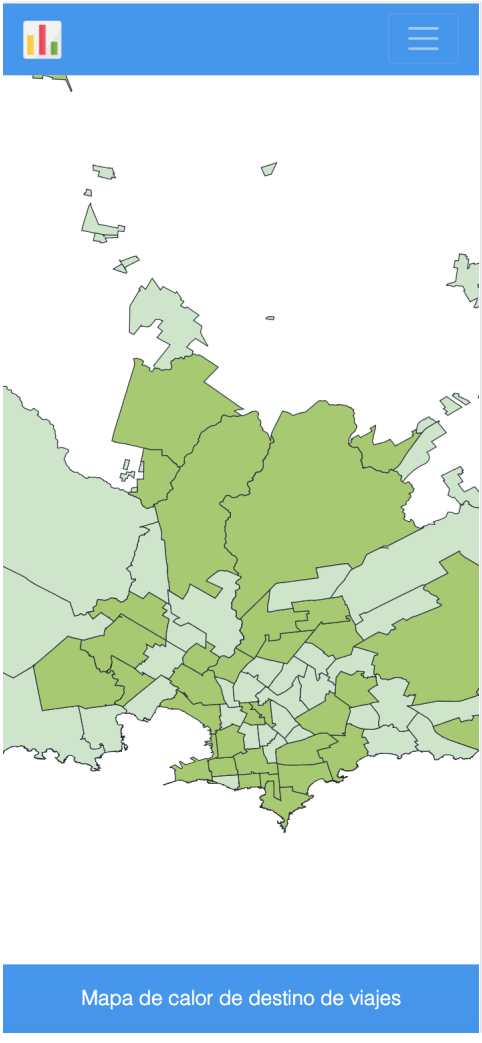

Figura 6.5: Footer colapsado

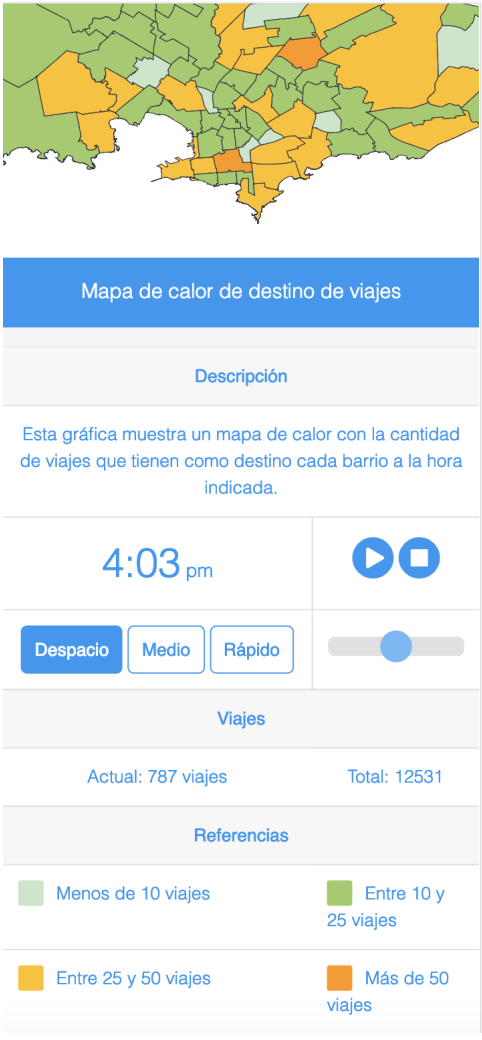

Figura 6.6: Footer desplegado

6.2.1.2. Funcionalidad de filtros

Aquí se describe el funcionamiento de la funcionalidad de filtros del dominio de datos para las visualizaciones. Esta funcionalidad es parte de la etapa *Interact* descripta en la Sección 2.1.

La funcionalidad de filtros permite al usuario aplicar las visualizaciones interactivas a un subconjunto del dominio de datos. Consiste en aplicar diferentes valores a las variables de cada encuesta, definidas en los diccionarios de cada una. Los filtros disponibles son para la base de Hogares, Viajes y de Personas, dado que cada encuesta se realiza a una persona, en el contexto de un hogar, que a su vez esta conformado por varias personas.

En la Figura 6.2 se puede ver que se accede a la funcionalidad de filtros desde el menú desplegable *Filtros* desde la parte derecha del cabezal. En función de la visualización en la que se esté, se puede acceder a diferentes filtros, en la tabla 6.1 se puede apreciar que visualizaciones aceptan cada filtro.

Cada filtro se representa mediante un *modal* con sus opciones disponibles, utilizando radios para su selección en caso de ser una opción de una lista, sexo por ejemplo, y *checkbox*es con *input* de tipo numérico si corresponde a un rango de valores numéricos, de edad por ejemplo. Para que los *input* de tipo rango estén activos, aparte de ingresarles valor, hay que activarlos haciendo clic en su correspondiente *checkbox*, ver Figura 6.7.

Una vez seleccionado un filtro, se vuelven a cargar los datos desde el servidor, mediante una llamada al servicio correspondiente, enviando mediante *queryString* los filtros seleccionados. Se actualiza la barra lateral con los nuevos datos, como se puede apreciar en la Figura 6.4.

A nivel de código, el filtro aplicado en un momento dado es global para toda la aplicación, aplicándose a las visualizaciones según corresponda. Para esto se creó un *service* que controla los filtros: *filter.service.ts*, siguiendo un patrón *Singleton*.

Este *service* almacena el filtro actual, y provee funciones para obtenerlo y modificarlo. Implementa un patrón *Observer* de subscripción para que al modificarse el filtro actual, se pueda notificar a los subscriptores (las visualizaciones) y pedir nuevamente los datos al servidor con los nuevos filtros aplicados.

Filtros disponibles por visualización			
Visualización	Filtro Personas	Filtro Viajes	Filtro Hogares
Inicio	No	No	No
Mapa de calor destino de viajes	Si	Si	Si
Matriz de viajes	Si	Si	Si
Mapa de líneas de ómnibus	Si	Si	Si
Mapa de uso de calles	Si	Si	Si
Actividades de personas	Si	Si	Si
Medios de viaje - Barras	Si	Si	Si
Medios de viaje - Dona	Si	Si	Si
Características de hogares 1	Si	No	Si
Características de hogares 2	Si	No	Si

Tabla 6.1: Tabla de filtros disponibles por visualización

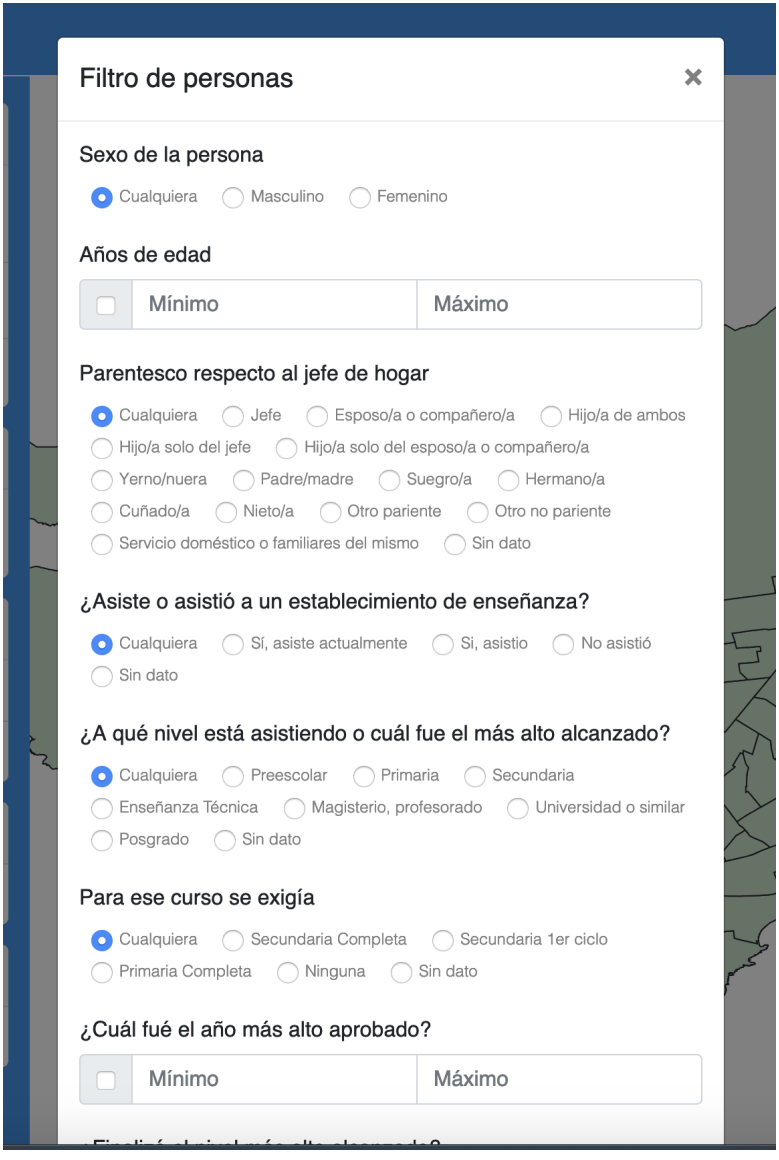

Figura 6.7: Modal de filtros de personas

6.2.2. Servidor

Como se menciona anteriormente, el servidor de la herramienta construida está escrito en *Javascript* utilizando el *framework Express* para construir aplicaciones *web* en un entorno de ejecución *NodeJS*.

La selección de este entorno y lenguaje está dada por varios factores. Primero que nada la posibilidad de libre elección de tecnologías para este proyecto, segundo experiencia personal de los autores de este trabajo, y tercero la sencillez con la cual es posible crear un servidor *API REST* con *Express* en *NodeJS*.

Al ser *Javascript* un lenguaje interpretado, hace que sea muy rápida y fácil su ejecución, sin tiempos de compilación ni generación de ejecutables intermedios. También el hecho de poder escribir el cliente y el servidor en el mismo lenguaje es un factor a tener en cuenta.

Otra alternativa es el lenguaje *Python* que también es interpretado y rápido, que se descarta debido a la mayor experiencia de los integrantes del equipo con *Javascript*, *Express* y *NodeJS*.

El servidor construido, define todos los servicios necesarios para que el cliente los consuma y a partir de ellos generar las visualizaciones, aplicar filtros, etc. A continuación se provee una lista junto con su descripción:

- ***/api/barrios***: Este servicio retorna un arreglo de los barrios de la ciudad de Montevideo presentes en la tabla *barrios_y_localidades*.
- ***/api/barrios-localidades***: Este servicio retorna un arreglo de los barrios de la ciudad de Montevideo y localidades aledañas, presentes en la tabla *barrios_y_localidades*.
- ***/api/base/hogares***: Este servicio retorna los datos de la tabla *encuesta_hogar* utilizados por las visualizaciones de hogares, aplicando los filtros recibidos mediante *queryString*. Los filtros aplicados son Hogares y Personas. Los datos que envía son el total de hogares que cumplen con el filtro aplicado junto con información sobre las características de hogares mostradas, para cada característica se envía la cantidad de hogares que cumplen con cada opción. Por ejemplo para la característica "Aire acondicionado" se envía la cantidad de hogares que cumplen con las opciones "Si", "No" y "Sin Dato".

Las características de hogares devueltas son:
`Tipo de vivienda, Es propietario, Material, Baño, Aire acondicionado, Computador, Acceso a internet, Lavarropa, Bicicleta, Ciclomotor, Automóvil, Garage, Integrantes del hogar, Servicio domestico`

- ***/api/base/viajes***: Este servicio retorna las entradas de la tabla encuesta_viajes aplicando los filtros recibidos mediante *queryString*. Los filtros aplicados son Viajes, Hogares y Personas.

- ***/api/base/viajes/actividades-personas***: Este servicio retorna la actividad diaria para cada persona que tenga al menos un viaje registrado en la encuesta. La actividad diaria está compuesta por un arreglo de viajes de la tabla encuesta_viajes, devolviendo para cada uno la hora de comienzo y propósito o destino del mismo, de esta manera se puede construir la actividad de la persona a lo largo de su día. Aquí también se aplican filtros de Viajes y Personas.

- ***/api/base/viajes/medios-viajes-personas***: Este servicio retorna el medio de transporte junto con la hora para cada viaje de la tabla encuesta_viajes en el día de cada persona, aplicando los filtros de Viajes y Personas.

- ***/api/base/diccionarios***: Este servicio retorna los diccionarios existentes en la tabla encuesta_diccionarios.

- ***/api/estadisticas/base***: Este servicio genera y retorna la información global sobre la encuesta, es decir cada indicador junto con sus resultados, para las tablas de personas, viajes, etapas y hogares. Este servicio no recibe ni aplica filtros, ya que devuelve todos los datos existentes en las tablas.

- ***/api/calles***: Este servicio retorna la información geográfica junto con las propiedades de cada cuadra de cada calle de la ciudad de Montevideo, para que *d3.js* las dibuje en un mapa.

- ***/api/líneas***: Este servicio retorna la información geográfica junto con las propiedades de cada línea de ómnibus de la ciudad de Montevideo, para que *d3.js* las dibuje en un mapa.

Todos los servicios definidos son sin estado, es decir que no dependen de un estado especifico de la aplicación creado por una acción anterior para ser consumidos por

el cliente.

También son todos *GET*, ya que el servidor solo expone los datos de la encuesta y en ningún momento se pueden editar mediante servicios *POST*, *PUT*, *PATCH* o *DELETE*. Al ser todos los servicios *GET*, todos son idempotentes, es decir que el estado de la aplicación no cambia en función de cuantas veces se llame al servicio.

Como se menciona anteriormente, cuando se desean aplicar filtros a los datos solicitados, los parámetros de los filtros se agregan mediante una *queryString* en las *URL* de acceso a los servicios, y de esta forma el servidor interpreta y aplica los filtros solicitados a los datos retornados.

Los servicios definidos no tienen seguridad, su acceso es público ya que se consideró innecesario agregarle algún tipo de autenticación para consumirlos para esta primer versión de la aplicación, y esto queda como trabajo a futuro junto con el inicio de sesión, esto volverá a ser tratado en el Capítulo 8.

La comunicación con el Cliente se da mediante *HTTP*, y no mediante *HTTPS*, quedando también esta característica como trabajo a futuro.

El servidor se comunica con la base de datos mediante el módulo *pg* para *NodeJS*, el cual provee una interfaz *Javascript* para realizar consultas *PostgreSQL* a la base de datos, más detalle sobre la base de datos y la elección de tecnologías se dan en la sección Base de Datos.

Otros módulos auxiliares que se utilizaron para construir y correr el servidor son:

- *Squel*: Herramienta que provee una interfaz simple e intuitiva para construir consultas *SQL* en *Javascript*.
- *Forever*: Herramienta que permite ejecutar un servidor en *NodeJS* y se asegura de que se vuelva a ejecutar si por alguna excepción se corta su ejecución.
- *Winston*: Herramienta que provee una interfaz simple e intuitiva para crear *log* en tiempo de ejecución sobre el estado del servidor.

6.2.3. Base de datos

La base de datos se construyó a partir de los datos de la encuesta. Es una base relacional y el motor utilizado para construirla es *PostgreSQL*. El fundamento de-

trás de la elección de una base relacional es la necesidad de representar cada base de la encuesta: Viajes, Personas, Etapas, Hogares y Diccionarios, como entidades, y también representar sus relaciones junto con sus claves foráneas. De esta forma se simplifica la aplicación de filtros, pudiendo hacer *JOIN* entre las distintas entidades.

6.2.3.1. Creación y carga inicial de datos

La base de datos se crea mediante un *script* en formato *SQL* que está incluido en el repositorio *Github* del código del proyecto [7]. Este *script* borra la base de datos, llamada *encuesta* y crea una nueva con las tablas vacías, creando también las claves foráneas, en nuestra base de datos no utilizamos índices, quedando esto como un trabajo a futuro.

Luego de realizar las distintas transformaciones mediante el *script* correspondiente, explicado anteriormente en el Capítulo 4, se procede a cargar la base de datos con los datos iniciales.

La carga inicial de datos se realiza mediante un *script* escrito en *Javascript* y ejecutado en el entorno de ejecución *NodeJS*. Este *script* lee los archivos en formato *JSON* que son salida de los *script*s de transformación y carga la base de datos vacía con ellos. Este *script* se puede encontrar en el repositorio del proyecto[8], y su tiempo de ejecución es 35,585 segundos en *Macbook Pro i7, 16GB RAM, SSD, con MacOS Mojave*.

Se carga de a una tabla a la vez, en un orden predefinido para respetar la creación de las claves foráneas. A modo de ejemplo, no se puede crear entradas para ninguna tabla antes de cargar la tabla *encuesta_hogar* ya que todas las demás tablas tienen una clave foránea con *encuesta_hogar*, el atributo NFORM.

6.3. Despliegue de la aplicación

El despliegue de la aplicación se realiza en la nube de *Amazon*: *AWS*.

AWS es actualmente el servicio de almacenamiento y distribución en la nube

[7] https://github.com/aguprado/proyecto-grado-18/blob/master/servidor/base_de_datos/crear_db.sql
[8] https://github.com/aguprado/proyecto-grado-18/blob/master/servidor/base_de_datos/cargar_db.js

más usado[9]. En su *suite* de aplicaciones hay cientos de herramientas disponibles para múltiples propósitos. Provee máquinas virtuales con todo tipo de recursos.

Para publicar nuestro servidor *API REST* y servidor de base de datos, utilizamos una de las versiones más básicas y económicas de las máquinas virtuales ofrecidas por *Amazon*, una *EC2 T2 Micro*. Ésta consta de 1GB de RAM, Procesador de 1 núcleo. El almacenamiento de la máquina virtual se selecciona en forma separada, nuestra máquina virtual tiene 12GB de almacenamiento.

El despliegue de la aplicación se realiza utilizando la herramienta *Github*, de forma que cada vez que se quiere desplegar al servidor de la aplicación, se descarga la última versión del código existente en el repositorio *Github* desde la instancia donde está corriendo en *AWS*, accediendo mediante *ssh* y ejecutando el comando `git pull`. Una vez que el código está actualizado, se vuelve a correr el servidor con la herramienta *forever*, descripta a continuación.

Esta tarea se puede automatizar utilizando herramientas de integración continua y despliegue continuo, quedando esta tarea como trabajos a futuro.

El servidor se corre utilizando la herramienta *forever*. Esta se encarga de que el servidor se vuelva a correr si ocurre una excepción en tiempo de ejecución y ésta se corta.

El servidor escucha solicitudes en el puerto 8080 mediante *HTTP*, la configuración de seguridad de la máquina virtual debió ser debidamente modificada para permitir el acceso a este puerto desde cualquier *IP*. El uso de *HTTPS* para la comunicación con los servicios *web* queda también como trabajo a futuro.

El código del *Frontend* está desplegado en un contenedor *S3* de *Amazon*. Un contenedor *S3*, es un servicio de la suite de *Amazon* que permite almacenar y distribuir archivos en la nube fácilmente y de manera muy económica. Los archivos pueden ser publicados de manera privada, pública o con acceso personalizado.

Para publicar el *Frontend* o Cliente primero se crea un *bundle* de la aplicación. Un *bundle* de la aplicación es una versión del código comprimido y minimizado, pronto para desplegar a producción y ser consumido por usuarios finales o *testers*.

[9] https://www.silicon.es/amazon-microsoft-y-google-protagonistas-absolutos-de-la-infraestructura-cloud-2379625

El *bundle* se compone del archivo *index.html*, de los *scripts Javascript*, de las hojas de estilo, de los recursos multimedia y otros recursos que utiliza la aplicación, como ser fuentes, etc. Esto se lleva a cabo mediante el comando `ng build -prod` de *angular-cli*, la herramienta que se describe en el Capítulo 6, ésta hace uso interno de otra herramienta: *webpack*.

Webpack es una herramienta que modifica y comprime el código, entre otras funcionalidades, haciéndolo más *portable*, menos pesado y menos *debuggeable*, aumentando su seguridad. Este proceso es transparente para el usuario que ejecuta el comando que genera el *bundle* y es una característica muy útil de *angular-cli*.

El *bundle* es el que se almacena en S3 de manera pública para que los usuarios lo puedan acceder, una de las características de S3 es la posibilidad de ser configurado para que aloje sitios *web* estáticos.

El alojamiento del *Frontend* de la aplicación también podría haber sido hecho en la instancia de *EC2 T2 Micro* donde se encuentra desplegado el servidor en que corre en *NodeJS*. En este caso el *Frontend* podría ser públicado utilizando un servidor *web* como ser *apache* o *nginx*.

Se opta por la opción del S3 por varias razones:

- Para no consumir recursos de la instancia *EC2 T2 Micro* innecesariamente.
- La instancia *EC2 T2 Micro* tiene un ancho de banda muy limitado al ser una versión económica, por lo que servir de archivos estáticos no vale la pena teniendo la posibilidad de usar otras herramientas.
- El acceso a archivos alojados en S3 es mucho más rápido dado que este es su propósito.
- La configuración de S3 para alojar un sitio *web* es mucho más rápida y se hace mediante una interfaz de usuario. La configuración de un servidor *web* para alojar el sitio en una máquina virtual es más engorrosa.

En la *suite* de *Amazon* también existe la herramienta *Cloudfront*. *Cloudfront* es una *CDN* (Red de distribución de contenido o *Content Delivery Network* en Inglés), que permite replicar el contenido de un S3 mundialmente para una mayor velocidad de acceso al contenido publicado en el. También provee la útil funcionalidad de servir los archivos mediante *HTTPS*, utilizando un certificado mismo de

Cloudfront o uno propio. El uso de *Cloudfront* para este proyecto no se consideró necesario debido a que el acceso a la aplicación, al menos en principio, será solamente desde nuestro país.

Los enlaces de acceso a la aplicación son:

`http://proygrado18.s3-website-sa-east-1.amazonaws.com`

para el cliente web, y

`http://52.67.131.86:8080/api`

para el servidor *API REST*.

Capítulo 7

Visualizaciones construidas

En este capítulo se describen las visualizaciones obtenidas así como las características de *d3.js* utilizadas en cada una. Las visualizaciones junto con las interacciones con usuarios forman parte de las etapas *Represent*, *Refine* e *Interact* descritas en Sección 2.1.

Se utilizaron ejemplos publicados en la web oficial de *d3.js*[1] y en la web *Blocks*[2] (también creada y mantenida por el creador de *d3.js*), para la construcción de las visualizaciones de Inicio, Hogares y Personas, según se agrupan en el listado a continuación.

7.1. Listado de visualizaciones

1. Inicio - Resumen de encuesta
2. Viajes:
    - Mapa de calor destino de viajes
    - Matriz de viajes entre barrios
    - Mapa de líneas de ómnibus
    - Mapa de uso de calles

[1] https://github.com/d3/d3/wiki/Gallery
[2] https://bl.ocks.org

3. Hogares:
    - Características de hogares 1
    - Características de hogares 2
4. Personas:
    - Actividades de personas
    - Medios de viaje

7.2. Descripción e implementación

En esta sección se describe cada visualización construida, abarcando también su implementación, y, como se mencionó anteriormente, detallando que características de *d3.js* se utilizan.

7.2.1. Inicio - Resumen de encuesta

Esta visualización se encuentra en la raíz de la aplicación, es decir que se accede a ella cuando se ingresa a la *URL* base del sitio, o cuando se intenta acceder a una *URL* inexistente. Esta consta de un *carrusel* o *slider*, construido con la herramienta *Bootstrap*, donde cada *slide* del mismo contiene una gráfica de tipo dona o *donut* que representa una base de la encuesta: Viajes, Personas, Hogares.

En cada dona se visualizan todas las preguntas para determinada base, siendo posible seleccionar la pregunta que se quiera visualizar, mediante controles en la barra lateral de la aplicación. Desde aquí mismo también se puede navegar entre bases, y leer una descripción de la visualización, como muestra la Figura 7.1

Una vez seleccionada la base y la pregunta se pueden visualizar sus resultados, es decir qué porcentaje de la población encuestada respondió cada opción disponible, aquí cada *slice* o trozo de la dona representa una opción posible de la pregunta.

Para cada *slice* de la dona se agrega un arco con el nombre de la opción junto con su porcentaje en paréntesis, como se puede apreciar en la Figura 7.2.

Figura 7.1: Barra lateral para la visualización de inicio, con controles

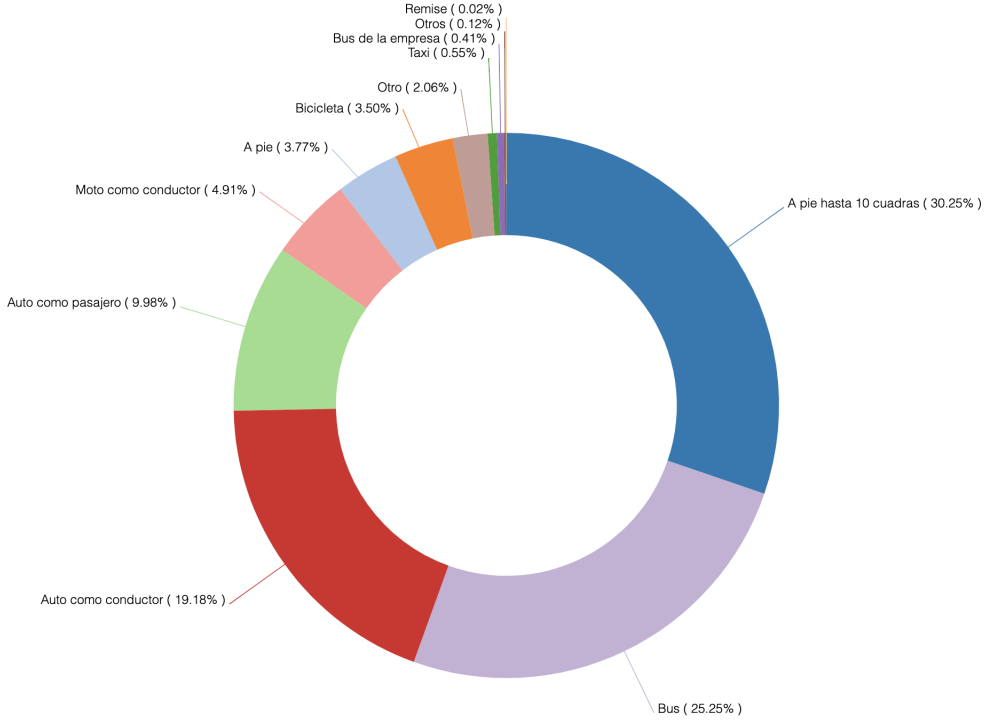

Figura 7.2: Gráfica de la base Viajes mostrando el modo principal de viaje

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consume el servicio */estadisticas/base* el cual retorna los datos a visualizar. Una vez que el servicio retorna esta información, se procede a crear el *carrusel* con un *slide* para cada base y se dibuja la gráfica dentro de cada *carrusel*, tomando la primer pregunta como pregunta seleccionada.

Los pasos para dibujar cada gráfica de dona son los siguientes:

1. Se limpian los elementos *HTML* contenedores para contemplar el caso de que se este volviendo a dibujar la gráfica, mediante las siguientes líneas de código:

   ```
   art = d3.select("$selector .art").html(""),
   labels = d3.select("$selector .labels").html("");
   ```

2. Se llama a la función de *d3.js* `d3.layout.pie()` que construye una nueva función *pie* con ángulo de inicio (0) y ángulo de final (2π), entre otros parámetros. Se aplica a continuación esta función al dominio de datos de entrada para así transformarla a modo legible para *d3.js* y se almacena en la variable *pieData* mediante las siguientes líneas de código:

   ```
   let d3Pie = d3.layout.pie();
   let pieData = d3Pie(data);
   ```

3. Se crea la función que representa el arco de la gráfica con las dimensiones radio interior y radio exterior y se almacena en la variable *pieArc*, mediante las siguientes líneas de código:

   ```
   let pieArc = d3.svg.arc().innerRadius(innerRadius)
   .outerRadius(outerRadius);
   ```

4. Se crea cada cuña o *slice* de la gráfica en función de los datos de entrada almacenados en la variable *pieData* y se almacena en la variable *enteringArcs* mediante la siguiente función:

   ```
   let enteringArcs = art.selectAll("$selector .wedge")
   .data(pieData).enter();
   ```

5. A cada cuña presente en la variable *enteringArcs* se le agrega la clase *wedge* y se le agregan los datos para dibujar, devueltos por la función *pieArc* correspondiente a la *pieData* de la cuña. Finalmente se le asigna el color, utilizando las siguientes líneas:

```
enteringArcs.append("path")
.attr("class", "wedge")
.attr("d", pieArc)
.style("fill", function(d, i){ return colors(i)})
```

6. Se crea una línea para cada cuña, perpendicular a la tangente que pasa por su arco, para ello se definen los puntos x1, y1 y x2, y2 que definen la línea.

7. A continuación se crean los textos con descripción y porcentaje de la opción, y se agregan al final de la línea.

8. Finalmente se aplica la función `relax()` a las líneas para alejarlas del arco mientras se superpongan con otras.

Las características de *d3.js* utilizadas son: `d3.layout.pie()`, `d3.svg.arc()`.

7.2.2. Mapa de calor destino de viajes

Esta visualización muestra información de base viajes de la encuesta sobre el destino de los viajes realizados en el día. Se puede ver un mapa político de Montevideo, divido por barrios, donde cada barrio se colorea con un color determinado en función de los viajes que lo tienen como destino, a la hora indicada por los controles interactivos, según se puede ver en la Figura 7.4.

Las referencias de los colores se pueden ver en la barra lateral, se puede interactuar con la visualización pausando, resumiendo o deteniendola con los controles, también se puede navegar en el día moviendo el deslizador de hora, según se puede ver en la Figura 7.3.

En la sección de la barra lateral se puede apreciar también la cantidad de viajes existentes a la hora indicada, así como la cantidad de viajes totales luego de aplicar los filtros seleccionados.

Los filtros posibles para esta visualización son: Viajes, Personas, Hogares. Se pueden ver y remover los filtros seleccionados mediante la sección correspondiente en

la barra lateral.

También, cuando se posa el cursor del ratón sobre un barrio del mapa se puede leer su nombre.

Figura 7.3: Barra lateral para la visualización de mapa de calor de destino de viajes

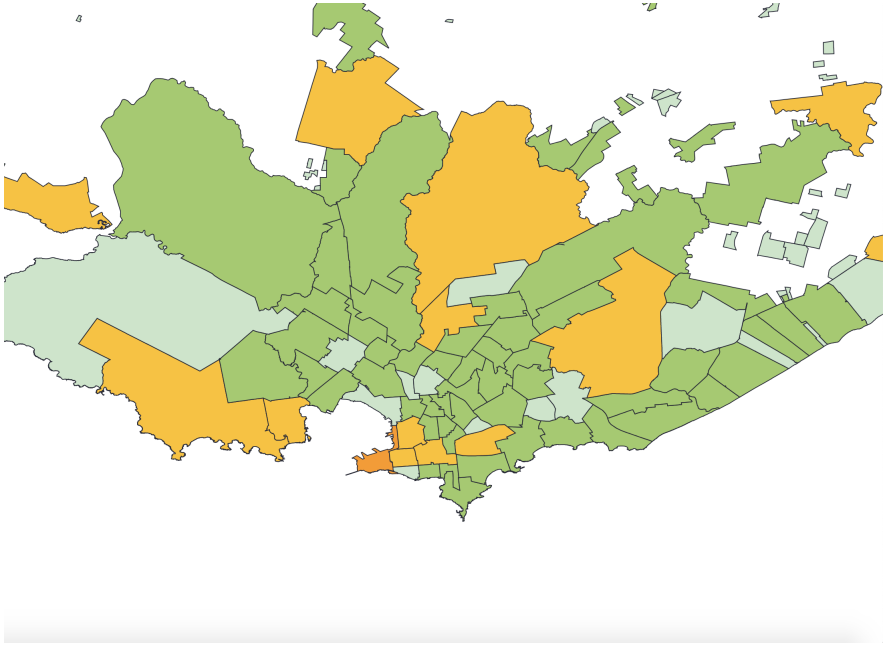

Figura 7.4: Visualización de mapa de calor de destino de viajes

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consume el servicio */barrios-localidades* el cual retorna los datos del mapa para que *d3.js* los dibuje, y */base/viajes* que retorna los viajes para visualizar, aplicando los filtros correspondientes.

También se consume el servicio */base/diccionarios* que retorna los diccionarios a utilizar por los filtros. Una vez que los servicios retornan esta información, se procede a dibujar el mapa. Para esto se utiliza el *GeoJSON* de barrios y localidades devuelto por el servicio anterior. Los pasos para dibujar el mapa son los siguientes:

1. Se crea la proyección del mapa usando pasando las coordenadas de Montevideo como centro y la escala 1:100000 para que se vea la ciudad, se guarda el resultado en la variable `projection`, mediante el siguiente código:

   ```
   let projection = d3
   .geo.equirectangular()
   .center([-56.15, -34.8])
   .scale(100000);
   ```

   La escala corresponde linealmente a la distancia entre dos puntos proyectados.

2. Utilizando la proyección creada anteriormente, se crea el generador geográfico y almacena en la variable `path`, utilizando las siguientes líneas de código:

   ```
   let path = d3.geo.path().projection(projection);
   ```

3. Se procede a crear un contenedor `g` dentro del SVG que contendrá al mapa, agregándole el atributo `id = map` y almacenándolo en la variable `barriosGroup`, mediante la siguiente línea:

   ```
   let barriosGroup = svg.append("g").attr("id", "map");
   ```

4. A continuación se construyen sobre el contenedor `barriosGroup` anterior los limites de barrios utilizando el *GeoJSON* de barrios y localidades y el generador geográfico creado en el paso 3) y almacenado en la variable `path`,

mediante las siguientes líneas de código:

```
barriosGroup.selectAll("path")
.data(this.barrios).enter()
.append("path")
.attr("d", path)
```

5. También se construyen sobre el contenedor `barriosGroup` anterior las etiquetas de los barrios, las mismas son contenedores `g` que contendrán el texto, se agregan mediante el siguiente código:

   ```
   let barrioLabels = barriosGroup.selectAll("g")
   .data(this.barrios).enter()
   .append("g").
   ```

   Este fue el ultimo paso en la construcción del mapa, en los siguientes pasos se explica la lógica para mostrar la información en el mismo.

6. Para agregar el color correspondiente a cada barrio, se calcula la cantidad de viajes que hubo en la hora indicada con destino a cada barrio y se decide el color correspondiente siguiendo la referencias existentes en la Figura 7.3. El color se asigna mediante la asignación de clases *CSS*. Los colores se calculan cuando cambia la hora, y no los minutos.

Las características de *d3.js* utilizadas son: `d3.geo.equirectangular()`, `.scale()`, `.center()`, `d3.geo.path()`, `.projection()`.

7.2.3. Matriz de viajes entre barrios

Esta visualización ilustra la cantidad de viajes en el día entre todo par de barrios.

Consiste de una matriz donde cada celda representa un par de barrios, y el color es el correspondiente a la cantidad de viajes entre ellos, siendo el barrio de la fila el origen, y el barrio de la columna el destino.

A los barrios se les asigna un número para representarlos mas fácilmente. En las filas se muestra el número asignado al barrio junto con su nombre, y en las columnas solo el número.

Al posicionar el ratón sobre una celda se puede ver el detalle de esta información en la barra lateral, como se puede ver en la Figura 7.5.

Una característica de esta visualización es la posibilidad de personalizar la escala de colores utilizada así como la cantidad de viajes correspondiente a cada uno, desde la barra lateral. La escala consiste en tres colores, y tres valores numéricos que corresponden a cada color.

d3.js interpola estos valores numéricos a la escala de colores que van desde el primero hasta el ultimo, pasando por el color del medio. Por defecto estos valores se definen con color verde pastel para la mínima cantidad de viajes que es cero, color amarillo pastel para la cantidad promedio de viajes que es 1.84 y color rojo pastel para la máxima cantidad de viajes que es 176.

También existe la posibilidad de *resetear* los valores mediante el botón de *reset*. Al cambiar estos valores utilizando la entrada numérica correspondiente, se actualiza automáticamente la matriz. De igual manera se actualiza la matriz al cambiar el color correspondiente a un valor numérico haciendo clic en el color.

En la Figura 7.6 se puede ver la matriz de viajes para la escala de colores definida en la Figura 7.5.

Figura 7.5: Barra lateral para la visualización de matriz de viajes entre barrios

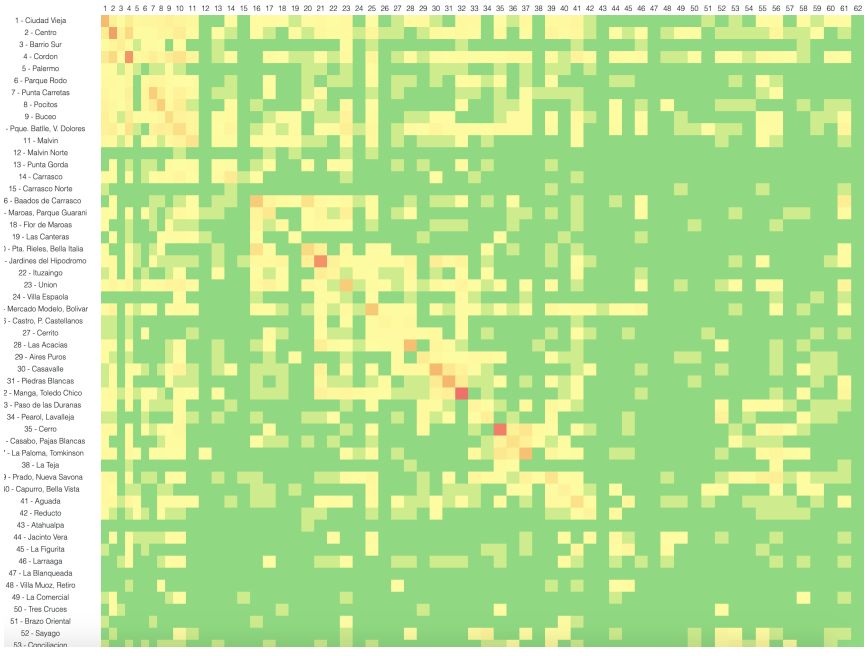

Figura 7.6: Visualización de matriz de viajes entre barrios

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consumen los servicios */barrios* el cual retorna los datos del los barrios y */base/viajes* que retorna los viajes a visualizar, aplicando los filtros correspondientes.

También se consume el servicio */base/diccionarios* que retorna los diccionarios a utilizar por los filtros. Una vez que los servicios retornan esta información, se procede a construir la matriz, calculando para cada barrio la cantidad de viajes hacia todos los demás barrios. Los pasos para dibujar la matriz son los siguientes:

1. Se crea la visualización de la matriz como una tabla, utilizando como datos la propia matriz, y se guarda el resultado en la variable `table`, mediante el siguiente código:

   ```
   let container = d3.select("#matriz-viajes");
   var table = container.append("table")
   ```

2. Se agregan todas las filas necesarias para la tabla:

   ```
   var tr = table.selectAll("tr")
   .data(this.matrix).enter()
   .append("tr");
   ```

3. Para cada fila, se crean todas las celdas con los datos correspondientes:

   ```
   var td = tr.selectAll("td").enter().append("td");
   .data(function(d)  return d; )
   ```

4. Utilizando la tabla creada anteriormente, se procede a colorear cada celda según la cantidad de viajes, mediante las siguientes líneas de código:

   ```
   td.style("background", (d) => { this.color(d) } })
   ```

5. Para agregar el color correspondiente a cada celda, se utilizan los métodos `range()` y `interpolate()` de *d3.js*, que generan dinámicamente una

escala de colores según los parámetros de color y valor utilizados, como muestran las siguientes líneas de código:

```
this.color = d3.scale.linear()
.range([color_inferior, color_medio, color_superior])
.interpolate(d3.interpolateHcl);
```

Las características de *d3.js* utilizadas son: `d3.selectAll().data().enter()` y `d3.range().interpolate()`.

7.2.4. Mapa de líneas de ómnibus

Esta visualización muestra la utilización de las distintas líneas de ómnibus a lo largo del día según la información de la base Viajes. Se puede ver como en el correr del día la representación de las líneas de ómnibus en el mapa aparecen, cambian de color y desaparecen según su utilización. Esto se puede observar en la Figura 7.8.

Las referencias de los colores se pueden ver en la barra lateral, se puede interactuar con la visualización pausando, resumiendo o deteniéndola con los controles. También se puede navegar en el día moviendo el deslizador de hora, según se puede ver en la Figura 7.7.

Además se permite sobresaltar una línea para identificarla del resto mediante los controles interactivos.

En la sección de la barra lateral se puede apreciar también la cantidad de viajes existentes a la hora indicada, así como la cantidad de viajes totales luego de aplicar los filtros seleccionados. Los filtros posibles para esta visualización son: Viajes, Personas, Hogares. Se pueden ver y remover los filtros seleccionados mediante la sección correspondiente en la barra lateral.

Figura 7.7: Barra lateral para la visualización del mapa de utilización de líneas

Figura 7.8: Visualización de mapa de utilización de líneas

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consumen los servicios */lineas* y */base/viajes*, los cuales retornan los datos de las líneas para que *d3.js* los dibuje, y los viajes para visualizar, aplicando los filtros correspondientes.

También se consume el servicio */base/diccionarios* que retorna los diccionarios a utilizar por los filtros. Una vez que los servicios retornan esta información, se procede a dibujar las líneas correspondientes para la hora inicial, utilizando el *GeoJSON* de líneas. Los pasos para dibujar el mapa de líneas son los siguientes:

1. Se crea la proyección del mapa usando pasando las coordenadas de Montevideo como centro y la escala 135000, se guarda el resultado en la variable `projection`, mediante el siguiente código:

   ```
   let projection = d3
   .geo.equirectangular()
   .center([-56.15, -34.8])
   .scale(135000);
   ```

   donde la escala corresponde linealmente a la distancia entre dos puntos proyectados.

2. Utilizando la proyección creada anteriormente se crea el generador geográfico y almacena en la variable `path`, utilizando las siguientes líneas de código:

   ```
   let path = d3.geo.path().projection(projection);
   ```

3. Se procede a crear un contenedor `g` dentro del SVG que contendrá al mapa, agregándole el atributo `id = map` y almacenándolo en la variable `lineasGroup`, mediante la siguiente línea:

   ```
   let lineasGroup = svg.append("g").attr("id", "map");
   ```

4. A continuación se construyen sobre el contenedor `lineasGroup` anterior los dibujos de las líneas utilizando el *GeoJSON* de líneas y el generador geográfico creado en el paso 3) y almacenado en la variable `path`, mediante las siguientes líneas de código:

```
lineasGroup.selectAll("path")
.data(this.lineas).enter()
.append("path")
.attr("d", path)
```

5. Para agregar el color correspondiente a cada línea se utiliza el contador que representa los minutos del día, comenzando 4AM y finalizando 3:59AM. Cada 30 minutos se calculan las cantidades de viajes correspondientes a cada línea y se decide el color correspondiente siguiendo la referencias existentes en la Figura 7.7. El color se asigna mediante la asignación de clases *CSS*.

Las características de *d3.js* utilizadas son: `d3.geo.equirectangular()`, `.scale()`, `.center()`, `d3.geo.path()`, `.projection()`.

7.2.5. Mapa de uso de calles

Esta visualización refleja el impacto de la utilización de las líneas de ómnibus sobre las calles de Montevideo, tomando la información de las líneas de los viajes de la base Viajes de la encuesta.

Se pueden ver todas las calles de Montevideo, y para cada tramo de calle por el cuál pasan líneas de ómnibus, una burbuja con el color correspondiente a la cantidad de viajes acumulados de todas esas líneas a la hora indicada en la barra lateral, ver Figura 7.9.

Se puede interactuar con la visualización pausando, resumiendo o deteniendo la misma con los controles, también se puede navegar en el día moviendo el deslizador de hora, según se puede ver en la Figura 7.3.

En la sección de la barra lateral se puede apreciar también la cantidad de viajes existentes a la hora indicada, así como la cantidad de viajes totales luego de aplicar los filtros seleccionados. Los filtros posibles para esta visualización son: Viajes, Personas, Hogares. Se pueden ver y remover los filtros seleccionados mediante la sección correspondiente en la barra lateral.

Figura 7.9: Barra lateral para la visualización del mapa de calles

Figura 7.10: Visualización de mapa de calles

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consumen los servicios */calles* el cual retorna los datos de las calles para que *d3.js* los dibuje y */base/viajes* que retorna los viajes para visualizar, aplicando los filtros correspondientes.

También se consume el servicio */base/diccionarios* que retorna los diccionarios a utilizar por los filtros. Una vez que los servicios retornan esta información, se procede a dibujar las calles. Para esto se utiliza el *GeoJSON* de calles devuelto por el servicio anterior. Los pasos para dibujar las calles son los siguientes:

1. Se crea la proyección del mapa usando pasando las coordenadas de Montevideo como centro y la escala 150000 para que se vea la ciudad, se guarda el resultado en la variable `projection`, mediante el siguiente código:

   ```
   let projection = d3
   .geo.equirectangular()
   .center([-56.28 , -34.92 ])
   .scale(150000);
   ```

   donde la escala corresponde linealmente a la distancia entre dos puntos proyectados.

2. Utilizando la proyección creada anteriormente se crea el generador geográfico y almacena en la variable `path`, utilizando las siguientes líneas de código:

   ```
   let path = d3.geo.path().projection(projection);
   ```

3. Se procede a crear un contenedor `g` dentro del SVG que contendrá al mapa, agregándole el atributo `id = map` y almacenándolo en la variable `callesGroup`, mediante la siguiente línea:

   ```
   let callesGroup = svg.append("g").attr("id", "map");
   ```

4. A continuación se construyen sobre el contenedor `callesGroup` anterior las calles utilizando el *GeoJSON* de calles y el generador geográfico creado en el paso 3) y almacenado en la variable `path`, mediante las siguientes líneas de código:

```
callesGroup.selectAll("path")
.data(this.calles).enter()
.append("path")
.attr("d", path)
```

5. Luego se construyen sobre el contenedor `callesGroup` anterior, las burbujas correspondientes a cada tramo de calle por el que pasen líneas de ómnibus. Dichas burbujas también son contenedores `g` que contendrán círculos *SVG*, los cuáles se irán coloreando de acuerdo a la cantidad de viajes. Se agregan mediante el siguiente código:

```
callesGroup.append("g")
.selectAll("circle")
.data(this.calles).enter()
.append("circle")
.attr("r", 2.5).
```

6. Para modificar el color de cada burbuja se utiliza el contador que representa los minutos del día, comenzando 4AM y finalizando 3:59AM, el cuál avanza de a 1 minuto. Cada vez que llega a cierta hora en punto, o a la media hora, se calculan los viajes que hubo a la hora correspondiente para cada una de las burbujas, es decir, acumulando los viajes de las líneas para cada tramo de cada calle, y se estas se colorean según ese valor, utilizando rangos de colores, mediante las siguientes líneas de código:

```
var color = d3.scale.linear()
.domain([1, 10, 35])
.range("#FFFF66", "#FF8000", "#990000"])
.interpolate(d3.interpolateHcl)
```

Las características de *d3.js* utilizadas son: `d3.geo.equirectangular()`, `.scale()`, `.center()`, `d3.geo.path()`, `.projection()`, `.range()`, `.interpolate()`.

7.2.6. Características de hogares 1

Esta visualización representa características o indicadores de los hogares encuestados. Cada una está representada mediante un circulo o burbuja flotante con

gravedad hacia el centro de la pantalla, agrupada en su categoría correspondiente.

Las categorías posibles son: “Vivienda”, “Vehículos” y “Servicios”. Estas categorías se representan mediante grupos de círculos. Un grupo es un conjunto de círculos que describen una misma entidad, siendo cada entidad posible un hogar, un vehículo o un servicio.

Los círculos pertenecientes a un mismo grupo tienden a unirse si se les separa arrastrándolos con el ratón. Para ver el detalle de una característica elegida se debe hacer clic en el circulo que la representa, mostrándose así un modal con la información correspondiente.

En la Figura 7.11 se puede observar los círculos agrupados en su grupo o categoría correspondiente, y en la Figura 7.12 se puede ver el modal con la información de la característica “Aire Acondicionado”.

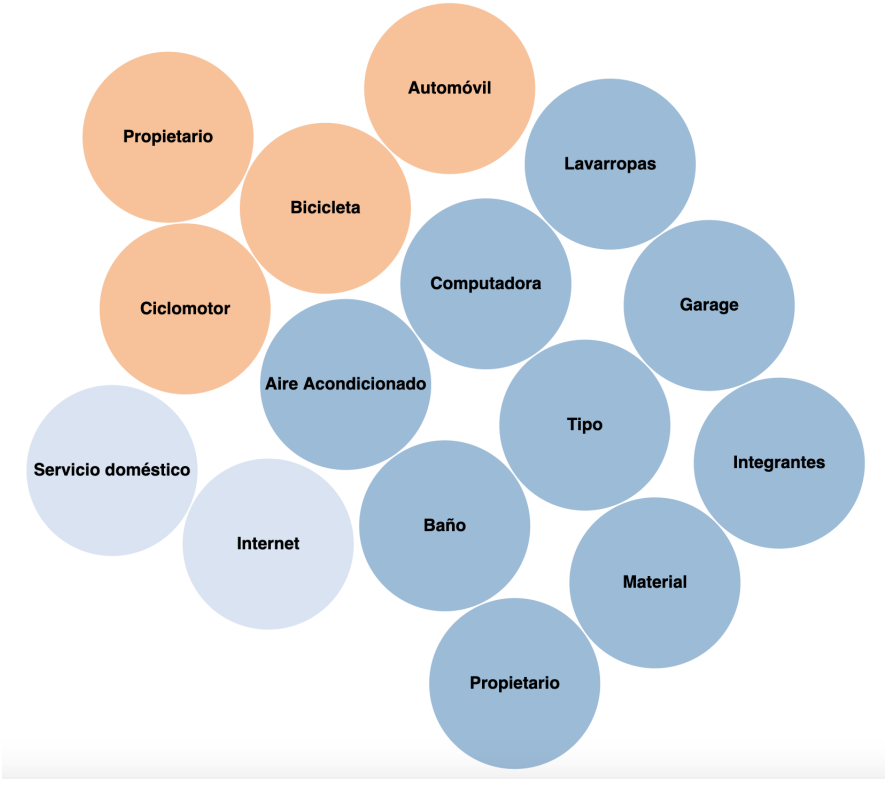

Figura 7.11: Visualización de círculos para hogares

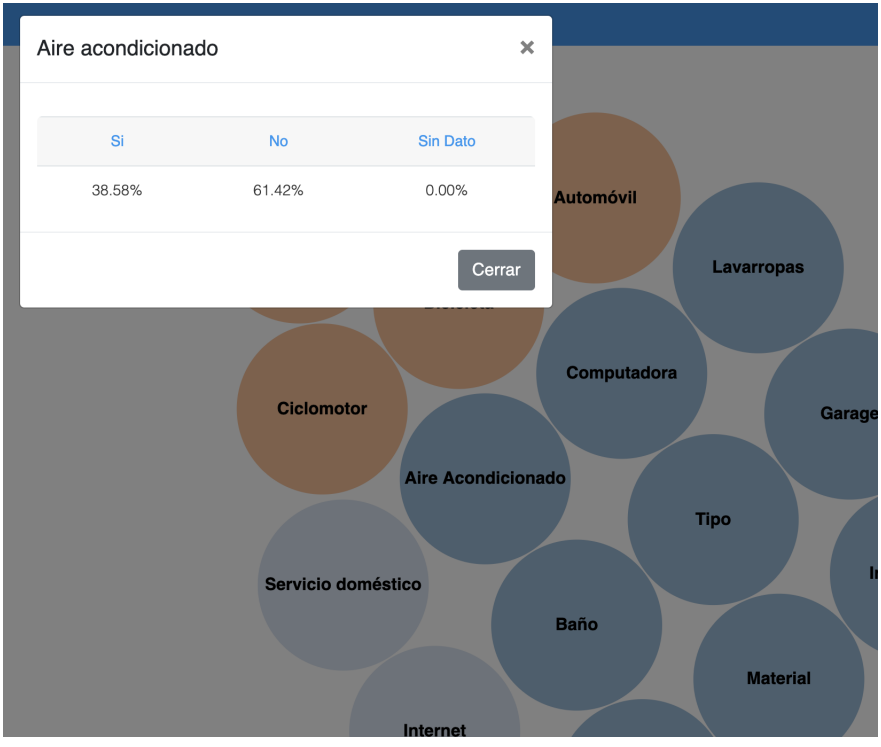

Figura 7.12: Modal de visualización de círculos para hogares

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consume el servicio */base/diccionarios* que retorna los diccionarios, entre ellos el de Hogares que se utiliza para obtener las características de los hogares a mostrar. Una vez obtenida esta información, se consume el servicio */base/hogares*, el cual retorna la encuesta de hogares junto con sus respuestas. A continuación se agrega la categoría a cada característica para agruparlas y darles el color correspondiente.

Una vez hecho esto se procede a dibujar la gráfica mediante los siguientes pasos:

1. El primer paso consiste en crear la variable *nodes*, en la cual se almacenan las características de los hogares a representar, junto con parámetros que luego serán utilizados para construir la visualización, estos parámetros son:

   `cluster`: Refiere a la categoría de la característica.
   `radius`: Es el radio del circulo que va a representar la característica.
   `preview`: Es el texto que va a mostrar el circulo en su centro.
   `x`: Es la posición x del circulo dentro de la visualización.
   `y`: Es la posición y del circulo dentro de la visualización.

   Esto es hecho mediante la función `create_nodes`.

2. Luego se procede a crear un diseño de tipo *force* sobre el cual se va a dibujar la visualización. Un diseño de tipo *force* es un diseño de grafo que crea la librería *d3.js* mediante pocas líneas de código. A este grafo se le pueden personalizar muchas características, como eliminar o editar las aristas, agregar fuerzas entre sus nodos, fuerza gravitatoria, agrupar entre grupo de nodos, etc. En nuestra visualización se crea el grafo mediante las siguientes líneas de código:

   ```
   var force = d3.layout.force()
   .nodes(nodes)
   .size([width, height])
   .gravity(0.2)
   .charge(0)
   ```

```
.on("tick", tick)
.start()
```

De esta forma creamos un diseño de tipo grafo con tamaño *width* y *height*, que son variables que corresponden a las dimensiones de la pantalla, le agregamos los nodos almacenados en la variable *nodes*, variable que fue creada utilizando las características de los hogares a representar.

También se define el valor del parámetro *gravity* del grafo en *0.2*, siendo *gravity* la fuerza que atrae a los nodos hacia el centro de la pantalla. Otro parámetro que se le define al grafo es el *charge*, el cuá l indica la fuerza con la cual se repelen los nodos, que en nuestro caso es cero.

Finalmente se indica la función *tick*, que permite definir que hacer en cada actualización de la visualización, en nuestro caso lo que hacemos es llamar a una función que resuelve las colisiones entre los nodos, esta función no fue desarrollada por nosotros, si no que es una función por defecto de *d3.js*.

3. A continuación se procede a crear un elemento de tipo *g* para cada nodo del grafo que representa una característica del hogar almacenado en la variable `nodes`, esto se hace mediante las siguientes líneas:

```
var node = svg.selectAll("circle")
.data(nodes).enter()
.append("g").call(force.drag)
```

`svg.selectAll("circle")` le dice al navegador que busque el elemento svg y busque dentro de él cualquier elemento con clase `circle`. Si encuentra, los devuelve en una selección que es un arreglo de elementos. Si no encuentra ninguno, devuelve una selección vacía, que es lo que sucederá en este caso ya que aun no se han agregado.

`.data(nodes)` agrega los nodos a la selección resultante, es decir al arreglo vacío.

También se le agrega al nodo la funcionalidad `drag`, que permite el arrastre interactivo del nodo utilizando el ratón y haciendo que interactué con otros nodos mediante las fuerzas fuerzas agregadas cuando se creo el grafo.

4. Luego se le agrega a cada nodo un circulo adentro, que sera el circulo que contiene el texto y el color correspondiente a la categoría de la característica del hogar. Se les agregan las funciones que responden a eventos de clic y de *mouse over*, que quiere decir cuando se posa el ratón sobre el circulo. Esto se hace mediante las siguientes líneas:

```
node.append("circle")
.style("opacity", 0.5)
.style("fill", function (d)  return color(d.cluster) )
.attr("r", function(d)  return d.radius )
.on({
"mouseover": (d) => {
d3.select(this).transition(t).style("opacity", "1");
},
"mouseout": (d) => {
d3.select(this).transition(t).style("opacity", ".5");
},
"click": (d) => {
this.showModal(d);
}})
```

   Aquí podemos ver que para el evento `mouseover` y `mouseout` se modifican estilos para hacer que el circulo cambie tenuemente de color, y se agrega el evento `click`, que abre el `modal` con información sobre la característica seleccionada.

5. Finalmente se agregan los textos que contienen el nombre de la característica a cada nodo, de igual manera que se agregaron los círculos, utilizando las siguientes líneas de código:

```
node.append("text")
.attr("dy", ".3em")
.style("text-anchor", "middle")
.style("font-weight", "bold")
.style("font-size", function(d) { return Math.max(d.radius/5,
8) })
.text(function(d) { return d.preview })
```

   Aquí se define el tamaño de fuente, la posición y del texto dentro del nodo

y el estilo del texto en negrita.

7.2.7. Características de hogares 2

Esta visualización también representa características o indicadores de los hogares encuestados, cada una representada mediante un conjunto de contenedores de líquido, donde cada contenedor representa una opción posible.

Por ejemplo, la característica "Aire Acondicionado" referente a un hogar está representada por tres contenedores de líquido, uno para la opción "Si", otra para la opción "No" y otra para la opción "Sin Dato".

El contenedor se muestra con la cantidad de agua correspondiente al porcentaje de la opción que representa. Por ejemplo, si la opción "No" representa un 61 % de las respuestas, el contenedor asociado a esta opción estará un 61 % lleno de líquido.

Existen cinco tipos diferentes de contenedores, mostrando cada uno en función del orden de la opción, volviendo a repetir si hay mas de cinco opciones.

Aquí también se agrupan las características o indicadores en categorías, siendo éstas las mismas que la visualización anterior: "Vivienda", "Vehículos" y "Servicios".

Para elegir el indicador a visualizar primero se tiene que escoger la categoría, y luego el indicador. Todo esto desde la barra lateral.

En la Figura 7.14 se puede ver la barra lateral donde las categorías están representadas mediante entradas de tipo selección, y en la Figura 7.13 se puede ver un ejemplo de contenedores de líquido para la característica "Aire Acondicionado".

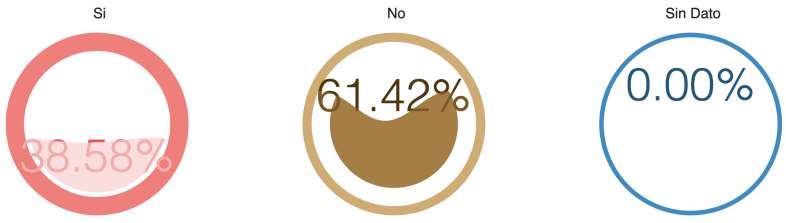

Figura 7.13: Visualización de contenedores de líquido para Hogares

Figura 7.14: Barra lateral de visualización de contenedores de líquido para Hogares

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consumen los servicios */base/diccionarios* y */base/hogares*, que retornan los diccionarios y las características de los hogares respectivamente, al igual que la anterior visualización de Hogares. Una vez obtenidos los datos de los servicios, se procede a construir la visualización a través de los siguientes pasos:

1. En primer lugar, se utiliza *liquidFillGauge.js*[3] para construir los contenedores.
2. A continuación se procede a llenar estos contenedores con los datos de la selección por defecto del panel lateral.
3. Se crean dos funciones, las cuáles actualizan los datos de los contenedores tanto para cambios de categorías como para las características a consultar.

   ```
   categorySelected()
   itemSelected()
   ```

En esta visualización se explora la posibilidad que brinda *d3.js* de utilizar librerías externas construidas con *d3.js* por la comunidad, e incorporarlas al código propio.

7.2.8. Actividades de personas

Esta visualización representa las actividades de personas a lo largo del día. Cada persona es representada mediante un pequeño circulo que se mueve entre diferentes actividades a medida de que avanza la hora del día. Las actividades disponibles son los destinos de los viajes que realiza la persona, que, junto con su hora de inicio, permiten ir construyendo su movimiento diario.

La visualización consiste en un circulo donde todas las actividades están en su borde, y la actividad "Hogar" en el centro al ser esta actividad la que tiene mayor porcentaje a toda hora del día.

Junto al nombre de cada actividad se puede ver el porcentaje de personas que

[3] http://bl.ocks.org/brattonc/5e5ce9beee483220e2f6

están actualmente en ella. En la Figura 7.15 se puede ver un ejemplo de esta visualización.

La hora del día se indica en la barra lateral, al igual que otras visualizaciones, pudiendo poner pausa, reanudarla o detenerla.

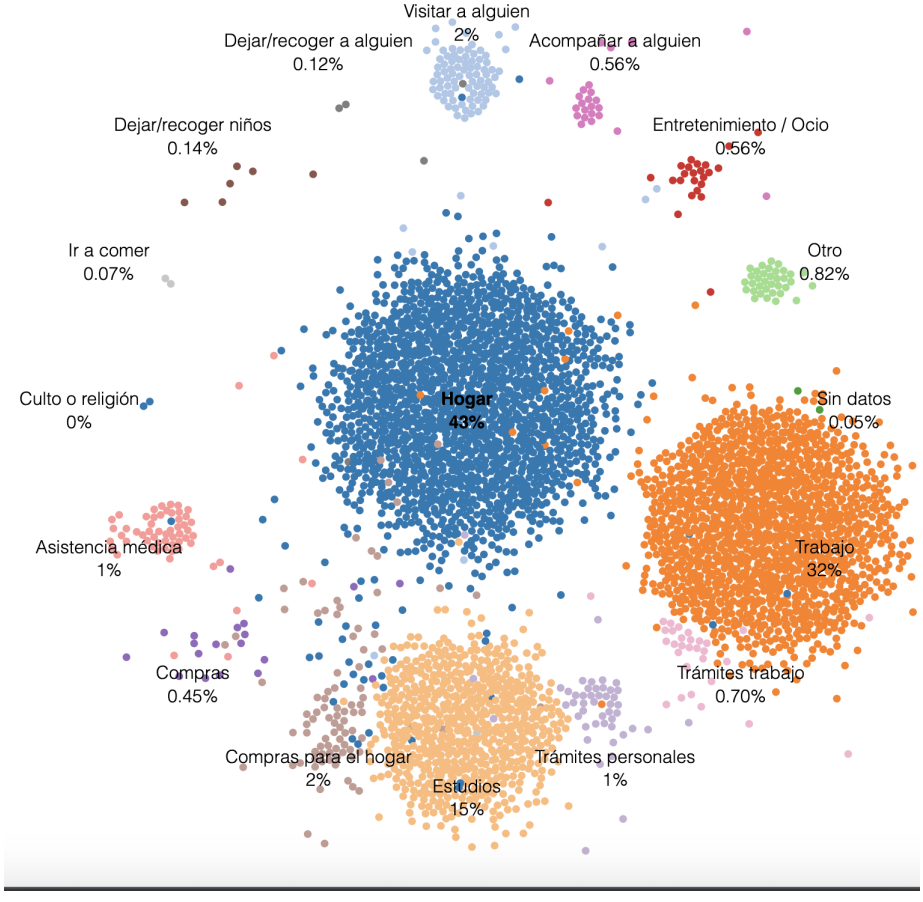

Figura 7.15: Visualización de actividades de personas

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consumen los servicios */base/diccionarios* y */base/viajes/actividades-personas*, que retornan los diccionarios y las actividades de las personas respectivamente. Para construir las actividades de las personas, el servicio */base/viajes/actividades-personas* toma en cuenta para cada persona la actividad en el origen y destino de cada uno de sus viajes, junto con la hora de inicio y su duración. Una vez obtenidos los datos de los servicios, se procede a construir la visualización a través de los siguientes pasos:

1. El primer paso consiste en crear la variable *nodes*, en la cual se almacenan las actividades de todas las personas a representar, junto con parámetros que luego serán utilizados para construir la visualización. Estos parámetros son:

   `act`: Refiere a la actividad de la persona.
   `radius`: Es el radio del circulo que va a representar a la persona.
   `color`: Es el color que va a mostrar el circulo, que se corresponde con actividad .
   `x`: Es la posición x del circulo dentro de la visualización.
   `y`: Es la posición y del circulo dentro de la visualización.
   `next_move_time`: Es el minuto en el que se pasa a la siguiente actividad.
   `sched:` Almacena los movimientos diarios entre actividades de la persona.

2. Luego se procede a crear un diseño de tipo *force* sobre el cual se va a dibujar la visualización, similar al de la visualización "Características de Hogares 1".

   ```
   var force = d3.layout.force()
   .nodes(nodes)
   .size([width, height])
   .gravity(0)
   .charge(0)
   .friction(.9)
   .on("tick", tick)
   .start()
   ```

   De esta forma se crea un diseño de tipo grafo con tamaño *width* y *height*, que

son variables que corresponden a las dimensiones de la pantalla, le agregamos los nodos almacenados en la variable *nodes*, que contiene las actividades de las personas.

Se define el valor del parámetro *gravity* del grafo en *0*, ya que no es necesaria una fuerza que atraiga a los nodos hacia el centro de la pantalla, y se también se define *friction* en *0*, el cual disminuye lentamente la velocidad de los nodos al acercarse a la actividad de destino. Otro parámetro que se le define al grafo es el *charge*, en cero, ya que no deseamos que los nodos se repelan entre si.

Finalmente se indica la función *tick*, similar a la de la visualización anterior.

3. A continuación se procede a crear un elemento de tipo *circle* para cada nodo que representa una persona, esto se hace mediante las siguientes líneas:

```
var circle = svg.selectAll("circle")
.data(nodes).enter()
.append("circle")
.attr("r", function(d) { return d.radius; })
.style("fill", function(d) { return d.color; });
```

4. Luego se crean las etiquetas de actividades, las cuales se almacenan en la variable *label* mediante las siguientes líneas:

```
var label = svg.selectAll("text")
.data(act_codes)
.enter().append("text")
.attr("class", actLabel)
```

5. Las etiquetas se posicionan de manera que la principal actividad inicial (Hogar), quede en el centro, y las restantes equidistantes en un círculo alrededor de la actividad central, definiendo los atributos x e y de la siguiente manera:

```
.attr("x", function(d, i) {
var theta = 2 * Math.PI / (act_codes.length-1);
return ((window.mobilecheck() ? width/4 : width/3)+50) *
Math.cos(i * theta) + width / 2 ;
```

```
}
})
.attr("y", function(d, i) {
var theta = 2 * Math.PI / (act_codes.length-1);
return ((window.mobilecheck() ? width/4 : width/3)+50) *
Math.sin(i * theta) + height / 2;
}
});
```

Aquí se puede ver que utilizamos trigonometría para posicionar las actividades equidistantes en el círculo.

6. Finalmente se crea el *timer* que minuto a minuto actualiza los parámetros de los nodos correspondientes, y la cuenta de la cantidad de personas en cada actividad. Dicho *timer*, para cada persona que cambie de actividad en el minuto actual, realiza ese cambio, actualiza la posición del nodo y los porcentajes correspondientes a las actividades.

Las principales características de *d3.js* utilizadas son: `d3.range()`, `.filter` y `d3.layout.force()`.

7.2.9. Medios de viaje

Esta visualización muestra información sobre los medios de viaje mediante los cuales los usuarios llegaron a su destino a la hora indicada.

La visualización consta de barras horizontales, donde cada una representa un medio de transporte, cuya longitud y porcentaje de uso indicado va variando durante la hora del día. El porcentaje se basa en el total de viajes registrados que tienen como hora de comienzo la hora indicada por el deslizador presente en la barra lateral.

Las barras están ordenadas de mayor a menor, estando siempre en la posición superior la barra de mayor longitud y en la inferior la de menor longitud, a medida que la longitud varía, éstas se reordenan automáticamente con una transición según corresponda

La interacción se realiza desde los controles presentes en la barra lateral, pudiendo avanzar la hora del día mediante el deslizador que indica la misma, permitiendo también resumir/pausar y detener la visualización.

Figura 7.16: Visualización de barras para medios de transporte

En la Figura 7.16 se puede ver la visualización con cada barra horizontal representando un medio de transporte y su porcentaje de uso, a una hora determinada.

Implementación

A continuación se describe la estructura del código para esta visualización.

Al inicializarse el componente, se consume el servicio */base/diccionarios* que retorna los diccionarios, entre ellos el de viajes que se utiliza para obtener los medios de transporte a mostrar y hacer la correspondencia entre los enumerados que representan cada medio y su etiqueta. Por ejemplo el número 13 representa viaje en ómnibus.

Una vez obtenida esta información, se consume el servicio */base/viajes/medios-viajes-personas*, el cual retorna los viajes por persona, ordenados cronológicamente en el día junto con su tiempo de duración en minutos, y la hora del primer viaje. Este servicio es compartido por la visualización "Actividades de personas".

Utilizando estos datos se procede a dibujar las barras mediante los siguientes pasos:

1. Se inicializa la visualización creando un objeto llamado *settings*, que contiene los parámetros necesarios para dibujarla. En este objeto se almacenan las dimensiones de la visualización, y también las escalas utilizando la característica *scales* de *d3.js*. Las escalas son funciones que se asignan desde un dominio de entrada a un rango de salida. Las escalas ordinales tienen un dominio discreto, como puede ser un conjunto de nombres o categorías. También hay escalas cuantitativas, que tienen un dominio continuo, como el conjunto de números reales.

   Las escalas son una característica opcional en *d3.js*, no es necesario usarlos si se prefiere hacer los cálculos uno mismo. Sin embargo, el uso de escalas puede simplificar enormemente el código necesario para asignar una dimensión de datos a una representación visual.

   Para el eje de las ordenadas, se define una escala ordinal, es decir, discreta, con dominio la altura de la pantalla y rango de salida los distintos tipos de medios de transporte, de esta manera se asigna una altura fija a cada barra, que representa un medio de transporte. Para el eje de las abscisas se define una escala cuantitativa no discreta, con dominio entre 0 y el máximo ancho posible de una barra que está dado por el máximo valor para un medio de transporte. Este dominio se aplica al rango del ancho de pantalla.

Las escalas se definen mediante las siguientes líneas de código:

```
var x = d3.scale.linear()
.domain([ 0, defaultBarWidth ])
.range([ 0, width ]);
```

Donde se utiliza el valor por defecto *defaultBarWidth* auxiliar y luego se corrige el dominio cuando se tienen los datos reales usando el máximo valor existente, mediante:

```
var barmax = d3.max(data, function(e)  return e.value );

x.domain([0, barmax]);
```

Luego, para el eje de las ordenadas, el rango de salida se define mediante las siguiente líneas:

```
var y = d3.scale
.ordinal()
.rangeRoundBands([0, height], 0.1, 0);
```

Y el dominio de entrada se define mediante:

```
y.domain(data.sort(function(a,b)
return b.value - a.value )
.map(function(d)  return d.key; ));
```

Esto quiere decir que se divide la altura de la pantalla equitativamente entre los medios de transporte representados por el atributo `key`, que referencia a su nombre, ordenados de mayor a menor mediante la función `sort`.

2. A continuación se procede a crear las barras de la visualización, una por categoría de transporte, mediante las siguientes líneas de código:

```
var chartRow = svg.selectAll("g.chartRow")
.data(datafunction(d)  return d.key )
.attr("fill", (d) => { return this.colors(d.code) })
```

3. Luego se definen las transiciones que se llaman cada vez que hay un cambio en los datos, disparado por la hora del día, el siguiente código actualiza el ancho de la barra en función del nuevo valor de la categoría que representa:

```
chartRow.select("".bar"")
.transition()
.duration(this.transTime)
.attr("width", function(d)  return x(d.value) )
.attr("opacity",.2)
```

   Junto con el ancho, también se actualiza el porcentaje mostrado mediante líneas de código similares.

4. A continuación se reordenan las barras en función de su valor y ancho, utilizando transiciones, mediante el siguiente código:

```
chartRow.transition()
.delay(delay)
.duration(this.transTime*3)
.attr("transform", function(d){return translate(0, y(d.key))})
```

Las características de *d3.js* utilizadas son: `d3.scale.linear()`, `d3.scale.ordinal()` y `transitions`.

Capítulo 8

Conclusiones y trabajo a futuro

8.1. Conclusiones

En esta sección se describen las conclusiones a las cuales se llegaron con respecto al modelado y visualización de datos, así como de la utilización de la librería *d3.js*.

Basándonos en los resultados obtenidos, se considera que se cumplen los objetivos planteados para este proyecto, tanto el objetivo general como los específicos. Se realiza un proceso de selección del caso de estudio que abarca diferentes publicaciones de datos abiertos, y una vez elegida la fuente, se hace un relevamiento del catálogo de datos disponibles para su selección.

Se realiza un modelado de los datos para el caso de estudio elegido, aplicando y documentando las etapas de visualización de datos definidas en introducción teórica, junto con las transformaciones necesarias para poder generar una base de datos sobre la cual se basa la herramienta *web* construida.

Se concluye que es imprescindible la realización de un correcto modelado de los datos del dominio para poder sacarles el máximo provecho. Es importante representarlos de forma clara y atractiva, y permitir una interacción por parte del usuario es fundamental para transmitir la información generada.

Para hacer posible la construcción de la herramienta *web* para la visualización de estos datos, se estudia la librería a utilizar, *d3.js*.
En este camino, se compara *d3.js* con otras herramientas alternativas disponibles.

Con la experiencia adquirida en el transcurso del proyecto, se llega a la conclusión de que es una herramienta muy poderosa para la creación de gráficos web, y creemos que es una de las mas completas de la actualidad.

Aunque su aprendizaje no es fácil, se puede utilizar con conocimientos básicos de desarrollo *web* y del lenguaje *Javascript*, haciéndola una excelente opción a la hora de elegir que herramienta utilizar para generar gráficos medianamente complejos, animados e interactivos. No recomendaríamos su uso para la generación de gráficos simples y estáticos como gráficas de barras, líneas, *pie*, etc, debido a la existencia de herramientas de mas fácil uso para este fin.

La aplicación *web* interactiva construida utiliza las últimas tecnologías, es responsiva, por lo que es posible su utilización tanto es dispositivos móviles como de escritorio, si bien no existen requerimientos establecidos dados por un cliente o contraparte, se cumple con el requerimiento de interacción y adaptabilidad. Se hizo una descripción detallada tanto de su diseño, su implementación y su ejecución, incluyendo una sección de ayuda en ella.

8.2. Trabajo a futuro

A continuación se describen algunos trabajos que se pueden continuar sobre la aplicación *web* construida.

- **Registro, autenticación y roles de usuarios**

  La posibilidad de poder hacer un registro de usuarios es algo que quedó pendiente y que puede desarrollarse de manera tal que el usuario se puede registrar y definir preferencias que quedan asociadas a su sesión, como filtros predefinidos o escalas de colores, que se cargan una vez que vuelve a iniciar sesión. También abre la posibilidad de crear roles de usuarios con distintos permisos o privilegios, permite la existencia de un usuario administrador que edite características de la aplicación.

- **Panel de administración**

  De la mano del punto anterior se da la posibilidad de la existencia de un panel de administración mediante el cual un usuario administrador pueda editar, habilitar, deshabilitar características de la aplicación y de las visualizaciones, globales para todos los usuarios.

- **Modificación de datos de entrada**

  Una de las características descritas en el punto anterior podría ser la modificación del dominio de datos, seleccionando una encuesta de otro año, por ejemplo. Esto es posible debido a los *script*s creados para el modelado de datos, que se puede reutilizar para otro dominio de datos de igual formato. Se intentó sin éxito conseguir datos de encuestas similares anteriores para llevar a cabo este trabajo.

- **Publicar la herramienta abierta al publico junto con un dominio**

  Si bien la herramienta esta actualmente publicada en:

  `http://proygrado18.s3-website-sa-east-1.amazonaws.com`

  y es posible acceder a la misma por cualquier usuario que disponga de un navegador web, este es un dominio generado por la nube de *Amazon* y aún no se dispone de un dominio propio. Se investiga la disponibilidad del dominio [1], estando disponible para su adquisición y utilización como trabajo a futuro.

- **Realizar la comunicación mediante *HTTPS***

  Actualmente la información enviada viaja mediante el protocolo *HTTP*, el hecho de que no se envíe información sensible y que se manejen datos públicos no hace necesario utilizar la versión segura de *HTTPS*, quedando esto como trabajo a futuro una vez se comience a manejar información sensible

[1] http://www.encuestaorigendestino.uy

de la mano de la autenticación.

- **Utilización de herramientas de integración y despligue continuos**

  Una buena práctica que acelera el desarrollo y la calidad del producto es la utilización de herramientas de integración y despliegue continuos. Consisten en motores en la nube o servidores que ejecutan tareas cada vez que se modifica el código fuente de la aplicación. Estas tareas pueden ser correr tests, chequear que el código cumpla con un estándar definido, hacer un despliegue del código automático, etc.

- **Agregado de índices a las tablas de la base de datos**

  Agregando índices para ciertos atributos en las distintas tablas permitiría mejorar la velocidad de búsqueda de datos.

- **Test unitarios y de integración**

  Si bien la herramienta *angular-cli* trae incorporado código *boilerplate* para la creación de *tests* unitarios, estos no se incluyeron en el código por razones de tiempo, enfocándose el esfuerzo en otras tareas de mayor relevancia, se reconoce que los *tests* unitarios son muy importantes a la hora de generar código libre de *bug*s, quedando estos, junto los tests de integración con la *API REST* como trabajo a futuro.

- **Ejecución del servidor de base de datos y servidor de servicios *REST* en distintos nodos, incluir un balanceo de carga y herramienta de analítica web**

  Estas características se agregan como trabajo a futuro ya que son características muy útiles a la hora de escalar un producto. Separar el servidor de base de datos y el servidor *REST* permite alivianar la carga de procesamiento en las máquinas virtuales que los ejecutan.

  A su vez, agregar un balanceador de carga, a medida que el tráfico del sitio es mayor, permite una mejor usabilidad del sistema, sin sobrecargar las maquinas virtuales que lo conforman.

Un balanceador de carga es un componente de la arquitectura que permite tener múltiples servidores *REST*, siendo el balanceador de carga el árbitro encargado de decidir a que servidor *REST* enviar la consulta, dependiendo de su porcentaje de uso actual, y así dividir eficientemente la carga de los servidores involucrados.

Finalmente, utilizar una herramienta de analítica web, como *Google Analytics* permite obtener métricas en tiempo real del uso de la aplicación, y así poder tomar decisiones como las anteriormente mencionadas: utilización de balanceo de carga, separación de nodos, aumento de recurso de servidores, etc.

Anexos

Anexo A

Relevamiento de datos del catálogo de datos abiertos

A continuación se hace un relevamiento de los datos del catálogo de datos abiertos del Uruguay disponibles en [1] y [2] al momento de hacer el estudio de los conjuntos de datos disponibles, Agosto de 2018.

Los conjuntos de datos disponibles se dividen en 15 categorías diferentes, en la tabla A.1 se pueden ver las categorías junto con la cantidad de conjuntos de datos de cada una.

[1] https://catalogodatos.gub.uy
[2] http://www.ine.gub.uy

Categoría	Cantidad de conjuntos de datos
Infraestructura	22
Transparencia	15
Economía	12
Transporte	12
Desarrollo Social	6
Cultura	3
Geográficos	3
Turismo	3
Medio Ambiente	2
Salud	2
Trabajo	2
Datos geográficos	1
Educación	1
Industria y energía	1
Vivienda	1

Tabla A.1: Categorías de datos del catálogo de datos abiertos

A continuación se detallan los conjuntos de datos disponibles por categoría:

Infraestructura:

- Vías de Montevideo con cabezales de numeración, significado, tipo y título. Volumen de datos: No aplica.
- Listado de calles de Montevideo con información de numeración, significado, tipo y título. Volumen de datos: No aplica.
- Parcelas catastrales: *shapefile* de polígonos con parcelas catastrales (único dato: número de padrón). Volumen de datos: No aplica.
- Límites de Barrios: *shapefile* de polígonos. Barrios de Montevideo según la división del Instituto Nacional de Estadística (INE). Volumen de datos: No aplica.
- Vías de tránsito: Shapefile de líneas con las vías de tránsito de Montevideo. Volumen de datos: No aplica.
- Curvas de Nivel cada 2 metros - Montevideo: *shapefile* de líneas con las curvas de nivel cada 2 metros, según relevamiento realizado en 1975. Volumen de datos: No aplica.
- Comisiones Especiales Permanentes POT: Zonas de funcionamiento de las Comisiones Especiales Permanentes. Volumen de datos: No aplica.
- Espacios libres de Montevideo: *shapefile* de polígonos de los espacios públicos (plazas, parques, canteros, separadores viales, etc). Volumen de datos: No aplica.
- Direcciones oficiales de Montevideo: *shapefile* de puntos con las direcciones oficiales de Montevideo (calle, nro. de puerta y padrón). Volumen de datos: No aplica.
- Inversión del presupuesto participativo: *shapefile* de polígonos que representan los 18 CCZ con información de inversión realizada según presupuesto participativo del año 2006, clasificada por rubro. Volumen de datos: No aplica.
- Manzanas: *shapefile* de polígonos con las manzanas de Montevideo y número de carpeta catastral. Volumen de datos: No aplica.
- Nomenclátor de calles de Montevideo y Área Metropolitana: Archivo de texto con los campos: código de calle, nombre, código de departamento: 1=Montevideo, 3=Canelones, 16=San José Código de localidad según INE. Volumen de datos: No aplica.
- Límites de Centros Comunales Zonales: *shapefile* de polígonos de los 18 centros comunales zonales. Volumen de datos: No aplica.
- Marco de Referencia Geodésico: shapefile de puntos correspondientes al marco de referencia geodésico de Montevideo (MRGMVD). Volumen de datos: No aplica.
- Oficinas de Centros Comunales Zonales: shapefile de puntos. Ubicación de las

oficinas de CCZ (dirección, teléfono). Volumen de datos: No aplica.
- Oficinas de Municipios: shapefile de puntos. Ubicaciones de oficinas de Municipios (direcciones). Volumen de datos: No aplica.
- Parcelas catastrales con datos del Plan de Ordenamiento Territorial: shapefile de polígonos con parcelas catastrales con datos del POT (retiros y alturas de edificación, ocupación del suelo, planes especiales, etc). Volumen de datos: No aplica.
- Población por zona censal en Montevideo: Para el recurso de 2004: shapefile de polígonos con información de población por zonas censales, relevada en el año 2004 Información se obtenida del INE. Volumen de datos: No aplica.
- Jerarquización vial del Plan de Ordenamiento Territorial: shapefile de líneas con la estructura vial jerarquizada definida en el Plan de Ordenamiento Territorial. Volumen de datos: No aplica.
- Cruces de calle de Montevideo: Cruces de calles correspondientes al departamento de Montevideo. Volumen de datos: No aplica.
- Obras del Presupuesto Participativo: shapefile de puntos con las ubicaciones de obras del Presupuesto Participativo realizadas entre 2007 y 2009. Volumen de datos: No aplica.
- Código Postal: El código postal es un identificador numérico o alfanumérico compuesto de una serie de letras y/o cifras e identifica un punto de entrega o un conjunto de puntos de entrega. Volumen de datos: No aplica.
- Padrones urbanos y rurales de Durazno: Lista el conjunto de padrones urbanos de Durazno con sus correspondientes manzana donde está ubicado, superficies totales y edificadas. Volumen de datos: No aplica.

Economía:

- Creación de Empresas a través de Empresa en el día: Es la cantidad de empresas creadas en el mes de referencia a través del sistema de empresa en el día, discriminado por departamento (Salto, Montevideo) donde se cuenta con oficina. Volumen de datos: Enero/2017 hasta Marzo 2018, 60 por mes.
- Padrones urbanos y rurales: Datos alfanuméricos relevantes de padrones urbanos y rurales de todo el país. Volumen de datos: Septiembre/2016 hasta Actualidad, $>$ 1.000.000 por mes.
- Crédito presupuestal asignado y ejecutado: Montos de ejecución y asignación del crédito presupuestal desde el año 2011 en adelante, por programa presupuestal, unidad ejecutora y tipo de gasto. Volumen de datos: 2013 hasta Actualidad, 9000.
- Histórico de crédito presupuestal ejecutado de 1961 a 2010: Montos de ejecución del Presupuesto por Incisos de los años 1961 a 2010, con apertura por tipo de

gasto (funcionamiento e inversiones) a partir de 1975. Volumen de datos: 2013 hasta Actualidad, 9000.
- Indicadores de Área Programática y Programa: se presentan los indicadores de contexto relacionados con cada área programática e indicadores para medir el cumplimiento del objetivo de cada programa.
- Presupuesto Nacional 2015 - 2019: Montos aprobados en la Ley de Presupuesto Nacional para el período 2015 - 2019.
Histórico de crédito y ejecución presupuestal por UE de 1999 a 2010: Montos de asignación y ejecución de crédito presupuestal por UE de los años 1999 a 2010, con apertura por tipo de gasto (funcionamiento e inversiones).
- Monto CIF exonerado para bienes de capital por CIIU: En el marco de la ley 16.906 referida al régimen para la promoción y protección de las inversiones realizadas en territorio nacional. - Balance de ejecución presupuestal: Se presenta el Resultado de Ejecución Presupuestal de Gastos de Funcionamiento, Inversiones, Retribuciones Personales e Ingresos.
- Declaraciones del Sistema de Precios al Consumidor: Histórico de las declaraciones del sistema de Precios al consumidor.
- Histórico Rendición de Cuentas 2012-2015: Rendición de Cuentas de la Intendencia de Durazno - Monto ejecutado en proyectos de inversión: Información sobre monto ejecutado en proyectos de inversión recopilada en el Sistema de Información y Seguimiento de Inversiones (SISI) en ocasión de cada Rendición de Cuentas.

Transporte:

- Vías de Montevideo con cabezales de numeración, significado, tipo y título: Listado de calles de Montevideo con información de numeración, significado, tipo y título. Volumen de datos: No aplica.
- Horarios de ómnibus urbanos (por parada) - Sistema de Transporte Metropolitano: Este juego de datos contiene los horarios de ómnibus del transporte colectivo urbano de Montevideo para cada una de las paradas. Volumen de datos: No aplica.
- Accidentes de Tránsito Montevideo: shapefile de puntos con los accidentes de tránsito con lesionados ocurridos en el período 2006-2010. Volumen de datos: 41000.
- Líneas de ómnibus (variantes no maximales): shapefile de líneas con el recorrido de los ómnibus (variantes no maximales). - Régimen de circulación de vehículos de carga: shapefile de polígonos con las zonas diferenciadas según régimen de circulación de cargas. - Líneas de ómnibus, origen y destino: Los datos descargados

entre otros, contienen la descripción de las líneas, su origen y destino. - Transporte colectivo: paradas y puntos de control: shapefile de puntos con las ubicaciones de paradas y de puntos de control de horario de los recorridos de los ómnibus. - Multas de tránsito: Se presenta la información de multas de tránsito impuestas por parte de la Intendencia de Montevideo, en el período 2006 - 2014.
- Horarios de ómnibus urbanos - Sistema de Transporte Metropolitano: este juego de datos contiene los horarios de ómnibus del transporte colectivo urbano de Montevideo.
- Personas fallecidas en siniestros de tránsito en el primer semestre del 2013: serie de personas fallecidas en Siniestros de Tránsito en el primer semestre de año 2013 en Uruguay.
- Siniestros de tránsito en Uruguay en el año 2011: cantidad de fallecidos en siniestros de tránsito: por Departamento y por mes, por Departamento y por sexo, por edad, por Jurisdicción.
- Serie de personas fallecidas en Siniestros de Tránsito en el año 2012 en Uruguay.

Desarrollo Social:

- Guía de trámites: información de los trámites y servicios que se encuentran publicados en [3].
- Ámbitos de participación ciudadana: relevamiento de ámbitos de participación ciudadana institucionalizados, creados por los organismos del Estado Uruguayo en los tres niveles de gobierno.
- Encuesta panel a participantes de Uruguay Trabaja (2009): en este espacio se comparten micro datos y *metadata* de la encuesta panel de Üruguay Trabaja". - Informes de autoevaluación del segundo plan nacional de Gobierno Abierto: en el Plan de Acción 2014-2016, Uruguay establece 40 compromisos con 91 metas que se encuentran agrupados en ocho ejes temáticos. - Asentamientos Irregulares: este mapa de Asentamientos Irregulares de Montevideo se relevó entre Noviembre de 2006 y Marzo de 2007. - Encuesta Continua de Hogares 2008 - Módulo Políticas Sociales: módulo anexo a la Encuesta Continua de Hogares que se llevó a cabo durante los meses de Agosto a Diciembre de 2008.

Cultura:

- Obras del acervo: lista de obras de arte del acervo del MNAV.
- Monumentos históricos y bienes de interés municipal: shapefile de polígonos que

[3] http://tramites.gub.uy

determinan zonas declaradas de interés municipal y ubicaciones de monumentos históricos localización de los inmuebles.
- De interés para el turista: información de interés para el turista, como eventos, como llegar a Durazno, lugares para comer, diversión y para visitar, cajeros, gomerías, redes de cobranzas.

Geográficos:

- Relevamiento LIDAR de la ciudad de Vergara - Treinta y Tres: conjunto de datos LIDAR de la ciudad de Vergara y alrededores en el departamento de Treinta y Tres. La densidad de puntos es de 1 punto por metro cuadrado.
- Modelo Digital de Superficie de Piriápolis.
- Modelo digital de superficie de Piriápolis y alrededores. El Modelo Digital de Superficie fue capturado en estaciones fotogramétricas 3D en base al vuelo fotogramétrico.
- Modelo Digital de Terreno de Piriápolis: Modelo Digital de Terreno de la ciudad de Piriápolis y alrededores.

Turismo:

- Operadores Turísticos Habilitados: el conjunto de datos incluye operadores inmobiliarios, alojamientos, establecimientos rurales, guías turísticos, etc. Estos datos se actualizan en tiempo real.
- Encuesta de Turismo Receptivo: datos estadísticos trimestrales del Ministerio de Turismo.
- De interés para el turista: información de interés para el turista, como eventos, como llegar a Durazno, lugares para comer, diversión y para visitar, cajeros, gomerías, redes de cobranzas.

Medio Ambiente:

- Gestión del agua: los datos y herramientas que ponemos a disposición se generaron en el marco del proyecto RÍO ABIERTO.
- Contenedores de residuos reciclables (plásticos, latas, vidrios y pilas): shapefile de puntos con la ubicación de contenedores habilitados para el depósito de residuos especiales: pilas, latas, plásticos y vidrio.

Salud:

- Movilidad regulada 2016: información sobre los Prestadores de Salud del Uruguay.
- Tasa de enfermedades y eventos de notificación obligatoria: se pone a disposición las series anuales correspondientes a las tasas de enfermedades transmisibles y eventos de notificación obligatoria.

Trabajo:

- Creación de Empresas a través de Empresa en el Día: es la cantidad de empresas creadas en el mes de referencia a través del sistema de empresa en el día, discriminado por departamento (Salto, Montevideo) donde se cuenta con oficina.
- Vínculos con el Estado: vínculos laborales con el Estado.

Datos geográficos:

- Rutas y puntos de calibración para segmentación dinámica de rutas nacionales: el conjunto de datos contiene las rutas nacionales y los puntos de calibración para trabajar con herramientas de segmentación dinámica y de referencia.

Educación:

- Bibliotecas Municipales de Montevideo: shapefile de puntos con las Bibliotecas Municipales Esta información es proporcionada por la División Promoción Cultural, Servicio de Bibliotecas de la Intendencia de Montevideo.

Industria y energía:

- Encuesta de Usos de las Tecnologías de la Información y Comunicación: primera encuesta a nivel nacional sobre el acceso y uso de las Tecnologías de la Información y la Comunicación (TIC) denominada EUTIC.

A continuación se describe el relevamiento de los datos de Demografía y Estadísticas Sociales de [4].

Los conjuntos de datos disponibles se dividen en 15 categorías diferentes, en la tabla A.2 se pueden ver las categorías junto con la cantidad de conjuntos de datos

[4] http://www.ine.gub.uy

Categoría	Cantidad de conjuntos de datos
Población	3
Trabajo	2
Condiciones de vida	3
Otras Estadísticas Sociales	12

Tabla A.2: Categorías de datos de demografía y estadísticas sociales

de cada una.

Población:

- Censos: series de cantidad de población, viviendas y hogares que surgen de los censos de 1852 al 2011.
- Datos desagregados por departamentos, localidades y secciones censales, dando cuenta de los diversos procesos de evolución de los tamaños poblacionales, en el período de 1963 a 1996.
- Censo 1996: El operativo Censos 1996 incluyó el VII Censo General de Población, III de Hogares y V de viviendas. El día de realización fue el 22 de mayo de 1996 en todo el territorio nacional.
- Censo 2004: Los principales objetivos perseguidos fueron la actualización de la cartografía con fines estadísticos y censales y la toponimia referencial de todo el país. La actualización del marco de unidades estadísticas: viviendas, hogares, población y locales no destinados a vivienda, la obtención de información útil para las estimaciones de los saldos migratorios y la revisión de las proyecciones demográficas. Finalmente contar con información básica para renovar el diseño de las encuestas por muestreo, especialmente de la Encuesta Continua de Hogares y de la Encuesta de Gastos e - Ingresos de los Hogares. - Censo 2011: El operativo Censos 2011 incluyó el relevamiento en campo de los censos de Domicilios, Entorno Urbanístico, Locales, Viviendas, Hogares y Población, y se ejecutó durante el período comprendido entre el de setiembre y el 30 de diciembre de 2011.
- Estadísticas Vitales: series sobre Defunciones, Matrimonios, Divorcios y Nacimientos a nivel nacional.
- Indicadores Demográficos: información sobre la evolución de los fenómenos demográficos (natalidad, fecundidad, mortalidad, nupcialidad, migración) y del crecimiento y estructura de la población residente en el país. Se obtiene a través de Censos, Encuestas o Registros.

Trabajo:

- Actividad, Empleo y Desempleo: tasa de actividad, empleo y desempleo, separados por Interior del País, Montevideo y el Total Nacional, conteniendo además tasas específicas pudiendo diferenciar por tramos de edad, sexo, horas trabajadas y categorías de la ocupación. - Seguridad Social:
AFAP: afiliados y montos anuales transferidos.
BPS: afiliados y pensionistas por año y departamento.
CNJP: afiliados.
CJPB: afiliados.

CJPPU: afiliados.
- Ingresos y egresos del sistema de seguridad social.

Condiciones de Vida:

- Ingresos de las Personas y los Hogares: información referente a ingresos y gastos de las personas y hogares así como Necesidades Básicas Insatisfechas y Línea de Pobreza.
- Pobreza: estudios de caracterización de la indigencia, la pobreza e indicadores de desigualdad y distribución del ingreso a partir de la Encuesta Continua de Hogares (ECH). Los análisis se realizan a partir de distintas variables de clasificación, como área geográfica, sexo del jefe del hogar, grupos de edad, ascendencia racial, entre otras.
- Necesidades Básicas Insatisfechas: información de la población con necesidades básicas insatisfechas (falta de acceso a determinados bienes y servicios que se consideran críticos para el desarrollo humano).

Otras Estadísticas Sociales:

- Salud: información sobre estadísticas de natalidad, mortalidad e indicadores básicos de salud. Los datos provienen en su mayoría del Ministerio de Salud Pública.
- Educación: información de alumnos matriculados en el Sistema Nacional de Educación del Uruguay. Población de 25 y más, según tramos de años de estudio completados.

Glosario

HTML Hypertext Markup Language: es el estándar para la elaboración de páginas web. 21, 22, 67

angular-cli Angular CLI es herramienta para desarrollar aplicaciones web o móvil con Angular, es una herramienta de linea de comandos que facilita la creación, generación, ejecución, testing, deploy.. 45, 61, 108

apache El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix, Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual según la normativa RFC 2616.. 61

API Una API es una interfaz de programación de aplicaciones (del inglés API: Application Programming Interface). Es un conjunto de rutinas que provee acceso a funciones de un determinado software.. 45, 56, 60, 62, 108

applet Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo, en un navegador web.. 21

AWS Amazon Web Services es una colección de servicios de computación en la nube pública que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com.. 59, 60

boilerplate Porción de de código auto generado que es utilizado en varios lugares con pocos cambios.. 44, 45, 108

Bootstrap Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web.. 45, 46, 64

bug Un error de software, error o simplemente fallo (también conocido por el inglés bug) es un problema en un programa de computador o sistema de software que desencadena un resultado indeseado.. 108

bundle Un bundle sirve para agrupar todos los ficheros de JavaScript en uno solo. Así el navegador no necesita hacer varias peticiones HTTP.. 60, 61

CAF Banco de desarrollo de América Latina.. 27

carrusel El Slider o Carrusel Web es un elemento de una página web que muestra múltiples imágenes y texto que se alternan entre ellas.. 64, 67

CDN Una red de distribución de contenidos es una red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red.. 61

checkbox El elemento HTML <input type=checkbox> es un elemento de entrada que te permite indicar que el elemento está o no seleccionado.. 53

cloudfront Amazon CloudFront es un servicio rápido de red de entrega de contenido (CDN) que distribuye a clientes globalmente datos, vídeos, aplicaciones y API de forma segura, con baja latencia, altas velocidades de transferencia y dentro de un entorno fácil para desarrolladores.. 61, 62

CSS Cascading Style Sheet: es el lenguaje utilizado para establecer el diseño visual de los documentos web.. 73, 82

CSS3 Es la tercera versión de CSS.. 13, 19

CSV Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.. 6, 29–32

DD Los grados decimales (DD) expresan las coordenadas geográficas de latitud y longitud como fracciones decimales y se utilizan en muchos sistemas de información geográfica (GIS), aplicaciones de mapas web como OpenStreetMap y dispositivos GPS. Los grados decimales son una alternativa al uso de grados, minutos y segundos (DMS).. 6, 33

DELETE Método de una API REST que permite borrar datos del servidor o base de datos.. 58

DOM *Document Object Model* es una convención para representar e interactuar con los objetos de los documentos *HTML*, basándose en la estructura jerárquica de *HTML*. El *DOM* permite que la estructura, estilo y contenido de un documento sean modificados dinámicamente.. 13, 19, 20, 44

EC2 T2 Micro Las instancias T2 son un tipo de instancia de uso general y bajo costo que proporcionan un nivel base de rendimiento de la CPU con la capacidad para ampliarse por encima del punto de referencia cuando resulta necesario. Con precios de instancias bajo demanda a partir de 0,0058 USD por hora, las instancias T2 son una de las opciones de instancias de Amazon EC2 más económicas y son ideales para una variedad de aplicaciones de uso general.. 61

ECMAScript ECMAScript es una especificación de lenguaje de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO 16262.. 21

Express Express.js, o simplemente Express, es un marco de aplicación web para Node.js, lanzado como software gratuito y de código abierto bajo la Licencia MIT. Está diseñado para construir aplicaciones web y APIs. Se le ha llamado el marco de servidor estándar por defecto para Node.js.. 56

footer El footer es la parte inferior de una estructura web.. 45, 48

framework Un framework, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.. 44, 45, 56

frontend Frontend es la parte de un sitio web que interactúa con los usuarios, por eso decimos que está del lado del cliente.. 60, 61

GeoJSON GeoJSON Es un formato estándar abierto diseñado para representar elementos geográficos sencillos, junto con sus atributos no espaciales, basado en JavaScript Object Notation.. 32, 33, 72, 81, 85

GET Método de una API REST que permite solicitar datos del servidor o base de datos.. 58

HTML5 es la quinta revisión de HTML, que añade nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. 13, 19, 21

HTTP El Protocolo de transferencia de hipertexto es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. 58, 60, 107

HTTPS El Protocolo seguro de transferencia de hipertexto, es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP.. 58, 60, 61, 107

IDE Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.. 43

IP La dirección IP es un número que identifica, de manera lógica y jerárquica, a una Interfaz en red de un dispositivo que utilice el protocolo IP o, que corresponde al nivel de red del modelo TCP/IP.. 60

javascript JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.. 18–20, 22, 56, 58, 59, 61, 106

JOIN La sentencia JOIN de SQL permite combinar registros de una o más tablas en una base de datos relacional. En el Lenguaje de Consultas Estructurado hay tres tipos de JOIN: interno, externo y cruzado.. 59

JQuery jQuery es una biblioteca multiplataforma de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.. 44

JSON JSON es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.. 6, 31, 32, 34, 40, 59

loading spinner Un loading spinner es una animación que es mostrada mientras se ejecutan diversos procesos en segundo plano.. 45

log En informática, se usa el término log, historial de log o registro a la grabación secuencial en un archivo o en una base de datos de todos los acontecimientos que afectan a un proceso particular. De esta forma constituye una evidencia del comportamiento del sistema.. 58

modal En el diseño de la interfaz de usuario para aplicaciones informáticas, una ventana modal es un elemento de control gráfico subordinado a la ventana principal de una aplicación. Crea un modo que deshabilita la ventana principal, pero la mantiene visible, con la ventana modal como una ventana secundaria delante de ella.. 53

nginx DescripciónNginx es un servidor web/proxy inverso ligero de alto rendimiento y un proxy para protocolos de correo electrónico. Es software libre y de código abierto, licenciado bajo la Licencia BSD simplificada.. 61

NodeJS Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.. 56, 58, 59, 61

PATCH Método de una API REST que permite actualizar parcialmente datos en el servidor o base de datos.. 58

POST Método de una API REST que permite crear datos en el servidor o base de datos.. 58

PostgreSQL PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL, similar a la BSD o la MIT. 58

PUT Método de una API REST que permite actualizar datos en el servidor o base de datos.. 58

Python Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.. 31, 34, 56

queryString Query string o, en español, cadena de consulta es un término informático que se utiliza para hacer referencia a una interacción con una base de datos. Es la parte de una URL que contiene los datos que deben pasar a aplicaciones web.. 53, 56–58

REST La transferencia de estado representacional (en inglés representational state transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.. 45, 56, 60, 62, 108, 109

S3 Amazon S3 o Amazon Simple Storage Service es un servicio de almacenamiento simple ofrecido por Amazon Web Services que proporciona almacenamiento de objetos a través de una interfaz de servicio web.. 60

script Es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.. 31, 42, 59, 61, 107

SEO El posicionamiento en buscadores, optimización en motores de búsqueda o SEO (del inglés search engine optimization), es un conjunto de acciones orientadas a mejorar el posicionamiento de un sitio web en la lista de resultados de Google, Bing, u otros buscadores de internet.. 22

shapefile El formato ESRI Shapefile es un formato de archivo informático propietario de datos espaciales desarrollado por la compañía ESRI, quien crea y comercializa software para Sistemas de Información Geográfica como Arc/Info o ArcGIS.. 32

SQL SQL (por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje específico del dominio utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.. 58, 59

ssh SSH es el nombre de un protocolo y del programa que lo implementa cuya principal función es el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada.. 60

SVG *Scalable Vector Graphics* es una familia de especificaciones para crear gráficos vectoriales en dos dimensiones.. 13, 19, 20, 86

toolkit GWT o Google Web Toolkit es un framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología AJAX. Es compatible con varios navegadores, lo cual es notorio ya que cada navegador suele necesitar código específico para lograr un front-end correcto en una aplicación web.. 19

ts Extension de archivos escritos utilizando el lenguaje TypeStript.. 45

TXT La extensión .txt es un documento sin formato, es simplemente texto. En general, cualquier editor de texto permite editar un archivo .txt.. 6, 29, 32, 40

typescript TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases. 44

URL Un localizador de recursos uniforme (más conocido por las siglas URL, del inglés Uniform Resource Locator) es un identificador de recursos uniforme (Uniform Resource Identifier, URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo. Están formados por una secuencia de caracteres de acuerdo a un formato modélico y estándar que designa recursos en una red como, por ejemplo, Internet.. 38, 45, 58, 64

UTM El sistema de coordenadas universal transversal de Mercator (en inglés Universal Transverse Mercator, UTM) es un sistema de coordenadas basado en la proyección cartográfica transversa de Mercator, que se construye como la proyección de Mercator normal, pero en vez de hacerla tangente al Ecuador, se la hace secante a un meridiano.. 6, 33

Bibliografía

[1] Ignasi Alcalde. *Visualización de la información: de los datos al conocimiento.* Editorial UOC, 2015.

[2] https://datos.gob.es. Visualización de datos: definición, tecnología y herramientas. `https://datos.gob.es/es/documentacion/visualizacion-de-datos-definicion-tecnologia-y-herramientas` - Último acceso 21 de Mayo de 2019.

[3] Ignasi Alcalde. Visualización de datos: el diseño de la comprensión. `https://ignasialcalde.es/visualizacion-de-datos-el-diseno-de-la-comprension` - Último acceso 21 de Mayo de 2019.

[4] Albert Obiols. ¿Qué es la Visualización de datos - DataViz? `https://inlab.fib.upc.edu/es/blog/que-es-la-visualizacion-de-datos-dataviz` - Último acceso 21 de Mayo de 2019.

[5] Ben Fry. *Visualizing Data.* O'Reilly, 2008.

[6] Open Source Initiative. The 2-Clause BSD License. `https://opensource.org/licenses/bsd-license.php` - Último acceso 21 de Mayo de 2019.

[7] Loup Verlet. *Computer Experiments on Classical Fluids.* 1967.

[8] Dmitry Baranovskiy. Raphaël—JavaScript Library. `http://dmitrybaranovskiy.github.io/raphael/` - Último acceso 21 de Mayo de 2019.

[9] Ben Fry and Casey Reas. Processing.js. http://processingjs.org - Último acceso 21 de Mayo de 2019.

[10] Stuart K. Card y James a. Landay Jeffrey Heer. Prefuse: a toolkit for interactive information visualization. http://bid.berkeley.edu/files/papers/2005-prefuse-CHI.pdf - Último acceso 21 de Mayo de 2019.

[11] Adobe. Action Script. https://www.adobe.com/devnet/actionscript.html - Último acceso 21 de Mayo de 2019.

[12] Adobe. Adobe Flash Player. https://www.adobe.com/es/products/flashplayer.html - Último acceso 21 de Mayo de 2019.

[13] Intendencia de Montevideo. De aca para allá. http://www.montevideo.gub.uy/institucional/noticias/de-aca-para-alla - Último acceso 21 de Mayo de 2019.

[14] Intendencia de Montevideo. Tu ruta es mi ruta. http://www.montevideo.gub.uy/institucional/noticias/tu-ruta-es-mi-ruta - Último acceso 21 de Mayo de 2019.

[15] Intendencia de Montevideo. En Movimiento. http://www.montevideo.gub.uy/institucional/noticias/en-movimiento - Último acceso 21 de Mayo de 2019.

[16] Scoot Murray. *Interactive Data Visualization for the Web.* O'Reilly, 2013.

[17] Georges G. Grinstein Usama M. Fayyad, Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery.* Morgan Kaufmann, 2002.

[18] Highsoft AS. Highcharts. https://www.highcharts.com - Último acceso 21 de Mayo de 2019.

[19] Massachusetts Institute of Technology. MIT. http://www.mit.edu - Último acceso 21 de Mayo de 2019.

[20] Slant. Best Javascript drawing libraries. https://www.slant.co/topics/28/~best-javascript-drawing-libraries#2 - Último acceso 21 de Mayo de 2019.

[21] Mike Bostock. Protovis. http://mbostock.github.io/protovis/ - Último acceso 21 de Mayo de 2019.